EUROPEAN COMMISSION
DIRECTORATE-GENERAL
INFORMATICS
Directorate B - Digital Business Solutions
**Service oriented Solutions**

DIGIT

# EU Login Mockup Server

## Installation Guide

| | |
|---|---|
| Date: | 05-04-2019 |
| Version: | 6.2.7 |
| Authors: | DIGIT ECAS DEVELOPMENT |
| Revised by: | |
| Approved by: | |
| Public: | |
| Reference Number: | |

Commission européenne, L-2920 Luxembourg. Telephone: (352) 43 01-1.

Commission européenne, B-1049 Bruxelles / Europese Commissie, B-1049 Brussel - Belgium. Telephone: (32-2) 299 11 11.
Office: B-28. Telephone: direct line (32-2) 2987737.

E-mail: DIGIT-ECAS-DEVELOPMENT@ec.europa.eu

## TABLE OF CONTENTS

# Document History

| Ver. | Author | Date | Comment | Modified Pages |
|---|---|---|---|---|
| 1.00 | verfade | 09/10/2009 | Document created. | |
| 1.01 | ackxyyo | 18/03/2010 | Merge with legacy 0.2 version.<br>Remove all references to the old doc.<br>Add link to Forge. | |
| 1.02 | verfade | 19/03/2010 | Changes when revising. | |
| 1.03 | verfade | 30/04/2010 | Changes for release 1.11. | |
| 1.04 | verfade | 11/05/2010 | Changes for release 1.11.1. | |
| 1.05 | verfade | 28/05/2010 | Changed reference ecas.ear to ecas-mockup.ear | |
| 1.06 | verfade | 31/05/2010 | Added documentation on userDataBase.xml | |
| 1.07 | verfade | 05/11/2010 | Changes for release 1.15.0. | |
| 1.08 | verfade | 18/01/2011 | Changes for release 1.16.0. | |
| 1.09 | verfade | 20/01/2011 | Added documentation on reload interval of userDataBase.xml. | |
| 1.10 | verfade | 24/01/2011 | Added clarification on “common name”. | |
| 1.11 | lauredo | 09/02/2011 | Updated Sun links to point to Oracle links.<br>Added information about <trustedCertificates>. | |
| 1.12 | verfade | 17/08/2011 | Changes for release 1.23.0-snapshot-20110817 | |
| 1.13 | peteror | 14/09/2012 | Changes for release 3.1.2 | |
| 1.14 | catizmi | 28/11/2013 | Added screens to section 2.3.2 | |
| 3.6.3 | lauredo | 16/12/2013 | Changes for release 3.6.3 | |
| | | 25/09/2014 | Recommendation to use WLS 10.3.6 | 5 |
| | | 25/09/2014 | Added the "i" and "v" employeeTypes | 14 |
| | | 25/09/2014 | Documented the WebLogic command line | 12 |
| 4.3.0 | catizmi | 15/06/2015 | Changes for release 4.3.0 | 5, 12-14 |
| 4.6.0 | lauredo | 08/02/2016 | Changes for release 4.6.0, general improvements | All |
| | | 24/02/2016 | Fixed incorrect property name | 14-15 |
| 6.2.1 | | 26/02/2019 | Update for EU Login 6.2.1 | |

# Referenced Documents

| Code | Title |
|---|---|
| [CLIENT-BASIC] | ECAS Client Installation and Configuration Guide - Basic.pdf |
| [CLIENT-ADV] | ECAS Client Installation and Configuration Guide - Advanced.pdf |
| [ECAS-FORGE] | http://www.cc.cec/wikis/display/IAM/ECAS+Forge |

Contact:

DIGIT-ECAS-DEVELOPMENT@ec.europa.eu

# 1. INTRODUCTION

This document will help you to install and configure the EU Login Mockup Server. It assumes you have knowledge of the ECAS protocol and clients.

For documentation on the ECAS Clients, please visit [ECAS-FORGE].

## 1.1. What is the mockup?

The mockup is a Web application simulating the behaviour of the production EU Login Server.

This means that the mockup application returns exactly the same responses to ticket validation requests than the production server. It offers the same kind of look-and-feel and user experience than the "real" ECAS Server.

It is a Java EAR file which you can deploy only on a WebLogic Server 12.1.3 and newer.

The mockup is intended for ECAS Client application developers, who do not have access to the "real" ECAS Server or want to perform ECAS-integration tests with total control over user data.

## 1.2. What the mockup doesn't do

- It does not send emails after password request. You need to check the log files for the email body text.
- It does not send SMS messages. You need to check the log files for the SMS message text.
- It does not provide the auditing capabilities of the ECAS Server.
- The mockup is not a secure application and was not designed to scale. The mockup should never be used for any production purpose.

## 1.3. What the mockup does

- It uses the same ECAS protocol as the "real" ECAS Server.
- It authenticates users based upon a simple XML configuration file.
- It validates Service and Proxy Tickets with a strength level set to "*BASIC*"[1]. Other supported strengths are "*CLIENT_CERT*", "*PASSWORD_SMS*", "*PASSWORD_TOKEN*" and "*PASSWORD_TOKEN_CRAM*".
- Unless otherwise specified, the ECAS Mockup Server has all the features from the real ECAS Server of the same version. These features can be tested via the ECAS-DEMO application.

[1] The "real" ECAS Server validation responses return a "STRONG" strength level.

# 2. PREREQUISITES

## 2.1. Downloading the ECAS Mockup Server

You can download the ECAS Mockup Server from the [ECAS-FORGE]. At the time of writing, the release is bundled into *ECAS-MOCKUP-6.2.7.zip*.

Extract the ZIP file in a temporary location of your choice. The *_READ ME_ Bill Of Materials.txt* file gives you a detailed description of the release contents.

## 2.2. Installing the JDK

The ECAS Mockup Server requires an Oracle JDK version 8.

Download the Oracle JDK version 8 from the Java SE Development Kit 8 Downloads page.

No other JDK or version is supported.

In the rest of this document, we will use `%JAVA_HOME%` (or `$JAVA_HOME`) to refer to the JDK installation directory.

## 2.3. Installing the JCE Unlimited Strength Jurisdiction Policy Files

The *Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 8* are needed to unlock stronger cryptography within your Java Runtime Environment, as required by the ECAS base code.

### *2.3.1. Download the ZIP file*

Download the *JCE Unlimited Strength Jurisdiction Policy Files 8* from the dedicated page at the Oracle website.

| Java version | Downloads |
|---|---|
| JDK/JRE 8 | Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 8 |

### *2.3.2. Extract and install the JAR files*

Extract the JAR files from the downloaded ZIP file.

Copy them into the `%JAVA_HOME%\jre\lib\security` (or `$JAVA_HOME/jre/lib/security`) directory and overwrite the existing files.

For example:

for Windows: copy to `D:\Java\jdk1.8.0_202\jre\lib\security`

for Linux: copy to `/usr/lib/jvm/java-8-oracle/jre/lib/security`

## 2.4. Installing Oracle WebLogic Server 12.1.3

The current ECAS Mockup Server only runs on Oracle WebLogic Server 12.1.3. Other versions are not yet supported.

You can download a developer version of this software on the Oracle website.

Please follow the Oracle documentation on how to install a WebLogic Domain and Server instance on your target platform.

During installation, when you have to specify a JDK, make sure you select the JDK installed in section 2.2.

In the rest of this document, we will use `%DOMAIN_HOME%` (or `$DOMAIN_HOME`) to refer to the path of the WebLogic Server domain directory created in this step.

## 2.5. Adding properties to the WebLogic Server startup line

The ECAS Mockup Server needs to run on a WebLogic Server instance launched with specific settings[2].

Please refer to the Oracle documentation on how to configure startup parameters for a WebLogic Server instance on your target platform.

### *2.5.1. General properties*

Add the following Java System properties to the WebLogic Server process startup line:

```
-Dsun.net.inetaddr.ttl=60
-Dsun.net.inetaddr.negative.ttl=5
-DUseSunHttpHandler=true
-Dsun.net.client.defaultConnectTimeout=60000
-Dsun.net.client.defaultReadTimeout=60000
-Dweblogic.ssl.JSSEEnabled=true
-Dweblogic.security.SSL.enableJSSE=true
```

For more information about each entry, please consult the related Java documentation.

### *2.5.2. Specific properties*

Regarding the following System properties, please consult the Java documentation and see whether they are applicable for your target platform.

```
-Djava.awt.headless=true
-Djava.security.egd=file:/dev/urandom
-Dweblogic.MuxerClass=weblogic.socket.PosixSocketMuxer
```

Add any relevant property to the WebLogic Server process startup line.

## 2.6. Installing the ECAS Client

In the ECAS Mockup Server release, look for a file called *ecas-weblogic-12.2.1-authprovider-4.32.0.jar*: it is the ECAS Client needed for the WebLogic instance where you will install the Mockup Server.

> We strongly recommend using the ECAS client available in the ECAS Mockup Server release.
>
> If, for any reason, you need a different one, make sure you install the real ECAS client, not the Mock Client.

The installation procedure is explained in [CLIENT-BASIC]. Note that some steps may have already been performed previously, such as enabling Java strong cryptography.

[2] One way to do it is by editing `setDomainEnv.cmd` (or `.sh`) and adding properties into `JAVA_OPTIONS`.

## 2.7. Creating an SSL certificate for WebLogic Server

You need to create an SSL certificate to be able to run WebLogic Server in SSL/TLS.

### *2.7.1. Generate an SSL certificate for your server*

To do this, you may use the `keytool` command, which is included in the JDK:

```
"%JAVA_HOME%/bin/keytool" -genkey -v -alias myServerAlias -keyalg RSA -keysize 2048 -
validity 3652 -keystore myServerKeystore.jks -storepass myKeystorePassword
What is your first and last name?
  [Unknown]:  <SERVERNAME>
What is the name of your organizational unit?
  [Unknown]:
What is the name of your organization?
  [Unknown]:
What is the name of your City or Locality?
  [Unknown]:
What is the name of your State or Province?
  [Unknown]:
What is the two-letter country code for this unit?
  [Unknown]:
Is CN=<SERVERNAME>, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown correct?
  [no]:  yes

Generating 2.048 bit RSA key pair and self-signed certificate (SHA256withRSA) with a
validity of 3.652 days
        for: CN=<SERVERNAME>, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
Enter key password for <myServerCert>
        (RETURN if same as keystore password):
[Storing myServerKeystore.jks]
```

Where:

- `-alias` *`myServerAlias`*: specifies the alias you want to give to your private key inside the keystore;
- `-keysize` *`2048`*: specifies the size of the key in bits;
- `-keystore` *`myServerKeystore.jks`*: specifies the name of the keystore file on the disk;
- `-storepass` *`myKeystorePassword`*: specifies the password for the keystore (and also for the key if you pressed 'Enter' for the last question above);
- `-<SERVERNAME>` is the CN (= Common Name) and represents the hostname of your server as is going to be used in the URL to access it from other Web applications.

In the example above, the key is stored in the keystore called *myServerKeystore.jks* and uses the alias *myServerAlias*.

At this point it can be used as a self-signed certificate and this means your SSL client applications will have to import this certificate in their own Java trustStore in order to trust you.

If you need to be trusted by other Commission applications, you must make a Certificate Signing Request (CSR) to the "Security Directorate" (HR DS 5) for them to sign it.

To verify the validity of your keystore you can also use the `keytool` command:

```
"%JAVA_HOME%/bin/keytool" -list -v -keystore myServerKeystore.jks
Enter keystore password: myKeystorePassword

Keystore type: JKS
Keystore provider: SUN

Your keystore contains 1 entry

Alias name: myserveralias
Creation date: 26-mrt-2010
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=<SERVERNAME>, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
Issuer: CN=<SERVERNAME>, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
```

```
Serial number: 4bace96e
Valid from: Fri Mar 26 18:05:50 CET 2015 until: Wed Mar 25 18:05:50 CET 2025
Certificate fingerprints:
         MD5:  C1:43:D4:41:A4:F2:39:0F:8E:45:31:51:03:CB:7D:BF
         SHA1: 19:7A:CE:EB:B8:80:69:B0:FB:33:FC:43:70:48:EE:23:BD:F1:99:B5
         Signature algorithm name: SHA256withRSA
         Version: 3


*******************************************
*******************************************
```

More information on `keytool` is available at
https://docs.oracle.com/javase/8/docs/technotes/tools/unix/keytool.html

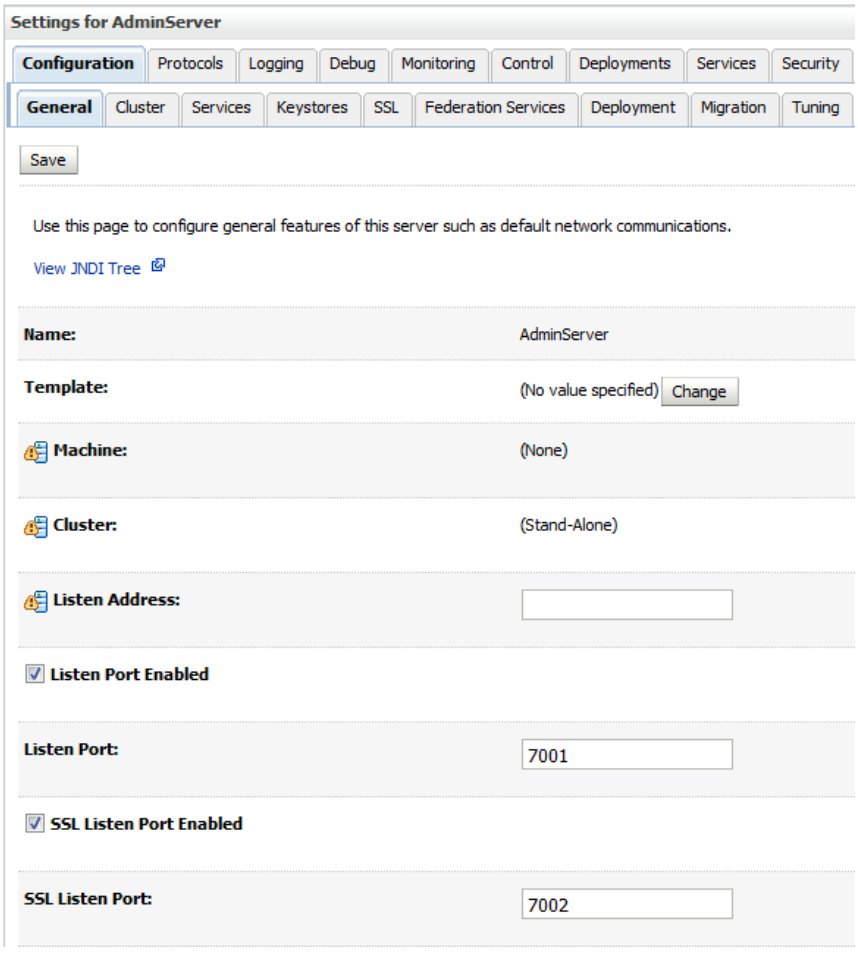
There are several other options to generate your own SSL certificate such as using graphical tools (for instance, Portecle (http://portecle.sourceforge.net/)) or OpenSSL.

### *2.7.2. Configure WebLogic Server to use the generated private key and certificate*

Install your new keystore, containing your private key and its associated certificate, using the WebLogic Administration Console.

In addition to the following instructions, please refer to the WebLogic documentation[3] for more information.

1. Connect to the WebLogic Administration Console and locate your server instance.
2. In the **Configuration** > **General** tab, enable the SSL port (usually 7002) as follows:

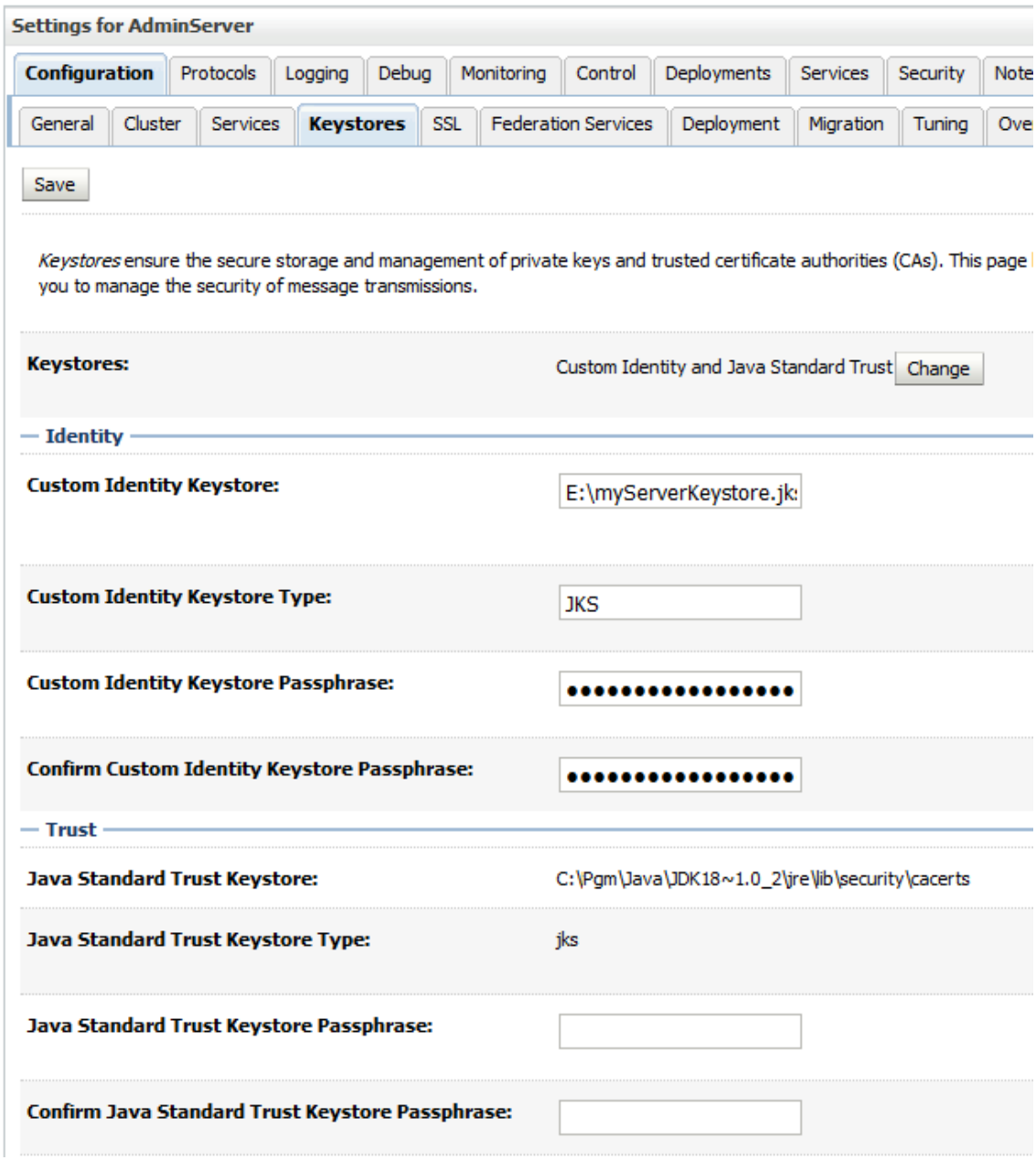



[3] http://docs.oracle.com/middleware/1213/wls/TASKS/servers.htm#TASKS322

3. In the **Keystores** tab, configure the *Identity Keystore* to point to the keystore you generated in the previous section.

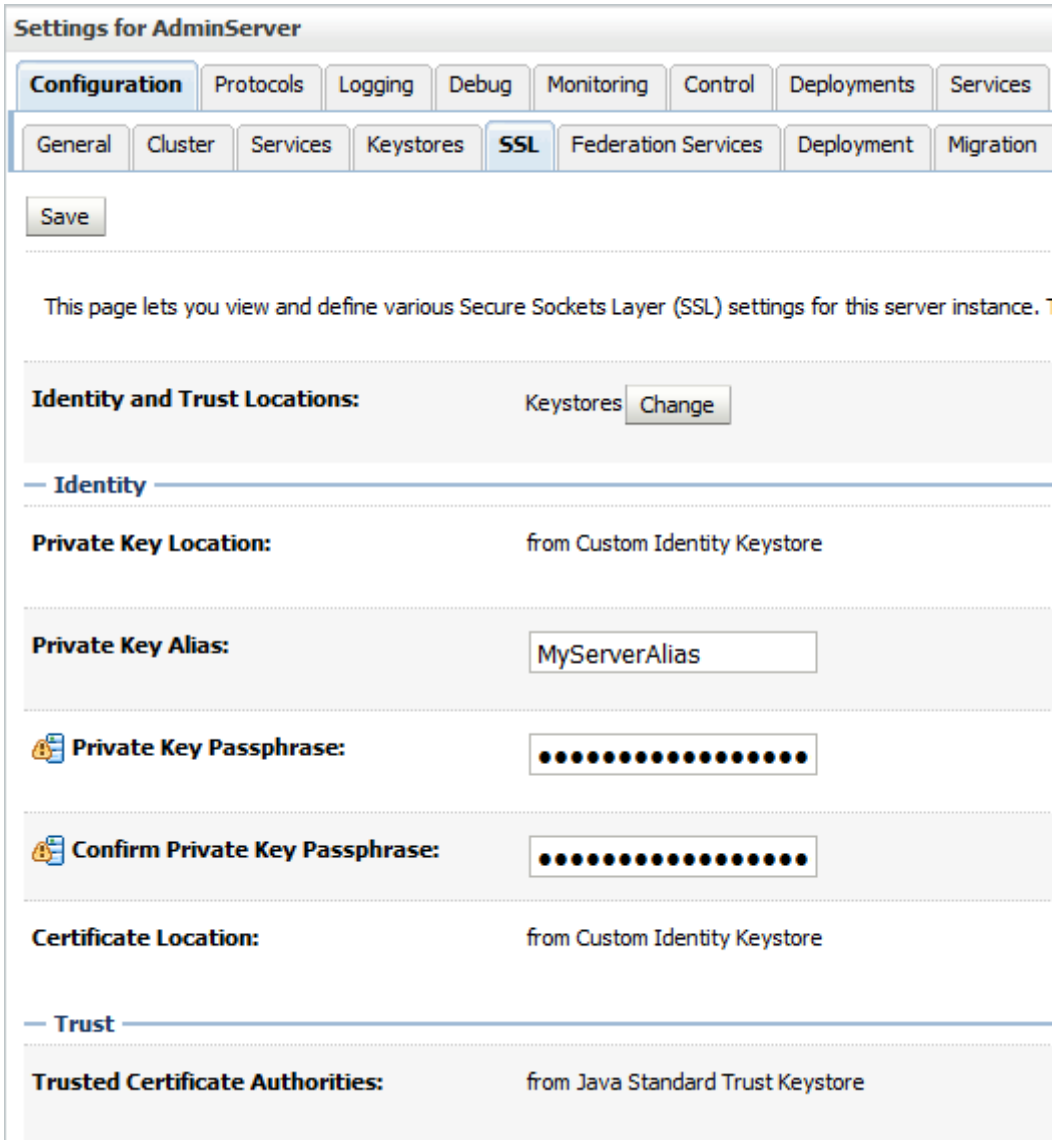
   Specifically, the **Keystores** field is where you define the method for storing and managing private keys/digital certificate pairs and trusted CA certificates. We recommend selecting the "`Custom Identity and Java Standard Trust`" option, where:

   - the "Custom Identity" would be the keystore you just created;
   - the "Java Standard Trust" automatically points to the `cacerts` file[4] in the `%JAVA_HOME%\jre\lib\security` (or `$JAVA_HOME/jre/lib/security`) directory.

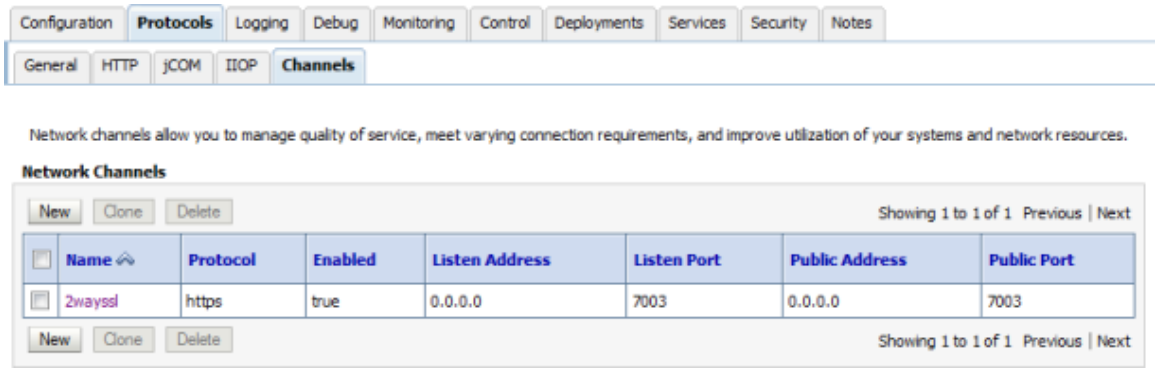



[4] The default keystore password for `cacerts` is "`changeit`" (without quotes).

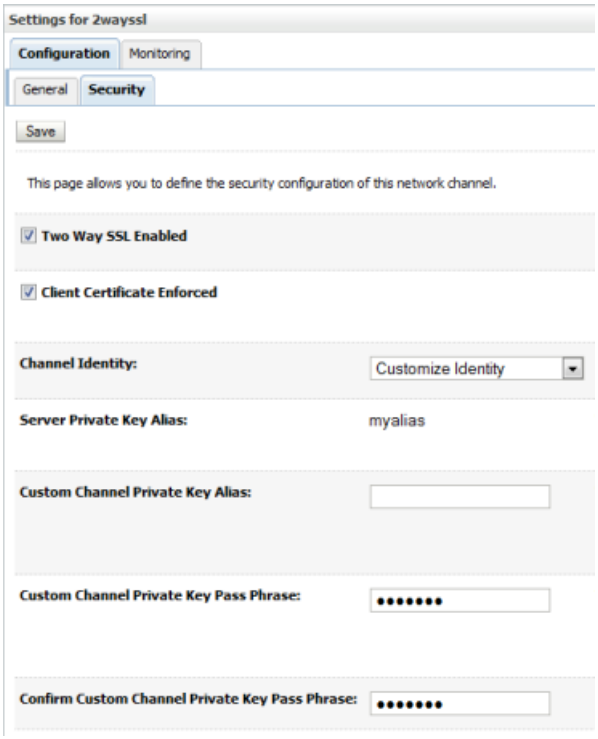
4. In the **SSL** tab, specify the alias and password of the private key generated in the previous section:




5. On top of that, if you want to use 2-way SSL to simulate the CLIENT_CERT strength, you need to create an additional NetworkChannel, in the **Protocols** > **Channels** tab:



You have to setup this channel with client certificates as *required* and you can use the same private key and certificate as before (for instance, activate the 7003 port for 2-way SSL).




## 2.8. Trusting the server SSL certificate

**Each client application** that needs to use your ECAS Mockup Server will need to import your SSL certificate into its Java trustStore (the JRE `cacerts` file).

The **ECAS Mockup Server <u>also</u>** needs to trust the SSL certificate, so you also need to import the SSL certificate into the trustStore of the JRE on which your WebLogic Server instance runs.

### *2.8.1. <u>Export your public SSL certificate</u>*

```
"%JAVA_HOME%/bin/keytool" -export -v -rfc -alias myServerAlias -file
myServerCert.cer -keystore myServerKeystore.jks
Enter keystore password:  myKeystorePassword
Certificate stored in file <myServerCert.cer>
```

This will store your public certificate into the file *myServerCert.cer*.

Note the -rfc option which tells keytool to use the base64 flat file format. The resulting file can then be opened or sent as plain text.

For example, a resulting file would look like:

```
-----BEGIN CERTIFICATE-----
MIIETTCCAzWgAwIBAgIBATANBgkqhkiG9w0BAQUFADB2MQswCQYDVQQGEwJCRTERMA8GA1UECBMI
QnJ1c3NlbHMxETAPBgNVBAcTCEJydXNzZWxzMQwwCgYDVQQKEwNDRUMxDjAMBgNVBAsTBURJR0lU
MSMwIQYDVQQDExpFQ0FTIENlcnRpZmljYXRlIEF1dGhvcml0eTAeFw0wNjA0MTExNzUzMzZaFw0x
MTA0MTIxNzUzMzZaMHYxCzAJBgNVBAYTAkJFMREwDwYDVQQIEwhCcnVzc2VsczERMA8GA1UEBxMI
QnJ1c3NlbHMxDDAKBgNVBAoTA0NFQzEOMAwGA1UECxMFRElHSVQxIzAhBgNVBAMTGkVDQVMgQ2Vy
dGlmaWNhdGUgQXV0aG9yaXR5MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAotKTdKad
m5tzmr59Tofq25hcY1jfhxlrcL5NnfsLc6j+nvdUdDdN2soJbbOBCMcc8BeXW3b4ib+1WYmrE8B0
wx/6dNonNf/4NFTYm+Xf80O81lsWO/E6279SBTvFw5bTfaPme+YLvpLiUa65XYpLkQK/LF4EEL7u
Q8Di2nlpSqug27Dkp5Z3Jeh76aHvro1Ul+0qGEHyiJuYT7DSQSjc8XEXPwLpWdrc8izy0e6OsZIW
jnrdqsBcIqylNriz7kC1Woft77aumgwiMVYMG2yCno4AIIrceI4j/VWYFMi60cX3zXGSfvLGA/uF
gVoyDze2FKMskBBCT72G/XZ5NvDVcQIDAQABo4HlMIHiMBEGCWCGSAGG+EIBAQQEAwIA9zAOBgNV
HQ8BAf8EBAMCAf4wHQYDVR0OBBYEFC4ZiCMBXEEqyHrAdhhn9nbAtv6gMB8GA1UdIwQYMBaAFC4Z
iCMBXEEqyHrAdhhn9nbAtv6gMBIGA1UdEwEB/wQIMAYBAf8CAQEwaQYDVR0lBGIwYAYEVR0lAAYI
KwYBBQUHAwEGCCsGAQUFBwMCBggrBgEFBQcDAwYIKwYBBQUHAwQGCCsGAQUFBwMFBggrBgEFBQcD
BgYIKwYBBQUHAwcGCCsGAQUFBwMIBggrBgEFBQcDCTANBgkqhkiG9w0BAQUFAAOCAQEAWtsqmsus
```

```
+8U4p/qZTxTGnmAm07ekim68nYJpJpif8Rg5OOBPv9RXfQABIrKHHkxyiZiG46xuHGbwNPVeXpZF
PeYQfrKC1YcuuWXUOwDUNaKCSVg7CLLUE6Io0j8aEGK8/c2rEIcmlHWb6NSxOixgsdEoHFEyK2I/
hL76/Bp9hAeuYFZTS8lwIU87ZUFGGSBZqfxn1f4NNFXjwjxn/sRGdnh5OJvu+0o1047haOmCbY1X
Xio0JRekMDvoq7G30E+WsdTivkvrZoyjiX5GZ1HPRK0JFrtp+KJwawx2uG2hC0RweUo9iTbiWch3
8axT7TmJf5/6Dqn43ro7STpwLpJOrw==
-----END CERTIFICATE-----
```

### *2.8.2. Configure all client applications to accept your SSL certificate*

Each client application that wants to use your ECAS Mockup Server will need to trust your SSL certificate.

If they don't trust your certificate, they will get errors during the SSL handshake and will never accept to open connections to your Mockup Server.

#### *2.8.2.1. Configure your mockup certificate as a trusted certificate in all ECAS client configuration files*

If the client application is not a Java application using the ECAS client 1.16.1, skip this section and proceed with the next one.

On the contrary, if the client application is a Java application using the ECAS client 1.16.1 or beyond, you can add your certificate directly into the ECAS client configuration file and don't need to modify the Java trustStore if you don't want to.

Refer to [CLIENT-ADV] section *3.3.42. trustedCertificates* for details.

If we take the sample certificate obtained above as example, you would update the ECAS client configuration XML file with the following snippet:

```
<trustedCertificates>
       <trustedCertificate>
MIIETTCCAzWgAwIBAgIBATANBgkqhkiG9w0BAQUFADB2MQswCQYDVQQGEwJCRTERMA8GA1UECBMI
QnJ1c3NlbHMxETAPBgNVBAcTCEJydXNzZWxzMQwwCgYDVQQKEwNDRUMxDjAMBgNVBAsTBURJR0lU
MSMwIQYDVQQDExpFQ0FTIENlcnRpZmljYXRlIEF1dGhvcml0eTAeFw0wNjA0MTExNzUzMzZaFw0x
MTA0MTIxNzUzMzZaMHYxCzAJBgNVBAYTAkJFMREwDwYDVQQIEwhCcnVzc2VsczERMA8GA1UEBxMI
QnJ1c3NlbHMxDDAKBgNVBAoTA0NFQzEOMAwGA1UECxMFRElHSVQxIzAhBgNVBAMTGkVDQVMgQ2Vy
dGlmaWNhdGUgQXV0aG9yaXR5MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAotKTdKad
m5tzmr59Tofq25hcY1jfhxlrcL5NnfsLc6j+nvdUdDdN2soJbbOBCMcc8BeXW3b4ib+1WYmrE8B0
wx/6dNonNf/4NFTYm+Xf80O81lsWO/E6279SBTvFw5bTfaPme+YLvpLiUa65XYpLkQK/LF4EEL7u
Q8Di2nlpSqug27Dkp5Z3Jeh76aHvro1Ul+0qGEHyiJuYT7DSQSjc8XEXPwLpWdrc8izy0e6OsZIW
jnrdqsBcIqylNriz7kC1Woft77aumgwiMVYMG2yCno4AIIrceI4j/VWYFMi60cX3zXGSfvLGA/uF
gVoyDze2FKMskBBCT72G/XZ5NvDVcQIDAQABo4HlMIHiMBEGCWCGSAGG+EIBAQQEAwIA9zAOBgNV
HQ8BAf8EBAMCAf4wHQYDVR0OBBYEFC4ZiCMBXEEqyHrAdhhn9nbAtv6gMB8GA1UdIwQYMBaAFC4Z
iCMBXEEqyHrAdhhn9nbAtv6gMBIGA1UdEwEB/wQIMAYBAf8CAQEwaQYDVR0lBGIwYAYEVR0lAAYI
KwYBBQUHAwEGCCsGAQUFBwMCBggrBgEFBQcDAwYIKwYBBQUHAwQGCCsGAQUFBwMFBggrBgEFBQcD
BgYIKwYBBQUHAwcGCCsGAQUFBwMIBggrBgEFBQcDCTANBgkqhkiG9w0BAQUFAAOCAQEAWtsqmsus
+8U4p/qZTxTGnmAm07ekim68nYJpJpif8Rg5OOBPv9RXfQABIrKHHkxyiZiG46xuHGbwNPVeXpZF
PeYQfrKC1YcuuWXUOwDUNaKCSVg7CLLUE6Io0j8aEGK8/c2rEIcmlHWb6NSxOixgsdEoHFEyK2I/
hL76/Bp9hAeuYFZTS8lwIU87ZUFGGSBZqfxn1f4NNFXjwjxn/sRGdnh5OJvu+0o1047haOmCbY1X
Xio0JRekMDvoq7G30E+WsdTivkvrZoyjiX5GZ1HPRK0JFrtp+KJwawx2uG2hC0RweUo9iTbiWch3
8axT7TmJf5/6Dqn43ro7STpwLpJOrw==
       </trustedCertificate>
<trustedCertificates>
```

#### *2.8.2.2. Import your SSL certificate into the Java trustStores of all client applications*

The second option consists in importing your new SSL certificate into the Java trustStore of each client application. You should skip this step if you applied the configuration explained in the previous section.

Each client application that needs to use your ECAS Mockup Server will need to import your SSL certificate into its Java trustStore. The default Java trustStore of the JVM is located at `%JAVA_HOME%/jre/lib/security/cacerts` and its default password is "`changeit`".

To import your certificate, use the following command:

```
"%JAVA_HOME%/bin/keytool" -import -v -alias ecasServerMockupCert -file
myServerCert.cer -keystore "%JAVA_HOME%\jre\lib\security\cacerts"
Enter keystore password:  changeit
Owner: CN=<SERVERNAME>, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
Issuer: CN=<SERVERNAME>, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
Serial number: 4bace96e
Valid from: Fri Mar 26 18:05:50 CET 2015 until: Wed Mar 25 18:05:50 CET 2025
Certificate fingerprints:
         MD5:  C1:43:D4:41:A4:F2:39:0F:8E:45:31:51:03:CB:7D:BF
         SHA1: 19:7A:CE:EB:B8:80:69:B0:FB:33:FC:43:70:48:EE:23:BD:F1:99:B5
         Signature algorithm name: SHA256withRSA
         Version: 3
Trust this certificate? [no]:  yes
Certificate was added to keystore
[Storing D:\Oracle\jdk1.8.0_72\jre\lib\security\cacerts]
```

**Note:**

Be sure that you imported your Server certificate into the JDK that is actually used by your server. If you don't import the certificate in the right place, it will be impossible to open SSL connections because of trust failures in SSL handshakes.

#### *2.8.2.3. Trust your Mockup SSL certificate for non-Java based technologies*

If the client application is not based on Java, you have to rely on the documentation provided for your platform on how to trust additional SSL certificates. The steps to follow should be very similar to the sections above and usually involve either copying the base64 certificate string in some configuration file or importing the certificate file itself using some user interface or command line utility.

## 3. INSTALLING ECAS MOCKUP SERVER CONFIGURATION FILES

1. Create a directory (e.g. *classes*) in your WebLogic Server's classpath.

For example:

- Create directory:

  e.g. for Windows: `%DOMAIN_HOME%\classes`

  e.g. for Linux: `$DOMAIN_HOME/classes`

- Add the directory to the server's classpath:

  e.g. for Windows, edit `%DOMAIN_HOME%\bin\setDomainEnv.cmd`
  append `set CLASSPATH=%CLASSPATH%;%DOMAIN_HOME%\classes`

  e.g. for Linux, edit `$DOMAIN_HOME/bin/setDomainEnv.sh`
  append `CLASSPATH="${CLASSPATH}${CLASSPATHSEP}${DOMAIN_HOME}/classes"`
  `export CLASSPATH`

2. Unzip the *classes.zip* file from the ECAS Mockup Server release into this new directory.

## 4. CONFIGURING THE ECAS MOCKUP SERVER

Several properties in different files need to be adapted to your environment. All files are found in the directory you created in the previous chapter (e.g. *classes*).

### 4.1. Edit `ecas.properties`

**`ecas.hostname`**=<SERVERNAME>

➔ add your own server name

e.g. `ecasmockup.mycompany.com`

**`ecas.port.HTTP`**=<PORT.HTTP>

➔ add your own HTTP PORT (WebLogic defaults to 7001)

e.g. `8080`

**`ecas.port.HTTPS`**=<PORT.HTTPS>

➔ add your own HTTPS (SSL) PORT

e.g. `8443`

**`ecas.port.HTTPS.2waySSL`**=<PORT.HTTPS.2WAYSSL>

➔ add your own 2-way SSL PORT

e.g. `8444`

**`antiDenialOfService.DEFAULT.bypass.list`**=127.0.0.1,<SERVERNAME>

➔ add your server name and IP addresses

e.g. `127.0.0.1,ecasmockup.mycompany.com,192.168.0.42`

**`antiDenialOfService.COSTLY.bypass.list`**=127.0.0.1,<SERVERNAME>

➔ add your server name and IP addresses

e.g. `127.0.0.1,ecasmockup.mycompany.com,192.168.0.42`

**`antiDenialOfService.BACKEND.bypass.list`**=127.0.0.1,<SERVERNAME>

➔ add your server name and IP addresses

e.g. `127.0.0.1,ecasmockup.mycompany.com,192.168.0.42`

**`antiDenialOfService.FRONTEND.bypass.list`**=127.0.0.1,<SERVERNAME>

➔ add your server name and IP addresses

e.g. `127.0.0.1,ecasmockup.mycompany.com,192.168.0.42`

**`crl.url`**=https://<SERVERNAME>:<PORT.HTTPS>/cas/ca/crl

➔ add your server name and HTTPS (SSL) PORT

e.g. `https://ecasmockup.mycompany.com:8443/cas/ca/crl`

### 4.2. Edit `ecas-application-settings.properties`

**`network.ipSubnets.internal`**=127.0.0.0/8,::1/128

➔ add your own IP subnets delimited by commas

e.g. for IPv4 `127.0.0.0/8,172.16.0.0/12,192.168.0.0/16,169.254.0.0/16`

e.g. for IPv6 `::1/128,2001:0DB8:1234::/48`

**`ecasServer.allowedHostsAndPorts`**=<SERVERNAME>:<PORT.HTTPS>

➔ add your own server name and HTTPS (SSL) PORT

e.g. `ecasmockup.mycompany.com:8443`

### 4.3. Edit `ecas-client-configuration.properties`

```
eu.cec.digit.ecas.client.filter.ecasBaseUrl=https://<SERVERNAME>:<PORT.HTTPS>
```

➔ add your own server name and HTTPS (SSL) PORT (same as for *allowedHosts*)

e.g. `https://ecasmockup.mycompany.com:8443`

### 4.4. Edit `ecas-config-ecas-demo.properties`

We recommend testing the installation with the ECAS-DEMO application. For this, you need to change the following properties in *ecas-config-ecas-demo.properties*.

```
edu.yale.its.tp.cas.client.filter.serverName=<SERVERNAME>
```

➔ add your own server name (same as for *allowedHosts*)

e.g. `ecasmockup.mycompany.com`

```
eu.cec.digit.ecas.client.filter.ecasBaseUrl=https://<SERVERNAME>:<PORT.HTTPS>
```

➔ add your own server name and HTTPS (SSL) PORT (same as for *allowedHosts*)

e.g. `https://ecasmockup.mycompany.com:8443`

## 5. DEFINING MOCK USERS

In the directory from chapter 3, the *userDataBase.xml* file contains several fake users for your Mockup Server, some with a minimal set of information and others with several attributes.

You may alter this file to define users according to your needs.

Note that, while the ECAS Mockup Server is up:

- changes to this file are taken into account immediately, no need to restart the server;
- the Mockup Server will write its own properties to this file.

An example file *exampleUserDataBase.xml* with all properties can be found in *ecas-mockup-6.2.7.ear*, within *cas.jar*, under *WEB-INF\classes\eu\europa\ec\ecas\mockup\data\binding*.

The minimal user configuration is:

```
<user>
    <email>minimal@email.eu</email>
    <firstName>Minimal_first_name</firstName>
    <lastName>MINIMAL_LAST_NAME</lastName>
    <manager>false</manager>
    <objectStatus>a</objectStatus>
    <organisation>eu.europa.ec</organisation>
    <organisationUsername>minimal</organisationUsername>
    <strongPasswordHash>Abc123</strongPasswordHash>
    <uid>minimal</uid>
</user>
```

The *email*-tag is the user's email address. Can be used to log in when the organisation is *external*.

The *manager*-tag denotes if the user is a manager (true/false).

The *objectStatus*-tag must be "a" (active); "d" stands for disabled.

The *organisation*-tag is the domain. A domain represents one population of users. Examples of valid values are:

- `eu.europa.ec` for European Commission

- `external` for External
- `eu.europa.curia` for Court of Justice of the European Union

The *organisationUsername*-tag must be unique within the domain. This is the username which you have to enter to log in. It is sometimes called *moniker*.

The *strongPasswordHash*-tag is the password to log in. You put it in plain text and it will be transformed into a hash after logging in. The hash depends on the uid so when you change the uid, you need to re-put the password in plain text.

The *uid*-tag must be unique within the whole user database. This is not used for logging in.

The optional *employeeType*-tag denotes the employee type. Example valid values are:

- f for Commission officials
- x for intra-muros contractors (working for the Commission)
- e for generic users and extra-muros contractors (working for the Commission)
- n for self-registered users (external users)
- i for interinstitutional users (other European Institutions, Agencies or Bodies)
- v for virtual users (users automatically created from eID authentications)

The optional *cudGroups*-tag contains the “application group” the user belongs to. Multiple groups are specified with multiple tags. The ECAS Client configuration can request these groups with the *groups* property in the ECAS Client configuration file.

## 5.1. Migration of previous versions of *userDataBase.xml*

### *5.1.1. Migration to version 4.3.0*

With the ECAS Mockup server release 4.3.0, the embedded ECAS service DB comes with flags for new authentication strengths, all enabled by default. To benefit from this update, it is recommended to clean up the existing ECAS service DB as described in section 5.2.

Moreover, in this release, users are entitled to have a Challenge-Response Authentication Mechanism ("CRAM") hardware token. In the userDataBase.xml from the release you'll find two users having such kind of token; if you wish to "upgrade" your own userDataBase.xml, you need to add to the desired user a *tokenCramId*-tag with a numeric value.

Example: <tokenCramId>1234</tokenCramId>

### *5.1.2. Migration to version 3.1*

With the ECAS Mockup server release 3.1, the storage format for mobile phones has changed. A mobile phone is now stored in the format <SOURCE>:<NUMBER>, where <SOURCE> is either "COMREF" or "ECAS". Any of them can be used; this is of no importance for the mockup.

Example: <mobilePhone>COMREF:+3212345</mobilePhone>

## 5.2. Clean up the embedded ECAS service DB

To clean up the embedded ECAS DB, delete the folder `%DOMAIN_HOME%/servicesDB`.

# 6. DEPLOYING THE ECAS MOCKUP SERVER EAR

Deploy the *ecas-mockup-6.2.7.ear* from the Mockup Server distribution into your WebLogic Server.

For example, if your WebLogic instance is running in development mode, you may copy the ear file into the *autodeploy* directory.

## 7. DEPLOYING AND TESTING WITH ECAS-DEMO

Deploy the *ecas-demo.ear* from the Mockup Server distribution into your WebLogic Server.

For example, if your WebLogic instance is running in development mode, you may copy the ear file into the *autodeploy* directory.

Open `http://<SERVERNAME>:<PORT.HTTP>/ecas-demo/` in your browser to start.

**Note:** It is possible that the first access to the ECAS Mockup Server is very slow due to the JSP compilation. Please be patient and do not close the browser too quickly.

## 8. DEPLOYING THE MOCKUP-MOBILE EAR

This mockup application simulates the behaviour of the EU Login Mobile App.

Deploy the *mockup-mobile.ear* from the Mockup Server distribution into your WebLogic Server.

For example, if your WebLogic instance is running in development mode, you may copy the ear file into the *autodeploy* directory.

The *mockup-mobile.ear* application is only needed in you intend to perform tests with the MOBILE_APP, PASSWORD_MOBILE_APP or PASSWORD_SOFTWARE_TOKEN authentication strengths.

## 9. DEPLOYING THE MOCKUP-OAUTH-IDP EAR

This mockup application simulates the behaviour of federation with social networks authentication provider such as Facebook, Goolge or Twitter.

Deploy the *mockup-oauth-idp.ear* from the Mockup Server distribution into your WebLogic Server.

For example, if your WebLogic instance is running in development mode, you may copy the ear file into the *autodeploy* directory.

The *mockup-oauth-idp.ear* application is only needed in you intend to perform tests with the SOCIAL_NETWORKS authentication strength.

## 10. DEPLOYING THE MOCKUP-PEPS EAR

This mockup application simulates the authentication with an eID.

Deploy the *mockup-peps.ear* from the Mockup Server distribution into your WebLogic Server.

For example, if your WebLogic instance is running in development mode, you may copy the ear file into the *autodeploy* directory.

The *mockup-peps.ear* application is only needed in you intend to perform tests with the STORK authentication strength (eID).

## 11. CLIENT APPLICATION(S) CONFIGURATION

The EU Login Mockup Server only accepts the BASIC, MOBILE_APP, PASSWORD_MOBILE_APP, PASSWORD_SOFTWARE_TOKEN, PASSWORD_SMS, PASSWORD_TOKEN, PASSWORD_TOKEN_CRAM, STORK, SOCIAL_NETWORKS and CLIENT_CERT strengths. This means that all client applications need to change or put the following property in their ECAS Client configuration file:

```
eu.cec.digit.ecas.client.filter.acceptStrengths=BASIC,MOBILE_APP,PASSWORD_MOBILE_APP,PASSWORD_SOFTWARE_TOKEN,PASSWORD_SMS,PASSWORD_TOKEN,PASSWORD_TOKEN_CRAM,STORK,SOCIAL_NETWORKS,CLIENT_CERT
```