	<p>IST-2001-34340</p> <p>Draft of D 4.2:</p> <p>External Communication Interface</p>
Distribution List:	Project Partners
Author:	WP4 Members
Authorised by:	---
Date of Issue:	02.12.2002
Issue:	Version 1.0
File name:	D4.2-draft-1.0.doc
Work package:	WP4 Specifications
Deliverable Number:	4.2
Deliverable Type:	Public
Deliverable Nature:	Specification
Total Number of Pages:	25
Contact Details for EUPKI:	christoph.schiller@gdm.de

Copyright © The EU-PKI Consortium

0 Document Control

<i>Issue</i>	<i>Date of Issue</i>	<i>Comments</i>
0.01	02 Oct 2002	First version
0.02	18 Oct 2002	Very First Draft by CertiNomis
0.03	21 Oct 2002	Merged with Draft by CGEY
0.04	22 Oct 2002	Update by CGEY
0.05	25 Oct 2002	Update by CertiNomis
0.06	28 Oct 2002	Deletion of smartcard undone by G&D
0.07	07 Nov 2002	Update by CertiNomis
0.08	22 Nov 2002	Changes after review meeting
1.0	02 Dec 2002	Validated Version

0.1 Abstract

This document D4.2 is the second of three deliverables of the specification phase of the EU-PKI project.

D4.2 specifies the external communication interfaces for each component of the EU-PKI system.

D4.2 is based on D4.1 which specifies the global architecture of the EU-PKI system. D4.2 is the basis for D4.3 which will contain the internal functional specification.

0.2 Table of Contents

- 0.1 ABSTRACT 2
- 0.2 TABLE OF CONTENTS..... 3
- 1.1 COMMUNICATION RA – CA..... 5
 - 1.1.1 *Certificate Production*..... 5
 - 1.1.1.1 Communication from RA to CA (request) 5
 - 1.1.1.2 Communication from CA to RA (response) 6
 - 1.1.1.3 Example of Register without key binding 6
 - 1.1.1.4 Example of Register with key binding (PKCS10) 6
 - 1.1.1.5 Example of positive response for Register..... 7
 - 1.1.1.6 Example of negative response for Register 7
 - 1.1.2 *Certificate Renewal*..... 7
 - 1.1.2.1 Example of Renewal with "valid status" 8
 - 1.1.2.2 Example of Renewal with "valid status" and a new Public Key..... 8
 - 1.1.2.3 Example of Renewal with "valid status" and ask for Private Key generation 8
 - 1.1.3 *Certificate Revocation (or Suspension)*..... 9
 - 1.1.3.1 Communication from RA to CA (request) 9
 - 1.1.3.2 Communication from CA to RA (response) 9
 - 1.1.3.3 Example of Revocation with key info 9
 - 1.1.3.4 Example of positive response for Revocation..... 10
 - 1.1.4 *Certificate Reactivation*..... 10
 - 1.1.4.1 Example of Reactivation with key info 10
- 1.2 COMMUNICATION CA – DIRECTORY 11
 - 1.2.1 *Certificate Production*..... 11
 - 1.2.2 *Certificate Renewal*..... 11
 - 1.2.3 *Certificate Revocation (or Suspension)*..... 11
 - 1.2.4 *Certificate Reactivation*..... 11
 - 1.2.5 *CRL Production*..... 12
 - 1.2.6 *Certificate properties in LDAP*..... 12
 - 1.2.7 *CRL properties in LDAP* 12
- 1.3 COMMUNICATION CA – CENTRAL KGS..... 13
 - 1.3.1 *Key Generation*..... 13
 - 1.3.1.1 *Key generation request* 13
 - 1.3.1.2 *Key generation request completed* 15
 - 1.3.2 *Certificate Storage*..... 16
 - 1.3.2.1 *Certificate storage request*..... 16
 - 1.3.2.2 *Certificate storage completed*..... 18
- 1.4 COMMUNICATION USER KGS – CA 19
 - 1.4.1 *Certificate Production*..... 19

- 1.4.1.1 Communication from User KGS to CA 19
- 1.4.1.2 Communication from CA to User KGS (response) 20
- 1.4.1.3 Example of Key Binding (KeyValue, ProofOfPossession and KeyBindingAuth) 20
- 1.4.1.4 Example of positive response for Key Binding 20
- 1.4.2 Certificate Renewal..... 21*
- 1.4.3 Certificate Revocation (or Suspension)..... 21*
 - 1.4.3.1 Example of Suspension with key info and proof of possession 21
- 1.4.4 Certificate Reactivation..... 21*
 - 1.4.4.1 Example of Reactivation with key info and proof of possession 21

1 Communication between Components

1.1 Communication RA – CA

XML is the document structure used on the web to store data.

To exchange XML documents using web technology we can use for example SOAP or EBXML.

There is a XML schema for key management in XML: XKMS.

In XKMS there are different services;

Key Information: Locate (Query) - Validate (Query).

Key Registration: Register (Prototype with optional "Key Binding").

XKMS does not manage certificates but only keys. So we need to add fields to provide entity information's that will be in certificates.

Most of public key proofs of possession are PKCS10, which is not managed by XKMS.

Thus a schema over XKMS has to be defined for EUPKI.

XKMS is an XML message so it can be digitally signed with XML-dsig framework.

So XML documents will be signed by an RA operator and transported to the CA with a secure protocol in order to authenticate which RA the document comes from (ssl).

1.1.1 Certificate Production

To produce a certificate, a public key must be provided or can be generated by the central KGS. In this case the request must ask for a private key with the "Private" tag.

1.1.1.1 Communication from RA to CA (request)

Service: Register with "Valid" status (Certification request)

Content:

Status

Certificate ID (KeyID)

Certificate Profile to use (X509Template)

Entity Information's (X509Info)

Public key name (KeyName)

Public key to bind (KeyValue or PKCS10) (Optional)

Values to return (Respond)

1.1.1.2 Communication from CA to RA (response)

Service: RegisterResult

Content:

Result (Success, Failure ...)

[In case of success] Answer: Status, KeyID and Requested Values

[In other cases] Reason code (Optional)

1.1.1.3 Example of Register without key binding

```
<Register>
  <Prototype>
    <Status>Valid</Status>
    <KeyID>12345678</KeyID>
    <ds:KeyInfo>
      <ds:KeyName>CN=Alice, OU=D4.2, O=EUPKI, C=EU</ds:KeyName>
    </ds:KeyInfo>
    <X509Template>
      <TemplateName>SampleCertificate</TemplateName>
    </X509Template>
    <X509Info>
      <ValidityInterval>
        <NotBefore>2002-10-23T12:00:00</NotBefore>
        <NotAfter>2003-10-23T12:00:00</NotAfter>
      </ValidityInterval>
      <SubjectAlternativeName>
        <Rfc822Name>Alice@cryptographer.test</Rfc822Name>
      </SubjectAlternativeName>
      <KeyUsage>DigitalSignature</KeyUsage>
      <KeyUsage>NonRepudiation</KeyUsage>
      <KeyUsage>KeyEncipherment</KeyUsage>
      <KeyUsage>DataEncipherment</KeyUsage>
      <NetscapeCertType>SslClient</NetscapeCertType>
      <NetscapeCertType>Smime</NetscapeCertType>
      <CRLDistributionPoint>
        <URI>http://www.eupki.org/ac.crl</URI>
      </CRLDistributionPoint>
      <CRLDistributionPoint>
        <DNSName>ldap.eupki.org</DNSName>
        <DirectoryName>CN=ac, O=EUPKI, C=EU</DirectoryName>
      </CRLDistributionPoint>
    </X509Info>
  </Prototype>
  <Respond>
    <string>KeyName<string>
  </Respond>
</Register>
```

1.1.1.4 Example of Register with key binding (PKCS10)

```
<Register>
  <Prototype>
    <Status>Valid</Status>
    <KeyID>12345678</KeyID>
    <ds:KeyInfo>
      <ds:KeyName>CN=Alice, OU=D4.2, O=EUPKI, C=EU</ds:KeyName>
      <PKCS10>MIIBuTCCASICAQAwETETMBEgAlUEBhMKTHV4ZWlib3VyZzETMBEgAlUECBMKTHV4ZWlib3VyZzEUMBIGAlUEBxMLU2NoaWZmbGFuZ2UxDzANBgNVBAoTBkluVGVjaDELMAkGAlUECXMCSU4xGTAXBgNVBAMTE
```

```

E5pY29sYXMgTEVGRVVWUkUwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMzTrStPIyUyygFTU5p6QjGyLFAx
ncUvwa/i+sK2wY1S6EFYGGd7luGXI3NekVvEEzwIZ+eQ+STB7J7XVik8REubJl6gXlZTrOdVzg6JJtHwbDoTpZ
nB09hGGZUzdyKsnGVIwZ4Un54Z44BZBm5qeOqYMKLK50YCC6ACqdfu/rpAgMBAAGgADANBgkqhkiG9w0BAQQFAA
OBgQBviesKVPfgkGSB2MYIln6yWPGmOjblsdGdzSr/EWbtIAuT75ROZpeKKHzpFuHDC1xpbs0iZYvRACuJyqeqR
HIzomHu6NW7v2+B5CkoP5YsxXswr25fBMRawRckqnzMuZz79G1bi3CtQbh+MbdwvDvvs7DucPgsI7Cn8Fbg214C
9Q==</PKCS10>
  </ds:KeyInfo>
  <X509Template>
    <TemplateName>SampleCertificate</TemplateName>
  </X509Template>
  <X509Info>
    <ValidityInterval>
      <NotBefore>2002-10-23T12:00:00</NotBefore>
      <NotAfter>2003-10-23T12:00:00</NotAfter>
    </ValidityInterval>
    <SubjectAlternativeName>
      <Rfc822Name>Alice@cryptographer.test</Rfc822Name>
    </SubjectAlternativeName>
    <KeyUsage>DigitalSignature</KeyUsage>
    <KeyUsage>NonRepudiation</KeyUsage>
    <KeyUsage>KeyEncipherment</KeyUsage>
    <KeyUsage>DataEncipherment</KeyUsage>
    <NetscapeCertType>SslClient</NetscapeCertType>
    <NetscapeCertType>Smime</NetscapeCertType>
    <CRLDistributionPoint>
      <URI>http://www.eupki.org/ac.crl</URI>
    </CRLDistributionPoint>
    <CRLDistributionPoint>
      <DNSName>ldap.eupki.org</DNSName>
      <DirectoryName>CN=ac, O=EUPKI, C=EU</DirectoryName>
    </CRLDistributionPoint>
  </X509Info>
  <PassPhrase>Pass</PassPhrase>
</Prototype>
<Respond>
  <string>KeyName<string>
</Respond>
</Register>

```

1.1.1.5 Example of positive response for Register

```

<RegisterResult>
  <Result>Success</Result>
  <Answer>
    <Status>Valid</Status>
    <KeyID>12345678</KeyID>
    <ds:KeyInfo>
      <ds:KeyName>CN=Alice, OU=D4.2, O=EUPKI, C=EU</ds:KeyName>
    </ds:KeyInfo>
  </Answer>
</RegisterResult>

```

1.1.1.6 Example of negative response for Register

```

<RegisterResult>
  <Result>Failure</Result>
  <Reason>BadX509TemplateName</Reason>
</RegisterResult>

```

1.1.2 Certificate Renewal

A certificate renewal is technically the same as a certificate request. But only a few values are needed to renew a certificate.

There are two possible renewals:

Only the validity interval change.

The validity interval change with a new public key.

1.1.2.1 Example of Renewal with "valid status"

```
<Register>
  <Prototype>
    <Status>Valid</Status>
    <KeyID>18181818</KeyID>
    <ds:KeyInfo>
      <ds:KeyName>CN=Alice, OU=D4.2, O=EUPKI, C=EU</ds:KeyName>
    </ds:KeyInfo>
    <X509Template>
      <KeyID>12345678</KeyID>
    </X509Template>
    <X509Info>
      <ValidityInterval>
        <NotBefore>2002-10-23T12:00:00</NotBefore>
        <NotAfter>2003-10-23T12:00:00</NotAfter>
      </ValidityInterval>
    </X509Info>
  </Prototype>
  <Respond>
    <string>KeyName<string>
  </Respond>
</Register>
```

1.1.2.2 Example of Renewal with "valid status" and a new Public Key

```
<Register>
  <Prototype>
    <Status>Valid</Status>
    <KeyID>18181818</KeyID>
    <ds:KeyInfo>
      <ds:KeyName>CN=Alice, OU=D4.2, O=EUPKI, C=EU</ds:KeyName>
      <PKCS10>MIIBuTCCASICAQAweTETMBEGAlUEBhMKTHV4ZW1ib3VyZzEUMBIGAlUEBxMLU2NoaWZmbGFuZ2UxDzANBgNVBAoTBkl1VGJjaDELMAkGALUECxmCSU4xGTAXBgNVBAMTE
      VyZzEUMBIGAlUEBxMLU2NoaWZmbGFuZ2UxDzANBgNVBAoTBkl1VGJjaDELMAkGALUECxmCSU4xGTAXBgNVBAMTE
      E5pY29sYXMGTEVGRVWUkUwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMzTrStPIyUyygFTU5p6QjGyLfAX
      ncUvWA/i+sK2wY1S6EFYGGd7luGXl3NekVvEEzwIZ+eQ+STB7J7XVik8REubJl6gXlZTcrOdVzG6JtHwDoTpZ
      nB09hGGZUzdyKsnGVIwZ4Un54Z44BZBm5qeOqYMKLK50YCC6ACqdfu/rpAgMBAAGgADANBgkqhkiG9w0BAQQFAA
      OBgQBVisKVPfgkGSB2MYIln6yWPGmOjblsdGdzSr/EWbtIAuT75ROZpeKKHzpfuHDC1xpbs0iZYvRACu jyqeQR
      HIzomHu6NW7v2+B5CkoP5YsxXswr25fBMRawRckqznMuZz79G1bi3CtQbh+MbdwvDvvs7DucPgsI7Cn8Fbg214C
      9Q==</PKCS10>
    </ds:KeyInfo>
    <X509Template>
      <KeyID>12345678</KeyID>
    </X509Template>
    <X509Info>
      <ValidityInterval>
        <NotBefore>2002-10-23T12:00:00</NotBefore>
        <NotAfter>2003-10-23T12:00:00</NotAfter>
      </ValidityInterval>
    </X509Info>
  </Prototype>
  <Respond>
    <string>KeyName<string>
  </Respond>
</Register>
```

1.1.2.3 Example of Renewal with "valid status" and ask for Private Key generation

```
<Register>
  <Prototype>
    <Status>Valid</Status>
    <KeyID>18181818</KeyID>
    <ds:KeyInfo>
      <ds:KeyName>CN=Alice, OU=D4.2, O=EUPKI, C=EU</ds:KeyName>
    </ds:KeyInfo>
```

```

<X509Template>
  <KeyID>12345678</KeyID>
</X509Template>
<X509Info>
  <ValidityInterval>
    <NotBefore>2002-10-23T12:00:00</NotBefore>
    <NotAfter>2003-10-23T12:00:00</NotAfter>
  </ValidityInterval>
</X509Info>
</Prototype>
<Respond>
  <string>KeyName<string>
  <string>Private<string>
</Respond>

```

1.1.3 Certificate Revocation (or Suspension)

A certificate revocation is technically the same as a certificate request. But only the status of the certificate is changed.

1.1.3.1 Communication from RA to CA (request)

Service: Register with "Invalid" status (Revocation request)

Service: Register with "Suspended" status (Suspension request)

Content:

Status

Certificate ID (KeyID)

Public key to revoke (KeyName or KeyValue)

Values to return (Respond)

1.1.3.2 Communication from CA to RA (response)

Service: RegisterResult

Content:

Result (Success, Failure ...)

[In case of success] Answer: Status, KeyID and Requested Values

[In other cases] Reason code (Optional)

1.1.3.3 Example of Revocation with key info

```

<Register>
  <Prototype>
    <Status>Invalid</Status>
    <KeyID>12345678</KeyID>
    <ds:KeyInfo>
      <ds:KeyName>CN=Alice, OU=D4.2, O=EUPKI, C=EU</ds:KeyName>
      <ds:KeyValue>

```

```

        <ds:RSAKeyValue>
          <ds:Modulus>998/T2PUN8HQlnhf9YIKdMHHGM7HkJwA56UD0a1oYq7E
            fdxSXAidruAszNqBoOqfarJIsfcVKLoblhGnQ/16xw</ds:Modulus>
          <ds:Exponent>AQAB</ds:Exponent>
        </ds:RSAKeyValue>
      </ds:KeyValue>
    </ds:KeyInfo>
  </Prototype>
<Respond>
  <string>KeyName</string>
  <string>KeyValue</string>
</Respond>
</Register>

```

1.1.3.4 Example of positive response for Revocation

```

<RequestResult>
  <Result>Success</Result>
  <Answer>
    <Status>Invalid</Status>
    <KeyID>12345678</KeyID>
    <ds:KeyInfo>
      <ds:KeyName>CN=Alice, OU=D4.2, O=EUPKI, C=EU</ds:KeyName>
      <ds:KeyValue>
        <ds:RSAKeyValue>
          <ds:Modulus>998/T2PUN8HQlnhf9YIKdMHHGM7HkJwA56UD0a1oYq7E
            fdxSXAidruAszNqBoOqfarJIsfcVKLoblhGnQ/16xw</ds:Modulus>
          <ds:Exponent>AQAB</ds:Exponent>
        </ds:RSAKeyValue>
      </ds:KeyValue>
    </ds:KeyInfo>
  </Answer>
</RegisterResult>

```

1.1.4 Certificate Reactivation

A certificate reactivation is technically the same as a certificate revocation, but with a "valid status". Only suspended certificates can be reactivated.

1.1.4.1 Example of Reactivation with key info

```

<Register>
  <Prototype>
    <Status>Valid</Status>
    <KeyID>12345678</KeyId>
    <ds:KeyInfo>
      <ds:KeyName>CN=Alice, OU=D4.2, O=EUPKI, C=EU</ds:KeyName>
      <ds:KeyValue>
        <ds:RSAKeyValue>
          <ds:Modulus>998/T2PUN8HQlnhf9YIKdMHHGM7HkJwA56UD0a1oYq7E
            fdxSXAidruAszNqBoOqfarJIsfcVKLoblhGnQ/16xw</ds:Modulus>
          <ds:Exponent>AQAB</ds:Exponent>
        </ds:RSAKeyValue>
      </ds:KeyValue>
    </ds:KeyInfo>
  </Prototype>
<Respond>
  <string>KeyName</string>
  <string>KeyValue</string>
</Respond>
</Register>

```

1.2 Communication CA – Directory

1.2.1 Certificate Production

There are two possibilities for publishing certificates in a directory:

- The CA is responsible for creating the entries in the directory.

- The CA only updates the certificate attribute in an already populated directory.

In the first case, the CA has to know the Directory Information Tree ("DIT") in order to construct the different entries between the top of the tree and the certificate leaf.

In the second case, the CA has to know how to find the leaf in the tree in order to update the certificate attribute value.

1.2.2 Certificate Renewal

There are two possibilities for certificate renewal:

- The new certificate replaces the old one, so it is an attribute update.

- The new certificate is added in the leaf, it is an attribute insertion.

In the second case one can imagine that the old certificate will be removed when its validity expires.

1.2.3 Certificate Revocation (or Suspension)

For certificate revocation the certificate attribute value is removed.

1.2.4 Certificate Reactivation

As the certificate has been removed by revocation, reactivation is exactly the same as certificate production.

There can be a difference if the CA is responsible for populating the directory, thus certificate reactivation is only an attribute value update and not a value insertion.

1.2.5 CRL Production

It is the same type of process to publish the certificate revocation list.

The CA updates the crl attribute in an already populated directory.

The directory is populated during the CA creation.

1.2.6 Certificate properties in LDAP

Data to publish: X509 Certificate (ASN1 – DER encoded)

Attribute to update: `userCertificate;binary`

Protocol to directory: Light Directory Access Protocol (LDAP)

1.2.7 CRL properties in LDAP

Data to publish: X509 Certificate Revocation List (ASN1 – DER encoded)

Attribute to update: `certificateRevocationList;binary`

Protocol to directory: Light Directory Access Protocol (LDAP)

1.3 Communication CA – Central KGS

1.3.1 Key Generation

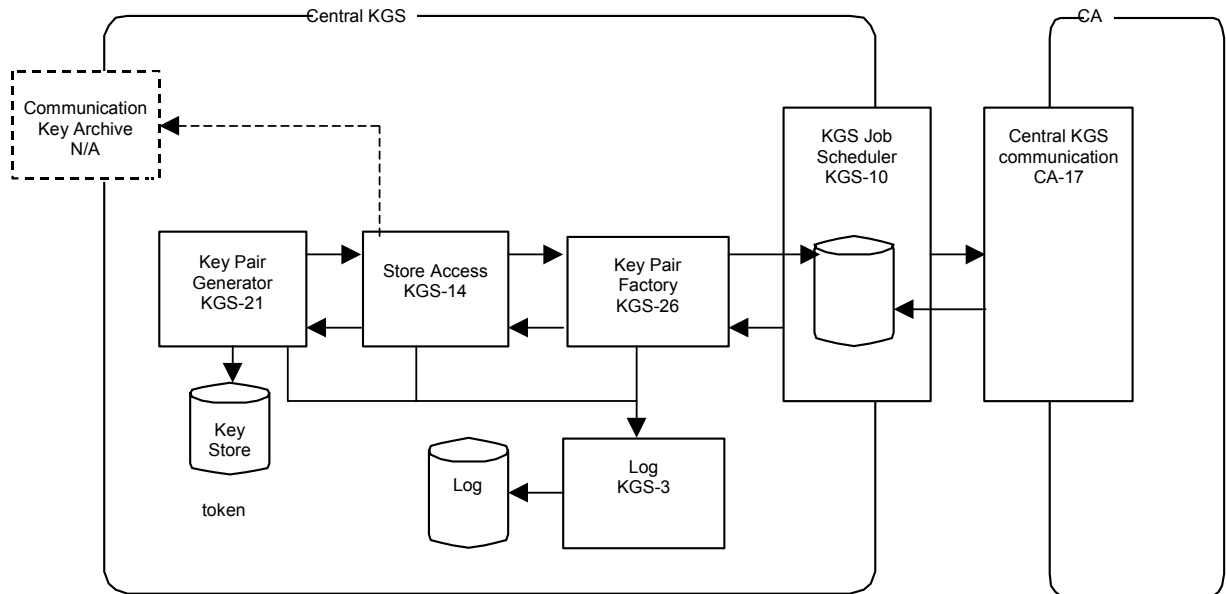


Figure 1: Key Generation Process

- Each called module is logged in the log data base via the Log Module KGS-3
- Each error is logged in the log data base via the Log Module KGS-3.

Each call can be synchron or asynchron:

If the Central KGS communication CA-17 calls a function of KGS Job Scheduler KGS-10 and wait for the response, the process is synchron (on-line mode).

If the Central KGS communication CA-17 calls a function of KGS Job Scheduler KGS-10 and continues without wait for the response, the process is asynchron (batch mode).

1.3.1.1 Key generation request.

The Central KGS Communication module CA-17 sends a message to the KGS10 module to create a key pair :

This message contains this data :

IDENT_KEY_GENERATION_REQUEST	Ident of the key generation request
------------------------------	-------------------------------------

PROFILE_KEY	Profile of the requested key pair
NUMBER_OF_KEY	Number of key pair to generate

Remark: If the number of key to generate is greater than one, all the key pairs will be generated with the same key profile.

The field IDENT_KEY_GENERATION_REQUEST serves to identify the generated key pair with the CA-17 module request :

- Ident

The field PROFILE_KEY defines some parameters for the key pair generator module :

- KEY_TYPE
- TOKEN
- Public key format (format of the data key value to return or PKCS10)
- List of extension + value (optional: if the public key format is PKCS10)

The field KEY_TYPE indicates the type of key pair asked by the Central KGS Communication CA-17 module :

- Key_algo: Type of algorithm to use (RSA)
- Key_size: Size of the key (compatible with the algorithm: 1024, 2048)

The field TOKEN indicates how to store the private key :

- Type of token (file, smartcard)
- Format to store (PKCS11)

This message received by the KGS-10 module is wrote in a job table. Then the key pair generation process starts.

When the key pair generation process ends, it updates the job table with the result.

If there is no error, the KGS Job Scheduler KGS-10 sends n messages for each key pair generated.

If a error occurs while the key generation process, no keys and no responses are returned to the Central KGS Communication CA-17. The administrator can restart the job later.

This message can be in XML format like:

```

<?XML version="1.0" encoding="UTF-8" ?>
<KEY_GENERATION_REQUEST>
  <IDENT_KEY_GENERATION_REQUEST>
    <IDENT>...</IDENT>
  </IDENT_KEY_GENERATION_REQUEST>
  <PROFILE_KEY>
    <KEY_TYPE>
      <KEY_ALGO>...</KEY_ALGO>
      <KEY_SIZE>...</KEY_SIZE>
    </KEY_TYPE>
    <TOKEN>
      <TOKEN_TYPE>...</TOKEN_TYPE>
      <STORE_FORMAT>...</STORE_FORMAT>
    </TOKEN>
    <KEY_FORMAT>...</KEY_FORMAT>
    <ATTRIBUTE_LIST>
      <DN>...</DN>
      ...
    </ATTRIBUTE_LIST>
  </PROFILE_KEY>
  <NUMBER_OF_KEY>...</NUMBER_OF_KEY>
</KEY_GENERATION_REQUEST>

```

1.3.1.2 **Key generation request completed.**

The KGS Job Scheduler module KGS-10 sends messages to the CA-17 module to return the result of the key pair generation.

This message contains this data:

IDENT_KEY_GENERATION_REQUEST_COMPLETED	Ident of the generation key pair.
PUBLIC_KEY	Contains the public key

The field IDENT_KEY_GENERATION_REQUEST_COMPLETED will serve to identify the generated key with the CA-17 module request :

- IDENT_KEY_GENERATION_REQUEST
- Number of the key (1 to NUMBER_OF_KEY)

The field PUBLIC_KEY contains the public key in the format asked in the initial request :

- Key format
- Key data

The field IDENT_KEY_GENERATION_REQUEST identifies the generate key pair with the CA-17 module request :

- Ident

This message can be in XML format like:

```
<?XML version="1.0" encoding="UTF-8" ?>
<KEY_GENERATION_REQUEST_COMPLETED>
  <IDENT_KEY_GENERATION_REQUEST_COMPLETED>
    <IDENT_KEY_GENERATION_REQUEST>
      <IDENT>...</IDENT>
    </IDENT_KEY_GENERATION_REQUEST>
    <KEY_NUMBER>...</KEY_NUMBER>
  <IDENT_KEY_GENERATION_REQUEST_COMPLETED>
  < PUBLIC_KEY>
    <KEY_FORMAT>...</KEY_FORMAT>
    <KEY_DATA>...</KEY_DATA>
  </ PUBLIC_KEY>
</KEY_GENERATION_REQUEST_COMPLETED>
```

1.3.2 Certificate Storage

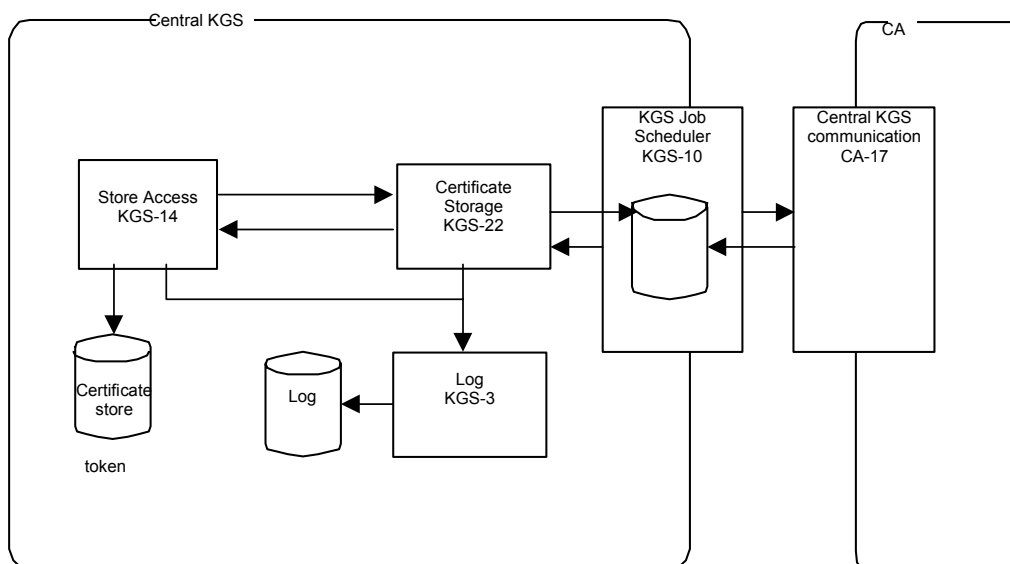


Figure 2: Storage Certificate Process

1.3.2.1 Certificate storage request.

The Central KGS communication CA-17 module sends messages to the KGS-10 module to store a certificate on token.

This message contains this data:

IDENT_KEY_GENERATION_REQUEST_COMPLETED	Ident of the generation key pair
CERTIFICATE	Certificate to store
TOKEN	Token
DISTRIB_PASS	Useful to distribute the pass phrase to use (for read the PKCS12 or a pin code for a smartcard)

The field IDENT_KEY_GENERATION_REQUEST_COMPLETED identifies the private key which correspond to the certificate to store :

- IDENT_KEY_GENERATION_REQUEST
- Number of the key (1 to NUMBER_OF_KEY)

The field CERTIFICATE contains the certificate to store :

- Format (X509 v3, RFC2459)
- Data certificate

The field TOKEN indicates how to store and send the certificate :

- Type of token (file, smartcard)
- Format to store (PKCS12, PKCS7)
- Address (depend of the type of token) where is sent or downloaded the certificate

The field DISTRIB_PASS is used to distribute the pass phrase which protect the certificate :

- mode (mail, letter, web) - optional
- address (mail address, postal address, url) - optional
- pass phrase (pass phrase for the PKCS12) - optional

Mode and address exist if the pass phrase which protect the pkcs12 or the smartcard is generated in central and must be transmit to a user. In this case, pass phrase is null.

Pass phrase exists if the pass phrase which protect the pkcs12 or the smartcard is transmit by the requester. In this case, mode and address are null. This pass phrase must be used by the KGS to protect the PKCS12 or the the smartcard.

This message received by the KGS-10 module is written in a job table. Then the certificate storage process starts.

When the certificate storage process ends, it updates the job table with the result.

If there is no error, the KGS Job Scheduler KGS-10 sends a messages to the CA-17 module to inform it.

If a error occurs while the certificate storage process, no responses are returned to the Central KGS Communication CA-17. The administrator can restart the job later.

This message can be write in XML format like:

```
<?XML version="1.0" encoding="UTF-8" ?>
<CERTIFICATE_STORAGE_REQUEST>
  <IDENT_KEY_GENERATION_REQUEST_COMPLETED>
    <IDENT_KEY_GENERATION_REQUEST>
      ...
    <IDENT_KEY_GENERATION_REQUEST>
      <KEY_NUMBER>...<KEY_NUMBER>
    <IDENT_KEY_GENERATION_REQUEST_COMPLETED>
  <CERTIFICATE>
    <FORMAT>...<FORMAT>
    <DATA_CERTIFICATE>...<DATA_CERTIFICATE>
  <CERTIFICATE>
  <TOKEN>
    <TOKEN_TYPE>...</TOKEN_TYPE>
    <STORE_TYPE>...</STORE_TYPE>
    <ADRESS>...</ADRESS>
  <TOKEN>
  <DISTRIB_PASS>
    <MODE>...<MODE>
    <ADDRESS>...<ADDRESS>
    <PASS_PHRASE>...<PASS_PHRASE>
  <DISTRIB_PASS>
</CERTIFICATE_STORAGE_REQUEST>
```

1.3.2.2 ***Certificate storage completed.***

The Central KGS Job Scheduler KGS-10 sends a message to the CA-17 to indicate that a certificate are correctly store.

This message contains this data:

IDENT_KEY_GENERATION_REQUEST_COMPLETED	Ident of the generation key pair
--	----------------------------------

The field IDENT_KEY_GENERATION_REQUEST_COMPLETED will serve to identify the generated key with the CA-17 module request :

- IDENT_KEY_GENERATION_REQUEST
- Number of the key (1 to NUMBER_OF_KEY)

This message can be written in XML format like:

```
<?XML version="1.0" encoding="UTF-8" ?>
<CERTIFICATE_STORAGE_COMPLETED>
  <IDENT_KEY_GENERATION_REQUEST_COMPLETED>
    <IDENT_KEY_GENERATION>...< IDENT_KEY_GENERATION >
    <NUMBER_KEY>...< NUMBER_KEY >
  </ IDENT_KEY_GENERATION_REQUEST_COMPLETED >
</CERTIFICATE_STORAGE_COMPLETED>
```

1.4 Communication User KGS – CA

1.4.1 Certificate Production

Communication between User KGS and CA consists only of Key Binding and require first a RA registration.

In that case a "One Time Password" is needed to associate the user with the RA registration.

In XKMS this OTP is stored in a "KeyBindingAuth":

Auth = HMAC-SHA1 (OTP, 0x1)

KeyBindingAuth = HMAC-SHA1 (KeyBinding, Auth)

1.4.1.1 Communication from User KGS to CA

Service: Register with "Valid" status (Key Binding)

Content:

Status

Certificate ID (KeyID)

Public key to bind (KeyInfo or PKCS10)

Values to return (Respond)

1.4.1.2 Communication from CA to User KGS (response)

Service: RegisterResult

Content:

Result (Success, Failure ...)

[In case of success] Answer: Status, KeyID and Requested Values

[In other cases] Reason code (Optional)

1.4.1.3 Example of Key Binding (KeyValue, ProofOfPossession and KeyBindingAuth)

```
<Register>
  <Prototype Id="keybinding">
    <Status>Valid</Status>
    <KeyID>12345678</KeyID>
    <ds:KeyInfo>
      <ds:KeyName>CN=Alice, OU=D4.2, O=EUPKI, C=EU</ds:KeyName>
      <ds:KeyValue>
        <ds:RSAKeyValue>
          <ds:Modulus>
            998/T2PUN8HQlnhf9YIKdMHHGM7HkJwA56UD0aloYq7E
            fdxSXAidruAszNqBoOqfarJIsfcVKLoblhGnQ/l6xw
          </ds:Modulus>
          <ds:Exponent>AQAB</ds:Exponent>
        </ds:RSAKeyValue>
      </ds:KeyValue>
    </ds:KeyInfo>
    <PassPhrase>Pass</PassPhrase>
  </Prototype>
  <AuthInfo>
    <AuthUserInfo>
      <ProofOfPossession>
        <ds:Signature URI="#keybinding"
          [RSA-Sign (KeyBinding, Private)] />
      </ProofOfPossession>
      <KeyBindingAuth>
        <ds:Signature URI="#keybinding"
          [HMAC-SHA1 (KeyBinding, Auth)] />
      </KeyBindingAuth>
    </AuthUserInfo>
  </AuthInfo>
  <Respond>
    <string>KeyName</string>
    <string>KeyValue</string>
    <string>RetrievalMethod</string>
  </Respond>
</Register>
```

1.4.1.4 Example of positive response for Key Binding

```
<RegisterResult>
  <Result>Success</Result>
  <Answer>
    <Status>Valid</Status>
    <KeyID>12345678</KeyID>
    <ds:KeyInfo>
      <ds:KeyName>CN=Alice, OU=D4.2, O=EUPKI, C=EU</ds:KeyName>
    </ds:KeyInfo>
  </Answer>
</RegisterResult>
```

1.4.2 Certificate Renewal

The certificate renewal can be send by the User KGS if a new public is generated. In that case it is exactly the same as the certificate production but with a new OTP provided by the RA.

1.4.3 Certificate Revocation (or Suspension)

The User can revoke himself with a proof of possession.

1.4.3.1 Example of Suspension with key info and proof of possession

```
<Register>
  <Prototype Id="keybinding">
    <Status>Suspended</Status>
    <KeyID>12345678</KeyID>
    <ds:KeyInfo>
      <ds:KeyName>CN=Alice, OU=D4.2, O=EUPKI, C=EU</ds:KeyName>
      <ds:KeyValue>
        <ds:RSAKeyValue>
          <ds:Modulus>998/T2PUN8HQlnhf9YIKdMHHGM7HkJwA56UD0a1oYq7E
            fdxSXAidruAszNgBoOqfarJIsfcVKLoblhGnQ/16xw</ds:Modulus>
          <ds:Exponent>AQAB</ds:Exponent>
        </ds:RSAKeyValue>
      </ds:KeyInfo>
    </Prototype>
  <AuthInfo>
    <AuthUserInfo>
      <ProofOfPossession>
        <ds:Signature URI="#keybinding"
          [RSA-Sign (KeyBinding, Private)] />
      </ProofOfPossession>
    </AuthUserInfo>
  </AuthInfo>
  <Respond>
    <string>KeyName</string>
    <string>KeyValue</string>
  </Respond>
</Register>
```

1.4.4 Certificate Reactivation

The User can reactivate himself with a proof of possession.

1.4.4.1 Example of Reactivation with key info and proof of possession

```
<Register>
  <Prototype Id="keybinding">
    <Status>Valid</Status>
    <KeyID>12345678</KeyID>
    <ds:KeyInfo>
      <ds:KeyName>CN=Alice, OU=D4.2, O=EUPKI, C=EU</ds:KeyName>
      <ds:KeyValue>
```

```
        <ds:RSAKeyValue>
          <ds:Modulus>998/T2PUN8HQlnhf9YIKdMHHGM7HkJwA56UD0a1oYq7E
            fdxSXAidruAszNqBoOqfarJIsfcVKLoblhGnQ/l6xw</ds:Modulus>
          <ds:Exponent>AQAB</ds:Exponent>
        </ds:RSAKeyValue>
      </ds:KeyValue>
    </ds:KeyInfo>
  </Prototype>
</AuthInfo>
  <AuthUserInfo>
    <ProofOfPossession>
      <ds:Signature URI="#keybinding"
        [RSA-Sign (KeyBinding, Private)] />
    </ProofOfPossession>
  </AuthUserInfo>
</AuthInfo>
<Respond>
  <string>KeyName</string>
  <string>KeyValue</string>
</Respond>
</Register>
```

2 Annexe

XKMS schema (to be completed)

```
<schema targetNamespace="http://www.xkms.org/schema/xkms-2001-01-20"
  xmlns:xkms="http://www.xkms.org/schema/xkms-2001-01-20"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns="http://www.w3.org/2000/10/XMLSchema"
  attributeFormDefault="unqualified" elementFormDefault="qualified">

  <import namespace="http://www.w3.org/2000/09/xmldsig#"
    schemaLocation="http://www.w3.org/TR/2000/CR-xmldsig-core-20001031/\
xmldsig-core-schema.xsd"/>

  <element name="Register">
    <complexType>
      <sequence>
        <element name="Prototype" type="xkms:KeyBindingType"/>
        <element name="AuthInfo" minOccurs="0">
          <complexType>
            <choice>
              <element name="AuthUserInfo" type="xkms:AuthUserInfoType"/>
              <element name="AuthServerInfo" type="xkms:AuthServerInfoType"/>
            </choice>
          </complexType>
        </element>

        <element name="Respond" minOccurs="0" >
          <complexType>
            <sequence>
              <element name="string" type="string" minOccurs="0" maxOccurs="unbounded"/>
            </sequence>
          </complexType>
        </element>
      </sequence>
    </complexType>
  </element>

  <complexType name="AuthUserInfoType">
    <sequence>
      <element name="ProofOfPossession" minOccurs="0">
        <complexType>
          <sequence>
            <element ref="ds:Signature" minOccurs="0"/>
          </sequence>
        </complexType>
      </element>

      <element name="KeyBindingAuth" minOccurs="0">
        <complexType>
          <sequence>
            <element ref="ds:Signature" minOccurs="0"/>
          </sequence>
        </complexType>
      </element>
      <element name="PassPhraseAuth" type="string" minOccurs="0"/>
    </sequence>
  </complexType>

  <complexType name="AuthServerInfoType">
    <sequence>
      <element name="KeyBindingAuth" minOccurs="0">
        <complexType>
          <sequence>
            <element ref="ds:Signature" minOccurs="0"/>
          </sequence>
        </complexType>
      </element>
      <element name="PassPhraseAuth" type="string" minOccurs="0"/>
    </sequence>
  </complexType>

  <complexType name="KeyBindingType">
    <sequence>
```

```

<element name="TransactionID" type="string" minOccurs="0"/>
<element name="Status" type="xkms:AssertionStatus"/>
<element name="KeyID" type="uriReference" minOccurs="0"/>
<element ref="ds:KeyInfo" minOccurs="0"/>
<element name="PassPhrase" type="string" minOccurs="0"/>
<element name="ProcessInfo" minOccurs="0">
  <complexType>
    <sequence minOccurs="0" maxOccurs="unbounded">
      <any namespace="##other"/>
    </sequence>
  </complexType>
</element>
<element name="ValidityInterval" type="xkms:ValidityIntervalType"
minOccurs="0"/>
  <element name="KeyUsage" type="xkms:KeyUsageType" minOccurs="0"
maxOccurs="unbounded"/>
</sequence>
<attribute name="Id" type="ID" use="optional"/>
</complexType>

<simpleType name="AssertionStatus">
  <restriction base="string">
    <enumeration value="Valid"/>
    <enumeration value="Invalid"/>
    <enumeration value="Indeterminate"/>
  </restriction>
</simpleType>

<simpleType name="KeyUsageType">
  <restriction base="string">
    <enumeration value="Encryption"/>
    <enumeration value="Signature"/>
    <enumeration value="Exchange"/>
  </restriction>
</simpleType>

<complexType name="ValidityIntervalType">
  <sequence>
    <element name="NotBefore" type="timeInstant" minOccurs="0"/>
    <element name="NotAfter" type="timeInstant" minOccurs="0" />
  </sequence>
</complexType>

<element name="RegisterResult">
  <complexType>
    <sequence>
      <element name="Result" type="xkms:ResultCode"/>
      <element name="Answer">
        <complexType>
          <sequence>
            <element name="KeyBinding" type="xkms:KeyBindingType"
minOccurs="0" maxOccurs="unbounded"/>
          </sequence>
        </complexType>
      </element>
      <element name="Private" type="string" minOccurs="0" />
    </sequence>
  </complexType>
</element>

<simpleType name="ResultCode">
  <restriction base="string">
    <enumeration value="Success"/>
    <enumeration value="NoMatch"/>
    <enumeration value="NotFound"/>
    <enumeration value="Incomplete"/>
    <enumeration value="Failure"/>
    <enumeration value="Refused"/>
    <enumeration value="Pending"/>
  </restriction>
</simpleType>

<element name="Validate">
  <complexType>
    <sequence>
      <element name="Query" type="xkms:KeyBindingType"/>
      <element name="Respond" minOccurs="0" />
    </sequence>
  </complexType>

```

```
<complexType>
  <sequence>
    <element name="string" type="string" minOccurs="0" maxOccurs="unbounded" />
  </sequence>
</complexType>
</element>
</sequence>
</complexType>
</element>

<element name="ValidateResult">
  <complexType>
    <sequence>
      <element name="Result" type="xkms:ResultCode" />
      <element name="Answer" minOccurs="0">
        <complexType>
          <sequence>
            <element name="KeyBinding" type="xkms:KeyBindingType"
              minOccurs="0" maxOccurs="unbounded" />
          </sequence>
        </complexType>
      </element>
    </sequence>
  </complexType>
</element>

<element name="Locate">
  <complexType>
    <sequence>
      <element name="TransactionID" type="string" minOccurs="0" />
      <element name="Query">
        <complexType>
          <sequence>
            <element ref="ds:KeyInfo" />
          </sequence>
        </complexType>
      </element>
      <element name="Respond" minOccurs="0">
        <complexType>
          <sequence>
            <element name="string" type="string" minOccurs="0" maxOccurs="unbounded" />
          </sequence>
        </complexType>
      </element>
    </sequence>
  </complexType>
</element>

<element name="LocateResult">
  <complexType>
    <sequence>
      <element name="TransactionID" type="string" minOccurs="0" />
      <element name="Result" type="xkms:ResultCode" />
      <element name="Answer" minOccurs="0">
        <complexType>
          <sequence>
            <element ref="ds:KeyInfo" minOccurs="0" maxOccurs="unbounded" />
          </sequence>
        </complexType>
      </element>
    </sequence>
  </complexType>
</element>
</schema>
```