



IST-2001-34340  
WP5 - D5.3 - Tests  
Documentation

Distribution List :	Project Partners
Author :	<b>Eduard Tric , Axetel</b>
Authorised by :	
Date of Issue :	23 April 2004
Issue :	0.2
File Name :	EuPKI_WP5_D5.3.doc
Work Package :	WP5
Deliverable Number :	D5.3
Deliverable Type :	Public
Deliverable Nature :	Deliverable
Total Number of Pages :	21
Contact Details for EuPKI :	eupki (at) axetel (dot) com

## 0 Table of Contents

0	Table of Contents.....	2
1	Document Control .....	3
1.1	Abstract .....	3
1.2	Keywords .....	3
2	Development test Organization.....	4
2.1.1	Requirements.....	4
2.2	Tested modules location .....	4
2.2.1	CA and CSP modules : .....	4
2.2.2	RA modules : .....	5
2.2.3	Key Ceremony modules : .....	5
2.2.4	Key Generation System modules : .....	5
2.2.5	Common and general crypto modules : .....	5
2.2.6	Crypto modules.....	6
3	Detailed list of tested classes .....	12
4	Additional tests .....	18
4.1.1	RA-CA communication test .....	18

# 1 Document Control

<i>Issue</i>	<i>Date of Issue</i>	<i>Comments</i>
0.1	2 november 2003	First draft version
0.2	3 march 2004	Update

## 1.1 Abstract

This document describes the unit tests related to the development.

This document must be used with the wp5 d5.1 document, containing the development documentation.

The unitary tests are strictly related to their modules. The integrator (wp6) and the demonstrator (wp7) may decide to make their own test (inter-modules , communication , scalability ).

## 1.2 Keywords

EuPKI, Development, Workflow, Module, Implementation ...

## 2 Development test Organization

### 2.1.1 Requirements

Many test classes in the project use JUnitEJB. They can be spotted by seeing if they inherit from EJBTestCase. JUnitEJB can be used to run tests remotely on the JBoss server's JVM, but the code to be run must be available on the server side (in the server classpath, most likely the deploy directory). By running in the server VM, these test classes are able to access some Enterprise Beans using their local interfaces (mainly entity beans). JUnitEJB uses a session bean (EJBTestRunner) to do this remote running, simply put EJBTestCase.run tells EJBTestRunner to load the class we want to run and run each test and also the setUp/tearDown methods. More documentation can be found in the sources of junitejb (<http://sourceforge.net/projects/junitejb>).

JUnitEJB can be thought of as an implementation of the EJB Command pattern (see "EJB Design Patterns" by Floyd Marinescu from The Middleware Company - <http://www.middleware-company.com>).

The output of these tests can be seen in the JBoss's console, since they will be running in the server's JVM and not as local applications. Very often the running of the tests can show success in the client console while in the server console we have a very long stack trace after an exception was thrown. On the client side errors should be propagated (the run method from EJBTestCase is supposed to throw an exception when something goes wrong) but this happens very seldom, probably this propagation of exceptions doesn't work that well.

All tests are run after deployment of the application jar/ear files, that is after the ./build.sh deploy step.

## 2.2 Tested modules location

The docs for every module contain a section concerning the related tests. The tests are located as follows:

### 2.2.1 CA and CSP modules :

<http://projects.axetel.com/cgi-bin/cvsweb.cgi/eupki/ca/src/docs/>  
<http://projects.axetel.com/cgi-bin/cvsweb.cgi/eupki/ca/src/main/org/eupki/ca/test/>

All tests can be run using ./build.sh tests from the root of the project.

We recommend that you run the db-tests (ca-setup/ca-test/ca-teardown) and the comm test separately.

### **2.2.2 RA modules :**

<http://projects.axetel.com/cgi-bin/cvsweb.cgi/eupki/ra/src/docs/>  
<http://projects.axetel.com/cgi-bin/cvsweb.cgi/eupki/ra/src/main/org/eupki/ra/test/>

### **2.2.3 Key Ceremony modules :**

<http://projects.axetel.com/cgi-bin/cvsweb.cgi/eupki/kc/src/docs/>  
<http://projects.axetel.com/cgi-bin/cvsweb.cgi/eupki/kc/src/main/org/eupki/kc/test/>

Command line for KC tests

Steps:

1. cd {eupkiProject\_home}/kc
2. Build the sources (for full test, a CA has to be already installed)
3. ./build.sh kc-test

### **2.2.4 Key Generation System modules :**

<http://projects.axetel.com/cgi-bin/cvsweb.cgi/eupki/kgs/src/docs/>  
<http://projects.axetel.com/cgi-bin/cvsweb.cgi/eupki/kgs/src/main/org/eupki/kgs/test/>  
<http://projects.axetel.com/cgi-bin/cvsweb.cgi/eupki/usrkgs/src/main/org/eupki/usrkgs/tests>

Command line for KGS tests

Steps:

1. Build the sources, deploy the KGS sub system and start JBoss
2. cd {eupkiProject\_home}/kgs
3. ./build.sh kgs-test

### **2.2.5 Common and general crypto modules :**

<http://projects.axetel.com/cgi-bin/cvsweb.cgi/eupki/common/src/docs/>  
<http://projects.axetel.com/cgi-bin/cvsweb.cgi/eupki/common/src/main/org/eupki/common/test/>

### 2.2.6 *Crypto modules*

<http://projects.axetel.com/cgi-bin/cvsweb.cgi/eupki/crypto/src/main/org/eupki/crypto/test/>  
<http://projects.axetel.com/cgi-bin/cvsweb.cgi/eupki/crypto/src/docs/>

Steps (set of hardcoded tests):

1. Build the sources
2. cd {eupkiProject\_home}/crypto
3. ./build.sh crypto-test

The tests:

The first set of tests is trying to establish the functionalities and the limits of crypto provider. This ensures the system has the expected behaviour. The input data are hardcoded in order to verify if the underlying module (classes from project, installed libraries) are doing their job.

1. Test 0: What providers do we have? It prints the list of accepted installed cryptographic providers. No hardcoded data are used.
2. Test 1: It tests the translation into/from Base64 encoding form. The test consists in:
  1. Byte data are translated to Base64 encoding form with brackets.
  2. The result is translated back to original form.
  3. The recovered data and brackets have to be identical.
  4. The Base64 stream is a little bit modified.
  5. The essay to recover the bytes has to fail.

Success: the initial data are recovered, the recover of corrupted data fails.

The stream to be translated is hardcoded.

3. Test 2: It tests the signature engine using a RSA key in length of 1024 bits. The data and the key are known a priori. The test consists in comparing the signatures obtained with two different engines:

one is the implementation provided through CryptoEngineFactory, and the second provided by default implementation of SUN JCA.

Success: the two signature are identical.

The data to be signed and the key pair are hardcoded.

4. Test 3: Similar to Test 2, the used RSA key is 2048 bits length.
5. Test 4: It tests the verifying process of a signature which uses a RSA key in length of 1024 bits. The test first makes a valid signature (using JCA/JCE default engine) and that one is verified with our engine. There are two branches: the signature is preserved (the result has to be true), the signature is a little bit modified (the results has to be false).

Success: the correct signature is verified, the corrupted signature is not.

The data to be signed and the key pair are hardcoded.

6. Test 5: Similar to Test 4, the used RSA key is 2048 bits length.
7. Test 6: It tests the generation of one RSA key pair of length 1024. First it produces a key pair, which is used to sign the data (with the private key), and after that to verify the signature (with the public key).  
Success: the signature is verified.  
The data to be signed are hardcoded.
8. Test 7: Similar to Test 6, the used RSA key is 2048 bits length.
9. Test 8: It tests the storing capability of PKCS12Engine. The test consists in:
1. Creating a chain of certificates which authenticates a Private Key. Then storing the key and the chain with "pass1" password.
  2. Creating a trusted certificate and storing it.
  3. Protecting the stream with "pass" password.
  4. Extracting with default JCA/JCE provider all elements and comparing them.
- Success: the recovered structures from built storage are identical with that ones which was inputed for store.  
The passwords, the key and the certificates are hardcoded.
10. Test 9: It tests the loading capability of PKCS12Engine. The storage is already built from known data. The test consists in:
1. Loading the p12 data in engine.
  2. Retrieving the key and comparing it with that one which was included in p12 data.
  3. Retrieving the certificate chain and comparing it with that one which was included in p12 data.
  4. Retrieving the trusted certificate and comparing it with that one which was included in p12 data.
- Success: the recovered structures from stream are identical with the ones known to be previously stored.  
The input storage stream and the known components are hardcoded.
11. Test 10: It tests the loading capability of PKCS10Engine. The p10 request is already built from known data. The test consists in:
1. Loading the p10 data in engine.
  2. Verifying the signature.
  3. Retrieving the X509Name and comparing it with that one which was included in p10 data.
  4. Retrieving the subject key and comparing it with that one which was included in p10 data.
  5. Retrieving the extension (which has to be null).
- Success: the recovered structures from stream are identical with that ones known to be previously stored.  
The input storage stream and the known components are hardcoded.
12. Test 11: It tests the storing capability of PKCS10Engine. The test consists in:
1. Creating the p10 engine.
  2. Feeding the engine.
  3. Storing into a byte stream.
  4. Comparing with the known material file.
- Success: the resulted request is identical with that one previously built.

The data to fill the p10 request and the expected stream are hardcoded.

13. Test 12: It tests the PKCS#11 engine. It uses the internal software token from NSS. Previously the NSS/JSS must be installed (not as JCA/JCE provider).

The test consists in:

1. Getting the internal software token.
2. Writing down a private key, the EE and CA cert.
3. Reading them and comparing with that ones which were stored.
4. Signing in token (the private key from it) and signing with default provider and the private key which was stored (plain text); after that comparing the signatures.

TODO: to become a conclusive test, it will be more appropriate if we'll use more than one software token (out of NSS release) for read/write.

TODO: maybe a nicer way to perform test or to lose the fake data from the token will be implemented. It is not the most beautiful way to delete the db files in order to be sure we do not have other data stored.

Success: the data read from token are identical with that ones which were written and the signature in token is verified.

The key, the certificates to be written in token, the data to be signed are hardcoded (and the token is softtoken from NSS).

14. Test 12': It tests PKCS8 engine. First it encrypts/encodes public and private key of length 1024, after that it decodes/decrypts and it compares them.

Success: the decrypted data are identical with that ones which were encrypted.

The keys to be encrypted and the password to encrypt are hardcoded.

15. Test 14: Similar to Test 12', the used key is 2048 bits length.

16. Test 15: Similar to Test 12', it encrypts bulk data and not a key.

17. Test 16: It tests the generation of one RSA key pair of length 1024 using secrets to initialize random source. First it produces a key pair, which is used to sign the data (with the private key), and after that to verify the signature (with the public key).

Success: the signature is verified.

The data to be signed and the secrets are hardcoded.

18. Test 17: It tests Treshlod Secret Sharing Scheme implementation engine (Shamir algorithm). It first creates the shares. After that, it uses a number of shares to retrieve the input data.

Scheme: 3 from 5; retrieving from shares: 1, 3 and 4.

Input data: the PKCS#8 form of RSA private key of length 1024 bits.

Success: if the key is recovered successfully.

The input data and the TSSS parameters are hardcoded.

19. Test 18: It tests Treshlod Secret Sharing Scheme implementation engine (Shamir algorithm). It first creates the shares. After that, it uses a number of shares to retrieve the input data.

Scheme: 3 from 5; retrieving from shares: 5, 2 and 4.

Input data: the PKCS#8 form of RSA private key of length 2048 bits.

Success: if the key is recovered successfully.

The input data and the TSSS parameters are hardcoded.

20. Test 19: It tests Treshlod Secret Sharing Scheme implementation engine (Shamir algorithm). It first creates the shares. After that, it uses a number

of shares to retrieve the input data.

Scheme: 11 from 25; retrieving from shares: 2, 23, 13, 17, 8, 25, 1, 19, 21, 7 and 11.

Input data: the PKCS#8 form of RSA private key of length 2048 bits.

Success: if the key is recovered successfully.

The input data and the TSSS parameters are hardcoded.

21. Test 20: It tests Treshlod Secret Sharing Scheme implementation engine (Shamir algorithm). It first creates the shares. After that, it uses a number of shares to retrieve the input data.

Scheme: 7 from 12; retrieving from shares: 4, 9, 10, 6, 12, 3 and 8.

Input data: large bulk data.

Success: if the data is recovered successfully.

The input data and the TSSS parameters are hardcoded.

22. Test 21: It tests the EntityVerifier engine. It sets the CA Root as trusted anchor, makes the certificate chain composed by End Entity and CA certificate and uses the two CRLs. The verifier has to succeed.

Input data: the chain, trusted anchor and CRLs.

Success: the certificate is verified.

The input data are hardcoded.

23. Test 22: It tests the EntityVerifier engine. It sets the CA Root as trusted anchor, makes the certificate chain composed by Revoked End Entity and CA certificate and uses the two CRLs. The verifier has to fail.

Input data: the chain, trusted anchor and CRLs.

Success: the certificate is not verified.

The input data are hardcoded.

Steps (set of operator input tests):

1. Build the sources
2. Make the file \$TMPDIR/eupki/crypto/tests/properties
3. Edit this file with the appropriate input
4. cd {eupkiProject\_home}/crypto
5. ./build.sh crypto-input-test

The second set of tests is trying to offer the user an ensurance that the system works as wanted. The data are expected from user and the tests rely on them, so if data are in wrong shape the real functionality of the system is not verified. However the system behaviour at wrong input is verified. The tests are not relying on the validity of input, so simply its will fail in case of IO errors or anything wrong supplied from user.

The user can build an input for testing the crypto package by setting the properties in file \$TMPDIR/eupki/crypto/tests/properties.

1. Test 0: What providers do we have? It prints the list of accepted installed

cryptographic providers.

2. Test 1: it tests the translation into/from Base64 encoding form. It reads the input as binary, encodes it (and save the encodation in output file) and again it decodes in order to retrieve the same binary input.

Input: file as binary.

Output: file containing the Base64 encoded form with appended brackets "Start Base64 Test" and "End Base64 Test".

Success: retrieving the same binary data.

3. Test 2: it tests the Signature Engine. It gets the user private and public key. After that, it signs the witness string and saves the signature as binary in output file. The test prints the result of the signature verification with the public key.

Input: 2 files containing the private and public keys.

Output: file with the signature.

Success: the signature engine did its job.

4. Test 3: it tests the key generation. The key pair is saved in PKCS#8 (the private key) and respectively X.509 (the public key) formats into files. Could be specified some secrets to initialize the random source. The key type and length are specified by operator.

Input: parameters of key generation.

Output: 2 files with the encoded key pair.

Success: the key pair factory engine did its job.

5. Test 4: it tests the store capability of PKCS#12 engine. It gathers the data, stores its in an encrypted PKCS#12 store.

Input: trusted certificates (from files named trusted\*.der) and a private key (privkey.pkcs8) with its certificate chain (cert[i].der).

Output: p12 file.

Success: the PKCS#12 engine did its job.

6. Test 5: it tests the load capability of PKCS#12 engine. It reads PKCS#12 store. The content of store is saved in output directory.

Input: p12 file.

Output: trusted\*.der the trusted certificates, privkey.pkcs8 the private key and cert[i].der its certificate chain.

Success: the PKCS#12 engine did its job.

7. Test 6: it tests the store capability of PKCS#10 engine. It gathers the data, stores its in an PKCS#10 request.

Input: a key pair, subject distinguished name and signature algorithm.

Output: p10 file.

Success: the PKCS#10 engine did its job.

8. Test 7: it tests the load capability of PKCS#10 engine. It reads PKCS#10 request. The content of request is printed on screen.

Input: p10 file.

Output: on screen.

Success: the PKCS#10 engine did its job.

9. Test 8: it tests the Secret Share Engine Dealer. It uses as input a file read as binary and implements asked algorithm.

Input: the file to be shared, the algorithm parameters.

Output: the shares.

Success: the TSSS engine did its job.

10. Test 9: it tests the Secret Share Engine Retriever. It uses as input the shares and builds the secret, which is saved in a file.

Input: the share files.

Output: the retrieved secret.

Success: the TSSS engine did its job.

11. Test 10: it tests the PKCS#8 encrypter. It uses as input a file read as binary and the password protecting the data.

Input: data to be encrypted and the password.

Output: encrypted data as PKCS#8 DER.

Success: the PKCS#8 engine did its job.

12. Test 11: it tests the PKCS#8 decrypter. It uses as input a file read as binary and the password protecting the data.

Input: data to be decrypted and the password.

Output: decrypted data.

Success: the PKCS#8 engine did its job.

### 3 Detailed list of tested classes

<b>Module number:</b>	<b>Module name:</b>	<b>Classes:</b>	<b>Tests:</b>
<b>FO-2</b>	<b>Communication to BO</b>		
<b>RA-1</b>	<b>User Interface</b>	org.eupki.ra.web.servlets.* org.eupki.ra.web.beans.* org.eupki.ra.web.tags.* org.eupki.ra.web.util.* eupki/ra/web/jsp/* org.eupki.common.gui.* org.eupki.common.gui.crypto.*	
<b>RA-2</b>	<b>Validation</b>	org.eupki.ra.ejb.services.raAPIBean org.eupki.ra.ejb.interfaces.raAPI org.eupki.ra.ejb.interfaces.raAPIHome	org.eupki.ra.test.raAPITest
<b>RA-3</b>	<b>Log</b>	org.eupki.common.logging.* org.eupki.ra.logging.* org.eupki.common.logging.services.*	
<b>RA-4</b>	<b>RA database</b>	org.eupki.ra.ejb.domain.* org.eupki.ra.ejb.services.SafeDeleteBean org.eupki.ra.ejb.interfaces.{tablename} org.eupki.ra.ejb.interfaces.{tablename}Home org.eupki.ra.ejb.interfaces.{tablename}Data org.eupki.ra.ejb.interfaces.AbstractData org.eupki.ra.ejb.interfaces.*Constants org.eupki.ra.ejb.dto.*	org.eupki.ca.test.RASetup org.eupki.ca.test.RATest org.eupki.ca.test.RATarDown
<b>RA-5</b>	<b>Entry / Renewal</b>	org.eupki.ra.comm.PKIMessageFactory org.eupki.ra.comm.BadRequestDataException org.eupki.ra.comm.RequestDataAddInfo org.eupki.ra.ejb.interfaces.RequestProcessor org.eupki.ra.ejb.interfaces.RequestProcessorHome org.eupki.ra.ejb.services.RequestProcessorBean	org.eupki.ra.test.CommTests
<b>RA-7</b>	<b>Communication to CA</b>	org.eupki.common.comm.* org.eupki.ra.ejb.services.ResponseProcessorMDB org.eupki.ca.comm.RequestReceiverMDB	
	<b>(XER-encoding)</b>	org.eupki.common.comm.encoding.xer.* org.eupki.common.comm.encoding.* org.eupki.common.comm.pkix.*	org.eupki.common.test.XERTest

	<b>(Signature)</b>	org.eupki.common.comm.cms.SignatureCMS	
<b>RA-8</b>	<b>Audit</b>		
<b>RA-10</b>	<b>RA-API</b>	org.eupki.ra.ejb.services.raAPIBean	org.eupki.ra.test.raAPITest
		org.eupki.ra.ejb.interfaces.raAPI	
		org.eupki.ra.ejb.interfaces.raAPIHome	
		org.eupki.ra.comm.PKIMessageFactory	org.eupki.ra.test.CommTests
		org.eupki.ra.comm.BadRequestDataException	
		org.eupki.ra.comm.RequestDataAddInfo	
		org.eupki.common.comm.cmp.*	
<b>RA-13</b>	<b>Revocation</b>	org.eupki.ra.ejb.interfaces.Revocation*	
		org.eupki.ra.ejb.services.RevocationBean	
<b>RA-14</b>	<b>Entity/Cert Browser</b>		
<b>RA-15</b>	<b>Access Control</b>	org.eupki.common.ac.*;	
		org.eupki.ra.ejb.interfaces.AccessControl	
		org.eupki.ra.ejb.interfaces.AccessControlHome	
		org.eupki.ra.ejb.services.AccessControlBean	
		org.eupki.ra.ejb.services.OperatorProfileImpl	
<b>RA-16</b>	<b>Admin Interface</b>		
<b>RA-17</b>	<b>Admin API</b>	org.eupki.ra.ejb.services.RAAdminAPIBean	org.eupki.ra.test.RAAdminAPITest
		org.eupki.ra.ejb.interfaces.RAAdminAPIHome	
		org.eupki.cra.ejb.interfaces.RAAdminAPI	
<b>RA-18</b>	<b>RA PK Acceptor</b>	org.eupki.crypto.ejb.interfaces.PKAcceptor	
		org.eupki.crypto.ejb.interfaces.PKAcceptorHome	
		org.eupki.crypto.ejb.services.PKAcceptorBean	
		org.eupki.crypto.ejb.services.PublicKeyRejectedException	
<b>RA-20</b>	<b>Communication to FO</b>		
<b>CA-1</b>	<b>User Request Handler</b>	org.eupki.ca.comm.UserRequestProcessorMDB	org.eupki.ca.test.CommUserTest
		org.eupki.common.comm.userkgs.*	
<b>CA-2</b>	<b>Writemail</b>	org.eupki.ca.ejb.services.WriteMail	org.eupki.ca.test.MailTest
		org.eupki.ca.ejb.services.MailSendException	
<b>CA-3</b>	<b>Log</b>	org.eupki.common.logging.*	
		org.eupki.ca.logging.*	
		org.eupki.common.logging.services.*	
<b>CA-5</b>	<b>Admin Interface</b>	org.eupki.ca.web.servlets.*	
		org.eupki.ca.web.beans.*	
		org.eupki.ca.web.tags.*	
		org.eupki.ca.web.util.*	
		eupki/ca/web/jsp/*	
<b>CA-6</b>	<b>Admin</b>	org.eupki.ca.ejb.services.CAAdminAPIBean	org.eupki.ca.test.CAAdminAPITest

	<b>API</b>		
		org.eupki.ca.ejb.interfaces.CAAdminAPIHomeLocal	
		org.eupki.ca.ejb.interfaces.CAAdminAPILocal	
<b>CA-8</b>	<b>Audit</b>		
	<b>OTP distributo</b>		
<b>CA-9</b>	<b>r</b>	org.eupki.ca.ejb.services.OTPDistributorBean	org.eupki.ca.test.OTPDistributorEJBTest
		org.eupki.ca.ejb.interfaces.OTPDistributorHome	
		org.eupki.ca.ejb.interfaces.OTPDistributor	
		org.eupki.ca.ejb.interfaces.OTPDistributorHomeLocal	
		org.eupki.ca.ejb.interfaces.OTPDistributorLocal	
<b>CA-10</b>	<b>CA-API</b>	org.eupki.ca.comm.RARequestProcessorMDB	org.eupki.ca.test.CommSetup
		org.eupki.ca.comm.KGSResponseProcessorMDB	
		org.eupki.ca.comm.RequestReceiverMDB	org.eupki.ca.test.CommTearDown
		org.eupki.ca.comm.RARequestValidator	org.eupki.ca.test.CommKGSTest
		org.eupki.ca.comm.BadRequestException	
		org.eupki.ca.comm.InvalidRAException	
		org.eupki.ca.comm.InvalidRequestException	
		org.eupki.common.comm.cmp.*	
		org.eupki.common.ext.*	org.eupki.common.test.ExtTest
		org.eupki.common.TimerThread	
		org.eupki.common.TimerListener	
		org.eupki.common.TimerException	
<b>CA-11</b>	<b>OTP Generator</b>	org.eupki.ca.ejb.services.OTPGenMbean.java	
		org.eupki.ca.ejb.services.OTPGen.java	
		org.eupki.ca.OTP	
		org.eupki.ca.OTPGenerator	
		org.eupki.ca.NullOTPPassException	
		org.eupki.ca.NullOTPIdException	
<b>CA-12</b>	<b>OTP Authenticator</b>	org.eupki.ca.ejb.services.OTPAuthenticatorBean	
		org.eupki.ca.ejb.interfaces.OTPAuthenticator*	
<b>CA-13</b>	<b>Communication to CSP</b>	org.eupki.ca.ejb.services.CSPSessionBean	
		org.eupki.ca.ejb.interfaces.CSPSession*	
<b>CA-14</b>	<b>CA database</b>	org.eupki.ca.ejb.domain.*	org.eupki.ca.test.CASetup
		org.eupki.ca.ejb.interfaces.{tablename}	org.eupki.ca.test.CATest
		org.eupki.ca.ejb.interfaces.{tablename}Home	org.eupki.ca.test.CATearDown
		org.eupki.ca.ejb.interfaces.{tablename}Data	
		org.eupki.ca.ejb.interfaces.AbstractData	
		org.eupki.ca.ejb.interfaces.*Constants	
		org.eupki.ca.ejb.services.SafeDeleteBean	
<b>CA-15</b>	<b>Access Control</b>	org.eupki.common.ac.*;	
		org.eupki.ca.ejb.interfaces.AccessControl	
		org.eupki.ca.ejb.interfaces.AccessControlHome	

		org.eupki.ca.ejb.services.AccessControlBean	
		org.eupki.ca.ejb.services.OperatorProfileImpl	
<b>CA-16</b>	<b>Publication</b>	org.eupki.ca.ejb.services.LdapAccessBean	org.eupki.ca.test.LdapTest
		org.eupki.common.Ldap.*	
<b>CA-17</b>	<b>Communication to KGS</b>	org.eupki.ca.comm.RARequestProcessorMDB	org.eupki.ca.test.CommKGSTest
		org.eupki.ca.comm.KGSResponseProcessorMDB	
		org.eupki.common.comm.kgs.*	
		org.eupki.common.comm.*	
<b>CA-18</b>	<b>CRL Factory</b>	org.eupki.ca.ejb.services.CRLFactory	
		org.eupki.ca.ejb.services.CRLFactoryMBean	
<b>CA-20</b>	<b>Certificate Update Agent</b>	org.eupki.ca.ejb.services.CertUpdateAgent	org.eupki.ca.test.Comm{Setup,Test,TearDown}
<b>CSP-1</b>	<b>CSP-API</b>	org.eupki.ca.csp.CSP	org.eupki.ca.test.CSPTestPerf
		org.eupki.ca.csp.CSPFactory	org.eupki.ca.test.CSPTest
		org.eupki.ca.csp.CSPBCProvider	
		org.eupki.ca.csp.CSPConstants	
<b>CSP-2</b>	<b>Certificate Signer</b>	org.eupki.ca.csp.X509CertSigner	org.eupki.ca.test.CSPTest
<b>CSP-3</b>	<b>CRL Signer</b>	org.eupki.ca.csp.X509CRLSigner	org.eupki.ca.ejb.services.CRLFactory
<b>(CSP-4)</b>	<b>(Crypto Engine, sub module)</b>	org.eupki.ca.csp.CSPCryptoEngine	
		org.eupki.crypto.SignatureEngine	
<b>KC-2</b>	<b>Certificate Factory</b>	org.eupki.kc.KCAPIImplementation	org.eupki.kc.test.KCTest.java
		org.eupki.kc.CertRequestStructure	
		org.eupki.kc.crypto.CertificateGenerator	
		org.eupki.kc.crypto.CertificateSignature	
		org.eupki.kc.*SignerResponseStructure	
		org.eupki.kc.*SignerRequestStructure	
<b>KC-5</b>	<b>Admin Interface</b>	org.eupki.kc.CommandLineRunner	
<b>KC-6</b>	<b>Admin API</b>	org.eupki.kc.KCAPI	
<b>KC-15</b>	<b>Access Control</b>	N/A	N/A
<b>KC-26</b>	<b>Key Pair Factory</b>	org.eupki.kc.KCAPIImplementation	org.eupki.kc.test.KCTest.java
		org.eupki.kc.crypto.KeysGenerator	org.eupki.crypto.test.CryptoTest.java
		org.eupki.kc.*KeysGenResponse	
		org.eupki.crypto.KeyPairFactory	
<b>KC-30</b>	<b>Secret Export</b>	org.eupki.kc.KCAPIImplementation	org.eupki.kc.test.KCTest.java
		org.eupki.crypto.ShareSecretsEngine	org.eupki.crypto.test.CryptoTest.java

		org.eupki.crypto.ShareSecretsDealer	
<b>KC-31</b>	<b>Secret Printing</b>	org.eupki.kc.KCAPIImplementation	
<b>KC-32</b>	<b>Secret Import</b>	org.eupki.kc.KCAPIImplementation	org.eupki.kc.test.KCTest.java
		org.eupki.crypto.ShareSecretsEngine	org.eupki.crypto.test.CryptoTest.java
		org.eupki.crypto.ShareSecretsRetriever	
<b>KGS-3</b>	<b>Log</b>	org.eupki.kgs.logging.*	
		org.eupki.common.logging.*	
		org.eupki.common.logging.services.*	
<b>KGS-5</b>	<b>Admin Interface</b>		
<b>KGS-6</b>	<b>Admin API</b>		
<b>KGS-8</b>	<b>Audit</b>		
<b>KGS-10</b>	<b>KGS-API Job Scheduler</b>	org.eupki.kgs.comm.CARRequestProcessorMDB.java	org.eupki.kgs.test.AllTests.java
			org.eupki.kgs.test.KGSTest.java
<b>KGS-15</b>	<b>Access Control (Central KGS)</b>	org.eupki.common.ac.*;	
		org.eupki.kgs.ejb.interfaces.AccessControl	
		org.eupki.kgs.ejb.interfaces.AccessControlHome	
		org.eupki.kgs.ejb.services.AccessControlBean	
		org.eupki.kgs.ejb.services.OperatorProfileImpl	
<b>KGS-22</b>	<b>Certificate Storage</b>	org.eupki.crypto.PKCS8Engine.java	org.eupki.kgs.test.AllTests.java
		org.eupki.crypto.PKCS12Engine.java	org.eupki.kgs.test.KGSTest.java
		org.eupki.crypto.PKCS11Engine.java	org.eupki.crypto.test.AllTests.java
		org.eupki.kgs.ejb.interfaces.CertRepository.java	org.eupki.crypto.test.CryptoTest.java
<b>KGS-26</b>	<b>Key Pair Factory</b>	org.eupki.crypto.PKCS11Engine.java	org.eupki.kgs.test.AllTests.java
		org.eupki.kgs.daemons.*	org.eupki.kgs.test.KGSTest.java
		org.eupki.crypto.PKCS8Engine.java	org.eupki.crypto.test.AllTests.java
			org.eupki.crypto.test.CryptoTest.java
<b>KGS-14</b>	<b>Key Store</b>	org.eupki.crypto.PKCS8Engine.java	org.eupki.kgs.test.AllTests.java
		org.eupki.crypto.PKCS12Engine.java	org.eupki.kgs.test.KGSTest.java
		org.eupki.crypto.PKCS11Engine.java	org.eupki.crypto.test.AllTests.java
		org.eupki.kgs.ejb.interfaces.KeyTable.java	org.eupki.crypto.test.CryptoTest.java
<b>KGS-21</b>	<b>Key Pair Generator</b>	org.eupki.crypto.ejb.interfaces.*	org.eupki.kgs.test.AllTests.java
		org.eupki.crypto.ejb.services.*	org.eupki.kgs.test.KGSTest.java
		org.eupki.crypto.KeyPairFactory.java	org.eupki.crypto.test.AllTests.java
		org.eupki.crypto.PKCS11Engine.java	org.eupki.crypto.test.CryptoTest.java
<b>KGS-23</b>	<b>Export Certificate Request to CA</b>		
<b>KGS-24</b>	<b>Import</b>		

	<b>Certificate Replies from CA</b>		
<b>KGS-25</b>	<b>User Interface</b>		
<b>KGS-27</b>	<b>User KGS API</b>		
<b>KGS-28</b>	<b>Access Control (User KGS)</b>		
<b>KGS-29</b>	<b>Export PKCS#12</b>	org.eupki.crypto.PKCS12Engine.java	org.eupki.crypto.test.CryptoTest.java
<b>KGS-30</b>	<b>KGS Database</b>	org.eupki.kgs.ejb.domain.*	org.eupki.kgs.test.AllTests.java
		org.eupki.kgs.ejb.interfaces.*	org.eupki.kgs.test.KGSTest.java
		org.eupki.kgs.ejb.dto.*	org.eupki.kgs.test.DatabaseTest.java
		org.eupki.kgs.ejb.vo.*	

## 4 Additional tests

During the November 30-th and December 15-th 2003 , we've made a quick integration between the modules , to facilitate the integration work.

The installation was made on 3 different servers , with separate Ra , CA , and Kgs .

The installation instructions are here : <http://projects.axetel.com/cgi-bin/cvsweb.cgi/eupki/doc/distributed-setup.txt>

The global test was successful, the RA, CA and KGS established successful communications and certificates were delivered using the central KGS .

The most important test was the communication.

Here is below a test for a successful RA /CA communication :

### 4.1.1 RA-CA communication test

The sample output snippets correspond to:

- ca-bad-profile.txt - bad profile specification when setting up the RA (you need to edit ra/src/conf/tests/ra-setup.properties and specify a correct profile id number).

- ca-sample-output.txt - sample full output of the CA receiving a keyless certificate request from the RA, asking the KGS for a key, generating the certificate and sending the KGS a 'storage' request.

#### 4.1.1.1 CA bad profile communication output

```

16:37:21,629 INFO [STDOUT] [EUPKI] Bad request exception: org.eupki.ca.comm.BadRequestException: Profile no. 52 not found
16:37:21,629 ERROR [RAResultProcessorMDB] Bad request exception: org.eupki.ca.comm.BadRequestException: Profile no. 52 not found
16:37:21,641 INFO [STDOUT] [EUPKI] sendReply called, reply: org.eupki.common.comm.cmp.CertRepMsg
16:37:21,649 ERROR [STDERR] org.eupki.ca.comm.BadRequestException: Profile no. 52 not found
16:37:21,650 ERROR [STDERR] at org.eupki.ca.comm.RARequestValidator.validateRequest(RARequestValidator.java:194)
16:37:21,650 ERROR [STDERR] at org.eupki.ca.comm.RARequestProcessorMDB.onMessage(RARequestProcessorMDB.java:285)
16:37:21,650 ERROR [STDERR] at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
16:37:21,650 ERROR [STDERR] at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
16:37:21,651 ERROR [STDERR] at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
16:37:21,651 ERROR [STDERR] at java.lang.reflect.Method.invoke(Method.java:324)
16:37:21,651 ERROR [STDERR] at org.jboss.ejb.MessageDrivenContainer$ContainerInterceptor.invoke(MessageDrivenContainer.java:434)
16:37:21,651 ERROR [STDERR] at
org.jboss.resource.connectionmanager.CachedConnectionInterceptor.invoke(CachedConnectionInterceptor.java:186)
16:37:21,651 ERROR [STDERR] at org.jboss.ejb.plugins.AbstractTxInterceptor.invokeNext(AbstractTxInterceptor.java:84)
16:37:21,651 ERROR [STDERR] at org.jboss.ejb.plugins.AbstractTxInterceptorBMT.invokeNext(AbstractTxInterceptorBMT.java:144)
16:37:21,651 ERROR [STDERR] at org.jboss.ejb.plugins.MessageDrivenTxInterceptorBMT.invoke(MessageDrivenTxInterceptorBMT.java:32)
16:37:21,651 ERROR [STDERR] at
org.jboss.ejb.plugins.MessageDrivenInstanceInterceptor.invoke(MessageDrivenInstanceInterceptor.java:62)
16:37:21,651 ERROR [STDERR] at org.jboss.ejb.plugins.RunAsSecurityInterceptor.invoke(RunAsSecurityInterceptor.java:90)
16:37:21,651 ERROR [STDERR] at org.jboss.ejb.plugins.LogInterceptor.invoke(LogInterceptor.java:191)
16:37:21,651 ERROR [STDERR] at org.jboss.ejb.plugins.ProxyFactoryFinderInterceptor.invoke(ProxyFactoryFinderInterceptor.java:122)
16:37:21,651 ERROR [STDERR] at org.jboss.ejb.MessageDrivenContainer.internalInvoke(MessageDrivenContainer.java:348)
16:37:21,651 ERROR [STDERR] at org.jboss.ejb.Container.invoke(Container.java:674)
16:37:21,652 ERROR [STDERR] at org.jboss.ejb.plugins.jms.JMSContainerInvoker.invoke(JMSContainerInvoker.java:732)
16:37:21,652 ERROR [STDERR] at
org.jboss.ejb.plugins.jms.JMSContainerInvoker$MessageListenerImpl.onMessage(JMSContainerInvoker.java:1020)
16:37:21,652 ERROR [STDERR] at org.jboss.jms.asf.StdServerSession.onMessage(StdServerSession.java:241)
16:37:21,652 ERROR [STDERR] at org.jboss.mq.SpyMessageConsumer.sessionConsumerProcessMessage(SpyMessageConsumer.java:637)

```

```

16:37:21,652 ERROR [STDERR] at org.jboss.mq.SpyMessageConsumer.addMessage(SpyMessageConsumer.java:442)
16:37:21,652 ERROR [STDERR] at org.jboss.mq.SpySession.run(SpySession.java:294)
16:37:21,652 ERROR [STDERR] at org.jboss.jms.asf.StdServerSession.run(StdServerSession.java:177)
16:37:21,652 ERROR [STDERR] at EDU.oswego.cs.dl.util.concurrent.PooledExecutor$Worker.run(PooledExecutor.java:727)
16:37:21,652 ERROR [STDERR] at java.lang.Thread.run(Thread.java:534)

```

#### 4.1.1.2 CA sample communication output

```

17:30:12,325 INFO [STDOUT] [EUPKI] Encapsulated message type: org.eupki.common.comm.cms.CMSMessage
17:30:12,326 INFO [STDOUT] [EUPKI] String representation: org.eupki.common.comm.cms.CMSMessage@1df01a0
17:30:12,326 INFO [STDOUT] [EUPKI] Encoded message: <PKIMessage><header><pvno><ietf-
version2/></pvno><sender><directoryName><rdnSequence><RelativeDistinguishedName><AttributeTypeAndValue><type>2.5.4.10</type>
<value><utf8String>axetel</utf8String></value></AttributeTypeAndValue><AttributeTypeAndValue><type>2.5.4.6</type><value><utf8Stri
ng>RO</utf8String></value></AttributeTypeAndValue><AttributeTypeAndValue><type>2.5.4.3</type><value><utf8String>ra1</utf8String
></value></AttributeTypeAndValue></RelativeDistinguishedName></rdnSequence></directoryName></sender><recipient><directoryName
><rdnSequence><RelativeDistinguishedName><AttributeTypeAndValue><type>2.5.4.10</type><value><utf8String>axetel</utf8String></v
alue></AttributeTypeAndValue><AttributeTypeAndValue><type>2.5.4.6</type><value><utf8String>RO</utf8String></value></Attribu
teTypeAndValue><AttributeTypeAndValue><type>2.5.4.3</type><value><utf8String>ca1</utf8String></value></AttributeTypeAndValue></Rel
ativeDistinguishedName></rdnSequence></directoryName></recipient><messageTime><utcTime>031211153315Z</utcTime></messageTi
me></header><body><cr><certReq><certReqId>253</certReqId><certTemplate><signingAlg><algorithm>1.2.840.113549.1.1.5</algorithm
></signingAlg><issuer><rdnSequence><RelativeDistinguishedName><AttributeTypeAndValue><type>2.5.4.10</type><value><utf8String
>axetel</utf8String></value></AttributeTypeAndValue><AttributeTypeAndValue><type>2.5.4.6</type><value><utf8String>RO</utf8String
></value></AttributeTypeAndValue><AttributeTypeAndValue><type>2.5.4.3</type><value><utf8String>ca1</utf8String></value></Attrib
uteTypeAndValue></RelativeDistinguishedName></rdnSequence></issuer><validity><notBefore><utcTime>031210220000Z</utcTime></n
otBefore><notAfter><utcTime>041210220000Z</utcTime></notAfter></validity><subject><rdnSequence><RelativeDistinguishedName><A
ttributeTypeAndValue><type>2.5.4.10</type><value><utf8String>cosa
nostra</utf8String></value></AttributeTypeAndValue><AttributeTypeAndValue><type>2.5.4.7</type><value><utf8String>New
York</utf8String></value></AttributeTypeAndValue><AttributeTypeAndValue><type>2.5.4.6</type><value><utf8String>US</utf8String><
/value></AttributeTypeAndValue><AttributeTypeAndValue><type>2.5.4.3</type><value><utf8String>Marlon
corleone</utf8String></value></AttributeTypeAndValue><AttributeTypeAndValue><type>2.5.4.11</type><value><utf8String>Business</ut
f8String></value></AttributeTypeAndValue></RelativeDistinguishedName></rdnSequence></subject><extensions><Extension><extnID>2.
5.29.17</extnID><critical><false/></critical><extnValue>30138111636F726C656F6E65406D616669612E726F</extnValue></Extension><Ex
tension><extnID>2.5.29.15</extnID><critical><false/></critical><extnValue>030206C0</extnValue></Extension></extensions></certTem
plate><controls><AttributeTypeAndValue><type>1.3.6.1.5.5.7.5.1.254</type><value>62</value></AttributeTypeAndValue></controls></c
ertReq></cr></body></PKIMessage>
17:30:12,338 INFO [STDOUT] Found: 1 controls nodes
17:30:12,339 INFO [STDOUT] Decoded oid:1.3.6.1.5.5.7.5.1.254
17:30:12,339 INFO [STDOUT] Control: profileId
17:30:12,339 INFO [STDOUT] Controls: {}
17:30:12,339 INFO [STDOUT] node name: value
17:30:12,339 INFO [STDOUT] node value: null
17:30:12,339 INFO [STDOUT] decoded controls: {profileId=62}
17:30:12,339 INFO [STDOUT] profileId = 62
17:30:12,340 INFO [STDOUT] [EUPKI] Decoded msg: ===== PKI Message begin: =====
Message header:
header: pvno = 1
header: sender = Directory name: O=axetel,C=RO,CN=ra1
header: recipient = Directory name: O=axetel,C=RO,CN=ca1
header: messageTime = Thu Dec 11 17:33:15 EET 2003
header: senderKID = null
header: recipKID = null
header: transationId = null
header: senderNonce = null
header: recipNonce = null
header: freeText = null
header: generalInfo = null
header: msgType = 0
Message body: certificate request message
body: certReqId = 253
body: template follows
cert template:
template: serial number = null
template: Signing algorithm: org.eupki.common.comm.cmp.PKIAlgorithmIdentifier@1e3bb9e
template: issuer = CN=ca1,C=RO,O=axetel
template: subject = OU=Business,CN=Marlon corleone,C=US,L=New York,O=cosa nostra
template: validity: not before = Thu Dec 11 00:00:00 EET 2003
template: validity: not after = Sat Dec 11 00:00:00 EET 2004
template: public key = null
template: ***** begin extensions list *****
*** Entry 0 in extensions list: ***

```

Subject Alternative Name extension (oid = 2.5.29.17):  
 1 alternative names included  
 0: Email address: corleone@mafia.ro  
 \*\*\* Entry 1 in extensions list: \*\*\*  
 Key Usage Extension (oid = 2.5.29.15):  
 Signature  
 Non-repudiation  
 template: \*\*\*\*\* end of extensions list \*\*\*\*\*  
 body: end template  
 ===== PKI Message end: =====  
 17:30:12,348 INFO [STDOUT] [EUPKI] Decoded signature: org.eupki.common.comm.cms.SignatureCMS@329ba8  
 17:30:12,348 INFO [STDOUT] Cert chain: null  
 17:30:12,359 INFO [STDOUT] [EUPKI] Enter onMessage() on MDB 23175637  
 17:30:12,359 INFO [RARRequestProcessorMDB] Enter onMessage() on MDB 23175637  
 17:30:12,363 INFO [STDOUT] [EUPKI] Calling validator for a class org.eupki.common.comm.cmp.CertReqMsg instance  
 17:30:12,391 INFO [STDOUT] [EUPKI] Enter validateRequest  
 17:30:12,394 INFO [STDOUT] [EUPKI] Checking CertReqMsg  
 17:30:12,398 INFO [STDOUT] [EUPKI] Checking extensions...  
 17:30:12,399 INFO [STDOUT] [EUPKI] Profile: ca/ProfileEJB:62  
 17:30:12,401 INFO [STDOUT] [EUPKI] Profile extensions found: 2  
 17:30:12,401 INFO [STDOUT] [EUPKI] Found 2 extensions in request  
 17:30:12,406 INFO [STDOUT] [EUPKI] Searching for required extension: 2.5.29.15  
 17:30:12,407 INFO [STDOUT] [EUPKI] Found all required extensions  
 17:30:12,439 INFO [STDOUT] [EUPKI] Validated cert request on behalf of OU=Business,CN=Marlon corleone,C=US,L=New York,O=cosa nostra  
 17:30:12,439 INFO [RARRequestValidator] Validated cert request on behalf of OU=Business,CN=Marlon corleone,C=US,L=New York,O=cosa nostra  
 17:30:12,439 INFO [STDOUT] [EUPKI] Exit validateRequest with return value: 42  
 17:30:12,441 INFO [STDOUT] [EUPKI] Done validating  
 17:30:12,441 INFO [STDOUT] [EUPKI] Sending our first reply  
 17:30:12,441 INFO [STDOUT] [EUPKI] Acknowledging message  
 17:30:12,441 INFO [STDOUT] [EUPKI] Sender: O=axetel,C=RO,CN=ra1  
 17:30:12,441 INFO [RARRequestProcessorMDB] Sender: O=axetel,C=RO,CN=ra1  
 17:30:12,442 INFO [STDOUT] [EUPKI] sendReply called, reply: org.eupki.common.comm.cmp.CertRepMsg  
 17:30:12,500 INFO [STDOUT] [EUPKI] End of the first transaction  
 17:30:12,501 INFO [STDOUT] profileId = 62  
 17:30:12,502 INFO [STDOUT] [EUPKI] Got cert request message: Message body: certificate request message  
 body: certReqId = 253  
 body: template follows  
 cert template:  
 template: serial number = null  
 template: Signing algorithm: org.eupki.common.comm.cmp.PKIAAlgorithmIdentifier@3267dd  
 template: issuer = CN=ca1,C=RO,O=axetel  
 template: subject = OU=Business,CN=Marlon corleone,C=US,L=New York,O=cosa nostra  
 template: validity: not before = Thu Dec 11 00:00:00 EET 2003  
 template: validity: not after = Sat Dec 11 00:00:00 EET 2004  
 template: public key = null  
 template: \*\*\*\*\* begin extensions list \*\*\*\*\*  
 \*\*\* Entry 0 in extensions list: \*\*\*  
 Subject Alternative Name extension (oid = 2.5.29.17):  
 1 alternative names included  
 0: Email address: corleone@mafia.ro  
 \*\*\* Entry 1 in extensions list: \*\*\*  
 Key Usage Extension (oid = 2.5.29.15):  
 Signature  
 Non-repudiation  
 template: \*\*\*\*\* end of extensions list \*\*\*\*\*  
 body: end template  
 17:30:12,504 INFO [STDOUT] [EUPKI] Sending key request to KGS: ===== PKI Message begin: =====  
 CA/Central KGS message header:  
 header: version = 1  
 header: transaction ID = null  
 header: request identifier = CA ID = 0  
 CA Request Id = 42  
 Subject ID = 0  
 Subject distinguish name = OU=Business,CN=Marlon corleone,C=US,L=New York,O=cosa nostra  
 header: sender = Directory name: O=axetel,C=RO,CN=ca1  
 header: receiver = null  
 header: message time = null  
 header: protection algorithm = null  
 header: sender KID = null  
 header: receiver KID = null  
 header: sender Nonce = null

```
header: receiver Nonce = null
header: free text = null
header: general info = null
header: message type = Request for generating key pair and returning public key
-----END MESSAGE HEADER-----
CA/ Central KGS request for public key body:
body: key type = RSA
body: key length = 1024
body: support identifier is missing
-----END MESSAGE BODY-----
===== PKI Message end: =====
17:30:12,534 INFO [STDOUT] [EUPKI] 1 message(s) sent to QUEUE.fromRACMSQueue
17:30:17,488 INFO [STDOUT] [EUPKI] Generated a cert for: OU=Business,CN=Marlon corleone,C=US,L=New York,O=cosa nostra
17:30:17,488 INFO [EuPKICertificateFactory] Generated a cert for: OU=Business,CN=Marlon corleone,C=US,L=New York,O=cosa nostra
17:30:17,825 INFO [STDOUT] [EUPKI] Creating reply to be sent to the RA
17:30:17,830 INFO [STDOUT] [EUPKI] Sending 'accepted' reply to the RA: C=RO,O=axetel,CN=ra1
17:30:17,934 INFO [STDOUT] [EUPKI] Sent request for key storing for certificate with serial no. 1070103079278
17:30:17,934 INFO [KGSResponseProcessorMDB] Sent request for key storing for certificate with serial no. 1070103079278
17:30:18,903 INFO [STDOUT] [EUPKI] Received message from KGS !
17:30:18,903 INFO [KGSResponseProcessorMDB] Received message from KGS !
17:30:18,909 INFO [STDOUT] [EUPKI] Got response from KGS for a Key Generation request !
17:30:18,909 INFO [KGSResponseProcessorMDB] Got response from KGS for a Key Generation request !
17:30:19,912 INFO [STDOUT] [EUPKI] Generated a cert for: OU=Business,CN=Marlon corleone,C=US,L=New York,O=cosa nostra
17:30:19,912 INFO [EuPKICertificateFactory] Generated a cert for: OU=Business,CN=Marlon corleone,C=US,L=New York,O=cosa nostra
17:30:20,247 INFO [STDOUT] [EUPKI] Creating reply to be sent to the RA
17:30:20,249 INFO [STDOUT] [EUPKI] Sending 'accepted' reply to the RA: C=RO,O=axetel,CN=ra1
17:30:20,354 INFO [STDOUT] [EUPKI] Sent request for key storing for certificate with serial no. 1070103088550
17:30:20,354 INFO [KGSResponseProcessorMDB] Sent request for key storing for certificate with serial no. 1070103088550
17:30:29,090 INFO [STDOUT] [EUPKI] Received message from KGS !
17:30:29,089 INFO [KGSResponseProcessorMDB] Received message from KGS !
17:30:29,117 INFO [STDOUT] [EUPKI] Got response from KGS for a store request !
17:30:29,117 INFO [KGSResponseProcessorMDB] Got response from KGS for a store request !
17:30:29,118 INFO [STDOUT] [EUPKI] Message: Central KGS/ CA response body:
body: response = Success
body: error = Request completed successfully
body: no exception caught
body: data material is missing
-----END MESSAGE BODY-----
17:30:29,144 INFO [STDOUT] [EUPKI] Could not find the certificate for this reqID : 41
17:30:29,144 ERROR [KGSResponseProcessorMDB] Could not find the certificate for this reqID : 41
17:30:29,264 INFO [STDOUT] [EUPKI] Received message from KGS !
17:30:29,264 INFO [KGSResponseProcessorMDB] Received message from KGS !
17:30:29,270 INFO [STDOUT] [EUPKI] Got response from KGS for a store request !
17:30:29,270 INFO [KGSResponseProcessorMDB] Got response from KGS for a store request !
17:30:29,270 INFO [STDOUT] [EUPKI] Message: Central KGS/ CA response body:
body: response = Success
body: error = Request completed successfully
body: no exception caught
body: data material is missing
-----END MESSAGE BODY-----
17:30:29,995 INFO [STDOUT] [EUPKI] Finished processing the certificate with serial no. 1070103088550
17:30:29,995 INFO [KGSResponseProcessorMDB] Finished processing the certificate with serial no. 1070103088550
```