# Towards the Homogeneous Access and Use of PKI Solutions: Design and Implementation of a WS-XKMS Server [1]

Jose M. Alcaraz Calero [a] Gabriel López Millán [a]
Gregorio Martínez Pérez *,[a] Antonio F. Gómez Skarmeta [a]

[a] *Departamento de Ingeniería de la Información y las Comunicaciones, University of Murcia, Campus de Espinardo, s/n, 30.071, Murcia, Spain*

**Abstract**

Nowadays, there exists certain important scenarios where different WS-* security related protocols and technologies are being used, such as e-commerce, resource control, or secure access to grid nodes. Additionally, most of these scenarios require the interaction with a trust management infrastructure (such as a PKI -Public Key Infrastructure-), usually to validate the digital certificates provided by communication peers belonging, in most cases, to different administrative domains. For doing this with WS-enabled technologies the W3C proposed the XKMS (XML Key Management Specification) standard a few years ago. However, few implementations exist so far of this standard, and most of them with important limitations. This paper presents an open-source WS-enabled implementation of the XKMS standard named Open XKMS, certain key scenarios where it can be used and the details of how it has been designed and implemented. This paper tries to motivate and foster the use of the XKMS standard and describe a software solution that can help to designers and developers of WS-based security scenarios.

*Key words:* XKMS Service, Secure Web Services, Homogeneous access to PKI services

* Corresponding author. Tel. +34 968367646 Fax. +34 968364151
*Email addresses:* jmalcaraz@um.es (Jose M. Alcaraz Calero), gabilm@um.es (Gabriel López Millán), gregorio@um.es (Gregorio Martínez Pérez), skarmeta@um.es (Antonio F. Gómez Skarmeta).

[1] This work has been partially funded by the MISTRAL (Middleware de gestión de Identidades de Seguridad en TRansacciones electrónicAs basado en código Libre, TIC-INF 07/01-0003) project done in collaboration with the company Bahía IT (http://www.bahiait.es/). Thanks also to the Spanish Ministerio de Educación y Ciencia for sponsoring the research activities under the grant AP2006-4150 of the

## 1 Introduction

W3C has defined a standard named XKMS (XML Key Management Specification) to highly reduce the cost in the client side by integrating the logic of the certificate management (and all the related difficulties) in one single XKMS server that can be programmed once and used by any device supporting basic XML and symmetric/asymmetric key processing. With these two elements and certain standard messages most of the PKI interaction processes can be done. Thus, small devices can integrate digital certificate support with minimum computing and storage requirements; also, integration of different PKI solutions can be done easily as the interaction with all of them is done in the same way through an intermediate server, being this server the one supporting the difficulties of processing any of the tasks related with the digital certificate lifecycle.

### *1.1 Overview of Web Service Security model*

In the last years, a big number of proposals related to web services infrastructures [22] have made appearance in order to standardize and homogenize the data exchange among the different services. Those are mainly coming from organizations such as OASIS and W3C.

Following a top-down approach, we can find first technologies for describing high level services like WS-Federation, WS-SecureConversation or WS-Authorization. Those proposals as focused on the representation of data needed to federate security realms, to provide secure communications between services, and to add authorization mechanisms to access control scenarios respectively.

Next, several proposals appear in order to help the previous ones to define general policies for web services (WS-Policy), to establish trust requirements among services (WS-Trust), or to specify security policies (WS-Privacy).

In the lower level we can find those proposals related to the Web Service Security model. Among then, we can distinguish between those related to protect communication messages, such as XML Digital Signature (XML-DS) or XML Encryption (XML-ENC), and those related to provide security services (au-

FPU program. Authors would also like to thank Silvan Krebs and Sebastian Fiechter from the Berne University of Applied Sciences for their valuable contributions to the OpenXKMS implementation. Authors would finally like to thank the Funding Program for Research Groups of Excellence with code 04552/GERM/06 granted by the Fundación Séneca.

thentication, authorization, access control, key management, etc) to the whole set of described technologies, such as SAML [11], XACML [23] or XKMS [24].

We can see how upper layer technologies are strongly defined over a security infrastructure provided by the lower ones, that is, we are continuously defining new proposals based on the existence of technologies such as SAML, XACML or XKMS. However, whereas SAML and XACML are getting more and more presence in the current security solutions, such as Liberty Alliance [10] or Shibboleth [27], the incorporation of XKMS is getting more slowly, mainly due to the fact that those technologies need first to be consolidated. As next section shows, several scenarios can be significantly improved through the use of this proposal.

### *1.2 Overview of XKMS*

The XKMS specification [24] was proposed by the W3C consortium with the objective to define the needed protocols to distribute and register public keys between user applications and PKI services, using the XML-DS and XML-ENC protocols to provide a secure information exchange. The main objectives of XKMS are the following:

- To define an abstract layer between applications and PKI services.
- To avoid applications be aware of complex syntaxes and semantics of PKI protocols, using simple and understandable XML protocols.
- To move complexity from user applications to PKI infrastructure in order to achieve easy and swift applications.

XKMS is mainly composed by two services X-KISS and X-KRSS. The former stands for *XML Key Information Service Specification*, the latest stands for *XML Key Registration Service Specification (X-KRSS)* and both are based in the concept of key binding. A key binding is the association between personal information (name, email or IP addresses, etc) and a public and private key pair. This concept is described by means of a *KeyInfo* element defined by XML-DS [15].

X-KISS offers the location and validation services. The *Locate* service is used to resolve *KeyInfo* elements to additional information required to validate or decrypt a secure message. For example, Alice may use the *Locate* service to recover the Bob's public key to verify a signed document sent by him. This document includes the *KeyInfo* element. The Validation service extends *Locate* including information about the validation of the key binding (validity period, revocation status, digital signature, etc).

X-KRSS defines the key binding management services. The *Register* service is used to assert key binding information to a public key pair. It also includes the *Reissue*, *Revocation* and *Key Recovery* services covering the whole key binding management life cycle .

XKMS also permits both compound requests, which permit one or more X-KISS and K-KRSS requests to be made at the same time, and asynchronous processing. When the service is not able to complete the request immediately notifies the client that the response is not ready, and subsequent requests for the pending service will be sent by this client.

### *1.3 Paper structure*

This paper presents the design of an XKMS service and the main issues of one implementation of this standard named Open XKMS. The paper is structured as follows. First, we introduce the main scenarios that have motivated this work. In the next section the design of a XKMS service is presented together with the different modules composing such design. Then, the implementation details are presented together with some performance results. Then, the paper presents the related work, and finalises with the main conclusions and future work.

## 2 Motivation and use case scenarios

This section briefly introduces the main scenarios that have motivated this work. They show that both emerging and well-known technologies need to make use of a standard interface to access security services, being this interface able to abstract the technology details for different security technologies.

The common infrastructure for all these scenarios is depicted in figure 1. For each scenario we can find different client technologies that could make use of a XKMS client to request some of the traditional PKI services (location, validation, certification, etc). For each service, the XKMS server offers a range of protocols able to deal with the client request, but hiding protocol details.

For example, if Alice receives a signed email from Bob, where Alice belongs to organization A and Bob belongs to B, Alice's email client could make use of the XKMS client to send a validation request for Bob's certificate. In this case, the XKMS server would make use of one of the pre-configured validation services (OCSP, SCVP, etc). The XKMS server delegates the path building and validation processes [12,16,17] to those services which will be responsible

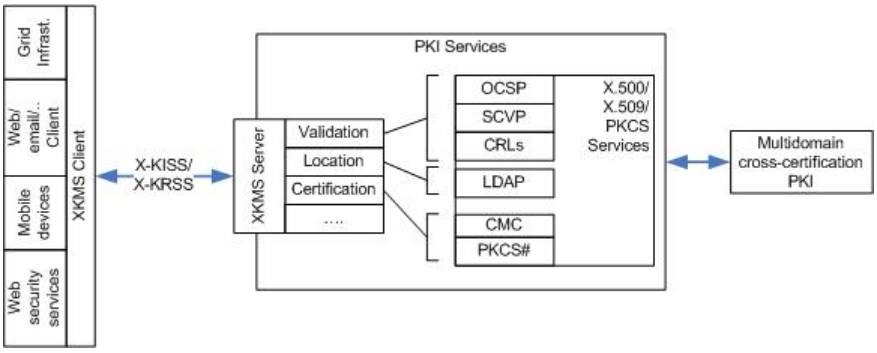


Fig. 1. Overview of a XKMS-based general scenario

to build the certification chain from Bob's certificate to the corresponding trust anchor trusted by A, depending on the kind of certification model established between A and B (peer-to-peer, hierarchical or bridge). The following scenarios follow a similar approach, taking into account the kind of XKMS client, the set of technologies involved and the PKI service(s) being requested.

### *2.1 Scenario 1: e-commerce*

Nowadays, e-commerce scenarios involve from end users, service providers, intermediate brokers to financial entities. On one hand, those entities rely on the use of standard PKI technologies such as peer-to-peer or Bridge CA solutions [2], [3]. On the other hand, service providers are making use of Web Service technologies to provide a standard interface to the end users appliances.

As described before, the main target of XKMS is to provide a homogenous interface to those service providers, brokers and companies that want to make use of these certification services regardless the kind of security technology deployed in each scenario (X.509, SAML, etc). As this scenario may imply different security domains, certificate path validation and building services [20] can be needed.

### *2.2 Scenario 2: Mobile platforms*

Roaming users making use of mobile phones or PDAs are requiring providers Internet access through the use of WIFI or GPRS technologies and, therefore, the use of the security services.

It would be very useful to provide a light and standard interface, like the one defined by XKMS, to offer a common security service to these devices without including significant overhead [21]. Making use of a XKMS client application, those devices can access to services such as key validation, location, etc.

### *2.3 Scenario 3: XML communication services*

New emerging services, such as ebMS (ebXML Messaging Services) or XML-RPC (XML-Remote Procedure Call), provide peers the possibility to exchange data messages protected through the use of XML Digital Signature or XML Encryption technologies. These services will require both location and validation services to accept or reject protected messages coming from the different communication peers.

XKMS provides simple and powerful interface to those clients, ranging from the registration process to get new security credentials to those location and validation services.

### *2.4 Scenario 4: Identity Federations*

The peak of identity federations is taking organizations to provide homogenous and standard interfaces to access protected services and authorities such as identity and attribute providers. Clear examples of those federations are Incommon [4] or SWITCH [5]. Most of those federations, related to protect Web Service resources, make use of solutions like Shibboleth, which is based on technologies such as SAML or XACML. As described in section 1, one of the main XKMS target scenarios are the ones based on those XML-based emerging technologies.

### *2.5 Scenario 5: Grid infrastructures*

Another interesting scenario where we can see the need of XKMS is the Globus Grid Computing project. Every GT node has a component, named Globus Security Infrastructure (GSI), which is in charge of offering the required services to achieve the security issues. GSI needs the use the standards such as CRLs, OCSP, SCVP, etc., which can be different for each organization belonging to the Grid infrastructure.

GSI can make use of the XKMS validation service instead of specific standards to get validity status of X.509 certificates, independently of the security technology of each organization.

## 3 Design of a WS-XKMS solution

The main driving objectives of the design of this XKMS solution were: first, to offer a web service-based interface; second, to design this solution conferring high usability in the major number of scenarios; and third, to allow the concurrent use of different PKI technologies, such as X.509, Kerberos, SAML, etc, by providing an interface to PKI providers based on the use of XKMS connectors.

On the one hand, web service access to a XKMS server provides an entry point for all web service technologies to PKI infrastructures. Therefore, the XKMS server could be seen as a gateway between web services and traditional PKI solutions. Moreover, high-level web service technologies related with security (and described in section 1), such as WS-Federation or WS-Authorization, are susceptible to make use of PKI infrastructures.

On the other hand, a web service interface can be seen as a blender engine with regards to scenarios where several PKIs and, maybe, based on different technologies, provide access to security services. This can be the case of Bridge CA or e-commerce scenarios described in section 2. In this context, this XKMS design provides an homogeneous access to all the different PKI existing solutions.

Web service-based access implemented under different web service technologies such as Apache SOAP [7] or JAX-WS [18], is supported by the division of the XKMS architecture in modules. Our design confers to the XKMS architecture an standard access interface for WS-* based technologies through a *WS-ExternalConnector* module. Other module, named *PkiForXKMS*, contributes with an easy form to add new underline PKI implementations or technologies through the use of *connectors*. In the middle, the *XKMSEngine* puts in contact these two modules, obtaining the users requests, selecting the appropriate *connector*, and providing the final answer.

The idea of modular design has been chosen to give priority to the adaptation of the server to new requirements providing at same time extensibility, doing possible the addition of new web service implementations and new future PKI services and technologies. It also provides reusability and scalability. The result has been a high scalable solution for PKI access in the WS-* context.

The modules are depicted in Figure 2 and are explained in details in the next subsections. Some more details can be found in the implementation section, which complements the description provided in this section.

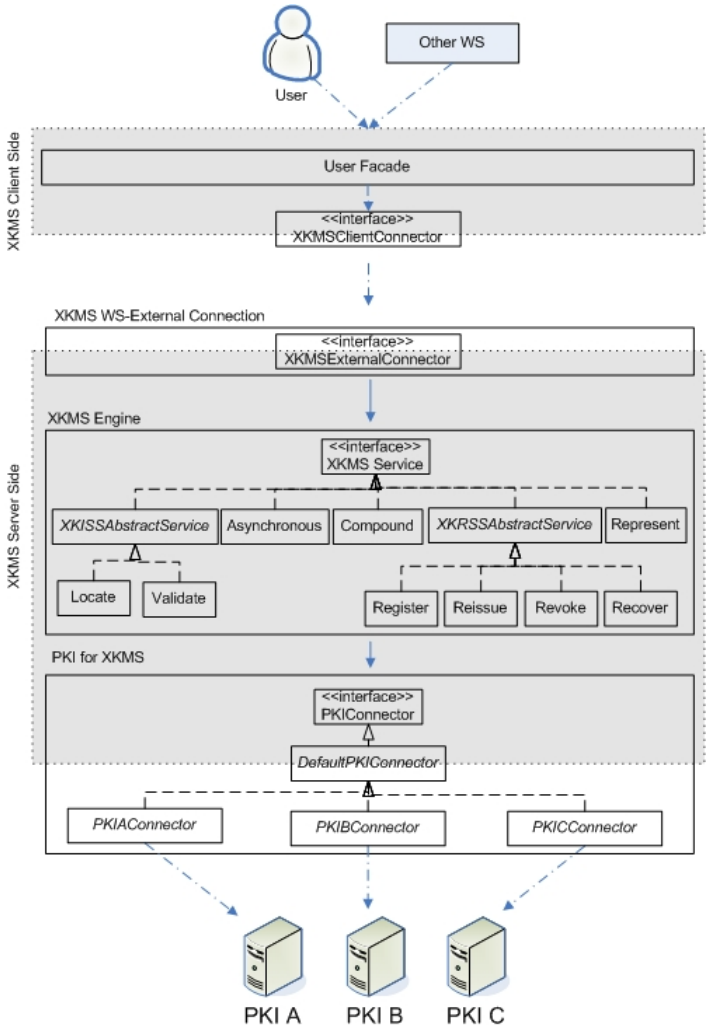


Fig. 2. XKMS server design

### *3.1 WS-ExternalConnection*

This module offers extensibility to the XKMS architecture in order to provide an easy way to add different underline transport protocols and different web service frameworks for accessing the XKMS server. In this context, a transport protocol specifies the way in which messages are delivered to the destination. On these transport protocols such as HTTP, SOAP 1.1 or SOAP 1.2, web services frameworks such as Apache SOAP [7], Apache AXIS [8], JAX-WS [18] or 'ASP .NET web service' could been deployed in order to provide a remote procedure call technology in a distributed environment like this.

The main target of this module is to hide the implementation details of one of previous web service framework alternatives. This objective implies that the module needs to cover both client and server side offering a facade to users to avoid this kind of details related with the transport and web services

technologies in use to provide the certification services to clients, get all the requests, and send back the appropriate response. In this sense, this module could be seen as a distributed transport layer witch directly connect client and server among networks and web services avoiding technological details.

In figure 3, the reader can see in a sequence diagram how the user creates a request with independence of the communication technology in use between the client and server sending the request to *WS-ExternalConnector* module. Then the *WS-ExternalConnector* module applying a unmarshaling operation obtains the original request from the specific format associated to the selected web service framework. Then, it passes this request to others modules for processing it. Analogously after processing the request, the response need to be deliver to the client; in this case the marshaling operation encapsules the original response with the implementation details associated to the web service framework chosen and send it to the client that de-encapsules it offering to the end-user the original response. Because of this module, all these steps can be done without any knowledge about the underline communication method used by both the end-user and the XKMS engine.

It is important to note that one of the main design restrictions of this module is the impossibility to change any part of the message received from (or sent to) the client as this message can be signed or ciphered and any minimum transformation would result in a validation error in the end receiver.

The inclusion of the WS-ExternalConnection module in the design of the XKMS server contributes with a clear division between the XKMS engine and the different access methods to the server allowing simultaneous heterogeneous accesses to the same engine.

In our implementation two different technologies (Apache SOAP and JAX-WS) could be used in order to access to the same XKMS engine. This module in the JAX-WS access has been implemented using Document Object Model (DOM)-based technologies for preparing the delivering of messages. In the access by Apache SOAP a Binding based technology as JAXB is used to prepare the delivering of messages.

### *3.2 XKMSEngine*

The *XKMSEngine* module is in charge of doing the real processing of the XKMS requests and responses. As such it provides most of the functionalities defined by the XKMS specification. In this sense, it is able to:

- Verify the XML digital signature of the requests.
- Offers all the X-KISS and X-KRSS services.

- Build the response to a given request, signing it when needed.
- Manage request being processed in asynchronous mode, as they needed to be stored in a temporal repository until they are processed.
- Process compound requests appropriately doing many different requests in parallel and combining all the results in a single response message.
- Select the appropriate PKI connector to a given request.
- Support represent mode for all the XKMS services witch divides the access to XKMS server into two phases challenge protocols.

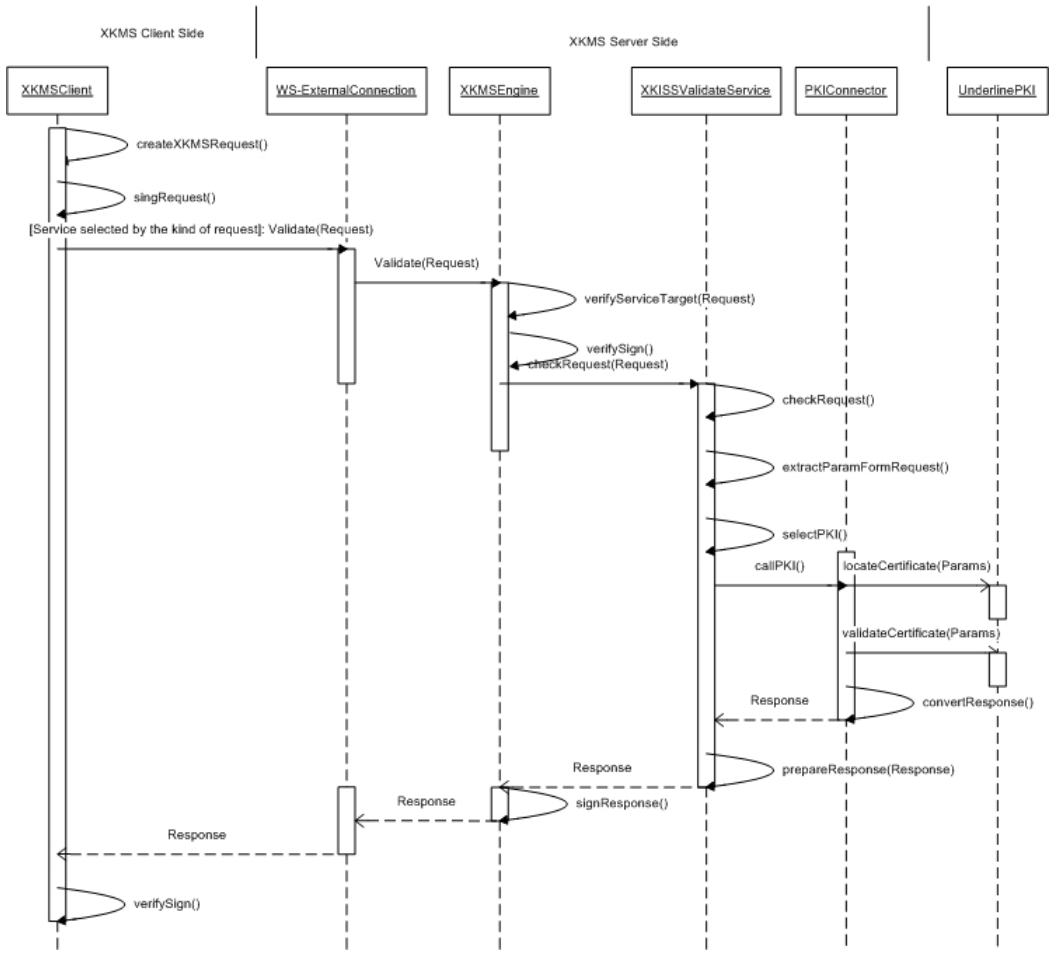


Fig. 3. XKMS synchronous validate request sequence diagram

Figure 3 can help to better understand all previous functionalities. This figure represents a sequence diagram of a request to the validation service. The sequence starts when Alice prepares a validation request for checking Bob's certificate, sign it and send the request to the XKMS server by mean of the XKMS client. The WS-ExternalConnection module catches the request and unwraps it calling to the XKMS engine, as explained in the previous section.

The XKMS engine verifies the service name in order to check if this server is the responsible of processing the request. To do it, a selective mechanism of the PKI provider based on the identity of the user is launched. The request

will be redirected to the appropriate server. Additionally, the XKMS engine verifies the signature on the request using the XML digital signature services.

Following the previous example, when the server receives Bob's email *bob@foo.com* for validation, the *@foo.com* domain is selected in order to decide if this is the appropriate server or not and in the case that it is the correct server, the same domain value is used to determine what PKI connector will be used to do the real validation of the Bob's certificate in the underline PKI.

If the request is a compound request a logical separation will be done, and then each of the resultant parts will be treated as individual requests. This kind of requests provide a performance benefit with respect to equivalent individuals requests as some operations, such a signature services, would need additional time, e.g. in five individual locate requests, five signature operations could be necessaries but in one compound request with five located inside it, only one signature operation is really needed.

At this point, a separation according to the synchronism nature of the request needs to be done. If the final XKMS service has been configured in asynchronous mode (see figure 4), as could be in the *Register* or the *Reissue* service, then a response will be returned to user the and after certain off-line activities with the PKI (i.e. to present credential on the PKI office) the process will continue again. To achieve this kind of processing a temporal repository is needed to store the first part of the request in order to continue the request processing with the new information provided for the PKI provider (i.e. an authentication code which ensures that user has presented his/her ID in the PKI offices).

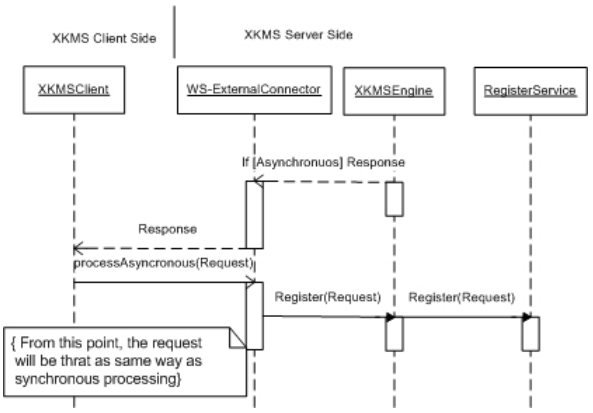


Fig. 4. XKMS asynchronous validate request sequence diagram

Additionally, the *represent* processing mode consists in a two phase challenge protocol used to do selective access to XKMS services according to user privileges. This service has been designed analogously to the asynchronous processing mode and is of particular importante in certain scenarios where, for example, the role of the user can give him access to a different set of services.

After the represent, asynchronous or compound processing in the XKMSEngine module an individual request (or a set of them) is received by the associated service. This service first check the validity of the received request and extract the parameters for the underline PKI.

Then, the underline PKI is called by mean of a *PKIForXKMSConnector* module. In the case of figure 3 the *locateCertificate* and *validateCertificate* PKI services has been called (see section 3.3 for more details). Once the PKI response is received, a new XKMS message will be generated and signed (if it is necessary) by the server. This message will contain the certificate status of Bob's certificate and finally, the response will be deliver to Alice by means of the WS-ExternalConnection module.

To perform all these functionalities, the *XKMSEngine* has been also divided in a set of submodules, thus separating among different tasks. In this sense, nine are the submodules that have been defined and designed, as indicated in the central part of the logical schema presented in Figure 2: the *XKMSService* is in charge of doing the common processing for all the messages; *Locate, Validate, Register, Recover, Reissue and Revoke* manage the different XKMS services; *Asynchronous* and *Represent* services process requests that need certain time to provide an answer; and finally the *Compound* service works with many different requests sent all together in a single XKMS request message.

### *3.3 PKIForXKMS connector*

The third module of the XKMS server is named *PKIForXKMS*, and it is in charge of interacting with the final PKI solution. The kind of interaction with the final PKI depends of the implementation of different connectors that allowing an extensible integration between different PKI infrastructures and the XKMS server.

Giving an example of connector, in our real scenario, a proxy connector has been implemented and it selects the underline PKI based on both e-mail address domain and the suffix of the Distinguished Name (DN field) of the certificate. The real underline PKI connectors that have been implemented are UMU-PKIv6 for *@um.es* domain and Dummy-PKI for other testing domains. In the UMU-PKIv6 connector, the certificate location has been done with LDAPv6 protocol and the certificate validation has been done with both OCSP and SCVP protocols. The former is implemented in the *locateCertificate* method and the later in the *validateCertificate* method of the connector. All the methods exposed in PKI connectors represent the set of actions of the PKI for covering the whole certificate life cycle.

According to our design a connector could provide direct access to a unique

PKI solution or it also could contribute with a proxy access among different PKI solutions in which the PKI provider is selected based on the information provided in each request, e.g. the email domain contained in a locate request.

Thus, a new connector can be defined through an API with a set of methods that a PKI can offer to be used in a XKMS server. In this sense, when a given PKI wants to be used in a XKMS server, the only requirement is to implement these methods. This implementation is usually done by parsing the generic method specified as part of the XKMS service with the particular protocol supporting this functionality in the PKI being added.

In the sequence diagram provided in figure 3, Alice wants to validate a certificate for a given e-mail address; when the validate XKMS service calls the PKI connector, the *locateCertificate* method is invoked obtaining the certificate associated to the provided e-mail address. Then, the *validateCertificate* is later invoked in order to provide a real validation of the certification in the final PKI. For this, the particular method implemented by the PKI (OCSP, SCVP, CRLs, etc.) will be invoked.

In conjunction with the PKI connector, this module also offers data structures to indicate implementers of this module for a given PKI how the information (certificates, CRLs, etc.) should be specified and sent to the XKMS server for it to be able to understand the answer in the *XKMSEngine* module and encapsulate it to be finally provided to the end client using the *WS-ExternalConnection* module.

An example of data structure is provided in figure 5. It is used to indicate the certificate status. This structure, which is currently defined in Java language, is composed by a variable indicating if a certificate is valid or not, and a list of reasons indicating the causes in a specific format. Moreover, when a PKI uses technologies such as OCSP, a conversion between OCSP validation codes, and this data structure need to be done inside the PKI connector by the developer. This will be usually done with a Java class, where the programmer of the PKIForXKMS connector will have to convert from (or to) the XKMS internal structure described in figure 5 to (or from) the OCSP (or similar) protocol message used to make the query to (or to receive the response from) the PKI certificate database.

Additionally, figure 6 shows the interface designed for the PKIForXKMS connector. This interface is used as bypass between an underline PKI and the XKMS server. Most of the methods of this interface matches with common PKI services such as *locate, validate, register, etc.* To make use of them, a Java class is needed. This class will have to implement the sequence of messages that need to be sent and received from the particular PKI service in use. For example, in the case of the *validateCertificate* method, if the PKI is imple-

```
class CertificateStatus{
 /** Store the status of the certificate */
 private String status = ENUM{VALID,INTERMEDIATE, INVALID};

 /** Store the list of the reason associates to this status */
 private List reasons = ENUM\{
   ISSUER-TRUST, /** Valid: Certificate path anchored by trusted root successfully constructed
                  * Invalid: Certificate path could not be constructed to a trusted root*/

   REVOCATION-STATUS, /**Valid: Certificate status validated using CRL or OCSP
                       *Invalid: Certificate status returned revoked or suspended.**/

   VALIDITY-INTERVAL,/**Valid: The certificate chain was valid at the requested time instant
                      *Invalid: The requested time instant was before or after the certificate
                       chain validity interval**/

   SIGNATURE /**Valid: Certificate Signature verified
              *Invalid:Certificate Signature verification failed */
   };
}
```

Fig. 5. PKI Connector data structure

menting an OCSP responder, the developer of an XKMS system will have to develop a Java class supporting the query-response set of messages defined as part of the OCSP protocol.

```
public interface PKIConnector {
    // MAIN PKI FUNCTION

    public Iterator<Certificate> locateCertificate(keyInfo, keyUsageRequested, useKeyWith);

    public CertificateStatus validateCertificate(certificate);

    public CertificateValided registerCertificate(publicKey, keyName, keyUsages, useWithWith, validityInterval);

    public CertificateValided reIssueCertificate(certificate, validityInterval);

    public CertificateStatus revokeCertificate(certificate, validityInterval, reasons);

    public byte[] getAuthenticationCode(String keyName) throws PKIException;

    public byte[] getAuthenticationCode(Certificate certificate) throws PKIException;

    public String getStatusOfRequest(String pkiId) throws PKIException;

    // AUXILIARY FUNCTION

    public X509CRL getCRL(Certificate certificate) throws PKIException;

    public RetrievalMethod getRetrievalMethod(Certificate certificate) throws PKIException;

    public Certificate[] getCertificateChain(Certificate certificate) throws PKIException;

}
```

Fig. 6. PKI Connector interface

### *3.4 Differentiation on Security levels*

When an end user makes use of X-KISS or X-KRSS protocols, he/she may wish to specify the level of security regarding the certification service requested. For example, if the PKI system offers, through the XKMS service, different services to validate the current status of a public key certificate, such as CRLs, OCSP or SCVP, the system could define the following behavior: if the user sends a validation request with a low security level, the system will use the

CRL mechanism to validate the public key certificate, which is less reliable but faster. If the security level is medium, the system will use the OCSP protocol, which is more reliable but not so fast. Finally, if the security level is high, the SCVP protocol will be used, which is highly reliable but slower (as in some cases it implies building and validating certification paths between different PKIs).

Decisions taken according to security levels cab been provided in three different ways; i) the client could provide the security level in which he/she wants to obtain the response; ii) second, the XKMS server could be configured by the administrator in a pre-established security level associated a specific working mode; iii) the PKI infrastructure could have a security manager in order to determine the way of process the request based on PKI administrator policies. Our design covers i) and ii) leaving iii) to the consideration of the PKI particular implementers.

In order to provide a design covering security levels in the client side, two are the main alternatives: to use the *MessageExtension* element that it is present in all XKMS requests in the same way as proposed in [31], or to use the *clientOpaqueData* field also existing in all requests in order to carry the security level to the server. The server could pay attention on the security level or could override it with the security level provided by the XKMS service administrator.

In the server side, a security level determines the working mode inside the server. In this sense, our design is making use of policies for describing the fashion in which these security levels can be selected. In this context different types of policies can be applied such as [9], or [1] for describing the policies. A policy-based decision allows behaviors in the server taking into account a context-aware environment that determines the security level based on the information provided by a set of policy rules. In our design, the policy decision is taking according to several components including i) the request, i.e. the kind of request or the content of the request; ii) the client features such as network, IP address, administrative domain or access method; and iii) PKI features described using a particular structure in the PKIForXKMS connector. These PKI features could cover a wide range of characteristics such as what is the final PKI that will process the request, the fashion in which this PKI is configured, time intervals or PKI server load. With the inclusion of policies in the design, the server is able to select the actual security level based on the request and the current context.

## 4 Implementation and Results

### *4.1 Implementation details*

The solution proposed in this work is a XKMS implementation that offers both client and server side called Open XKMS[29]. The server side has been developed to be used with any kind and number of PKIs. To show that, some underline PKIs has been connected in order to provide different examples of usage. One of them is a dummy PKI implementation, which offers all basic services, but all of them are built-in inside itself.

In addition, a more complex connector has been deployed for the UMU-PKIv6 software [28], as a certification service provider. UMU-PKIv6 provides the use of LDAPv6 to store public key certificates and CRLs, smart cards use to store key material, CMC and SCVP protocols, policy definition to establish certification restrictions, Bridge CA, peer-to-peer and hierarchical cross certification, support of SCEP, DNSSEC, time stamping and OCSP protocols, and communications based on both IPv4 and IPv6.

The solution proposed also provides several entry point methods to the WS-ExternalConnection module in order to cover a wide spread of web services technologies. It offers access via web service clients compatibles with Apache SOAP [7], AXIS [8] or JAX-WS [18] technologies. The implemented architecture can be seen in figure 7.

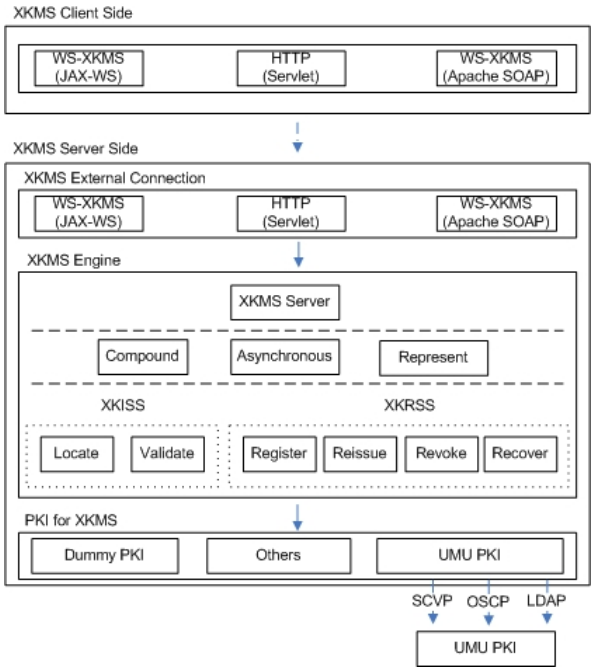


Fig. 7. OpenXKMS server architecture

OpenXKMS fulfills XKMS 2.0 specification as all the tests provided by W3C [6] for checking the behavior of an XKMS server implementation has been passed. This testbed offers 34 tests which all XKMS implementation should pass in order to ensure compatibility. Fulfilling XKMS 2.0 specification our implementation ensures the support for both asynchronous and synchronous mode, compound requests and all X-KISS and X-KRSS services.

It is worth mentioning that the *represent* mode has been implemented using nonce challenges in which, first the client send a message and the server answers with the challenge. Then, if the user is an authorized user, he/she resolves the challenge and send the request again with the solved challenge inside. Finally the server starts processing the request.

Given the overhead included by the XML Digital Signature mechanism, XKMS services has been implemented offering the possibility to enable or disable both server and client side digital signature. At same time the implementation offers the possibility to change the digital signature and encryption providers to include in future implementations other provides such as XaDeS[13].

### *4.2 XKMS Performance*

Once we have described the main issues of the server implementation, we present a complete testbed scenario. In it we continue using the previous example where Alice is browsing or checking her email and wants to validate the X.509 certificate of Bob. This scenario follows the sequence diagram seen in figure 3, and starts when she sends a validate request to the pre-configured XKMS server. After the web service routing layer has been processed, and the request has been received and validated by the *XKMSService*, it invokes the suitable method of the *Validate* component, as described before.

*Validate* determines the requested data and makes use of UMU-PKIv6 methods to retrieve the Bob certificate by means, for example, of LDAP; afterward it invokes the validation method of the UMU-PKIv6 connector that derives in a OCSP request. When the response arrives to *Validate*, and later to the *XKMSService*, it builds a WS response message to Alice with the final answer.

This scenario could be divided into two different parts; the first one in which the underline PKI needs to localize the Bob certificate through LDAP, and then it is validated through the OCSP protocol. In the second one, Alice provides the certificate, so the localization process is skipped.

Additionally to these two scenarios, a location scenario has been also developed. In this case, only the localization of the certificate is needed, but not the validation process.

We have tested these three scenarios running the testbed in a Intel Centrino Core 2 DUO T7500, 2.2 Ghz with 4 Gb RAM running a Windows XP operating system for the client. The server is running an Apache Tomcat 6 with the XKMS server side in a computer with similar features. Note that all software involved in the testbed has been developed in Java and therefore they are platform-independent and could be running in any OS system. Performance data can be found in Figure 8.

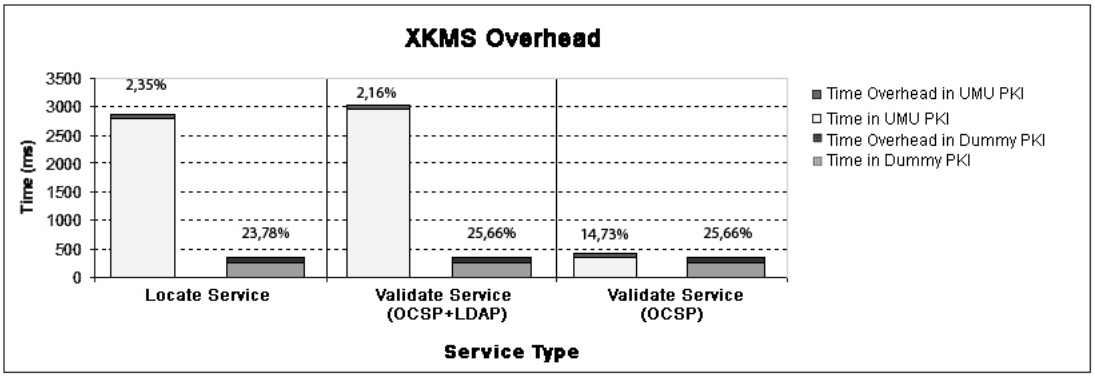


Fig. 8. Overhead by the use of Open XKMS server

In figure 8, reader can see the overhead imposed by the usage of XKMS server grouped by the three previous described scenarios. In the figure appears stacked bars (light color bars represents time access to the UMU-PKIv6 and dark color bars represents time access to the Dummy PKI). The overhead time can be seen on top of each bar and the percentage of this overhead with respect total time. The results has been obtained running 100 instances of each pki/service analyzed and calculating the average of the measured times.

These results lead in an overhead time of 25% in case of Dummy PKI implementation and 6.4% in case of UMU-PKIv6. The former result might be consideres as excessive but it is due to the minimal access time that consumes the PKI when accessing the dummy PKI, without any kind of delay due to connections and protocols. The latest result represents a more realistic scenario and it is more representative, as the UMU-PKIv6 is running in a different machine in the same network. In this last case, results can be considered as acceptable as the overhead is minimum. The standard deviation in the case of Dummy PKI is 13% and in the case of UMU-PKIv6 is 3,8%.

Some probes has been done in relation with scalability of the OpenXKMS server obtaining the following results (see Figure 9).

Figure 9 shows the average time by user in processing a certificate location request given a simultaneous number of users requesting the same service. In the same figure a lineal trend line can also be seen.

Regarding figure 9, as simultaneous user accesses increase, the average re-

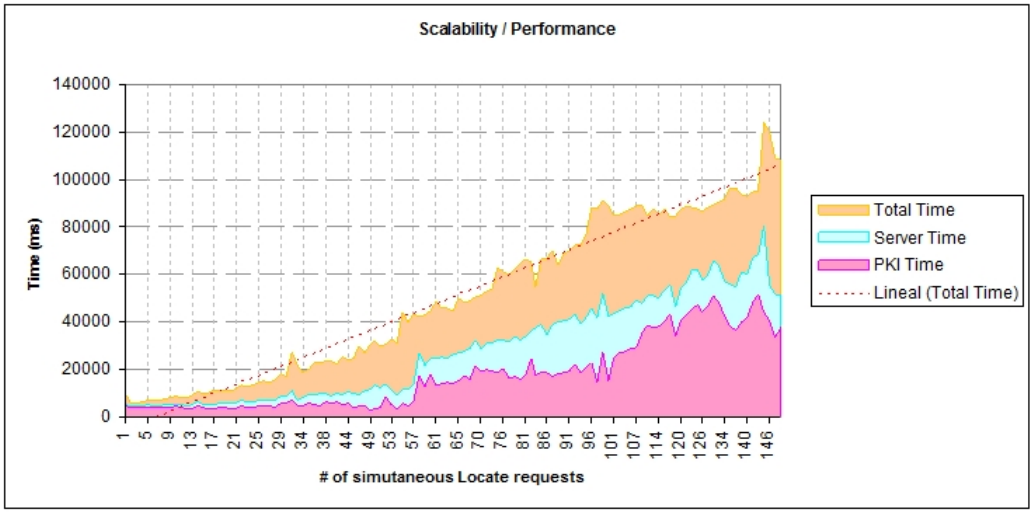


Fig. 9. Open XKMS server scalability result

sponse time is also increasing. This is normal behavior in these scenarios. The relationship between the average of time increased by user with the number of simultaneous user accesses provides a good metric about server scalability. In our implementation, this relationship follows an almost linear increase that can be observed in the trend line presented in figure 9. It fact, it confers a good scalability feature to our solution. Giving an example, it could be observed that the average response time for 30 simultaneous request is 18 sec and for the double of simultaneous request is 44 sec.

## 5 Related Work

Since the publication of XKMS version 1.0, a wide number of security related companies have defined their XKMS reference implementation or prototype. Examples of these companies are Verisign, Microsoft or RSA Security. Verisign and Microsoft are working in the definition of a XML Trust Services [30] solution, using XKMS to define an easy way to access to PKI services. Microsoft has also deployed a XKMS solution based on ASP.NET [14], and RSA Security has included XKMS support in the BSAFE and Keon solutions [26].

These implementations are usually based on different programming languages, such as Java, ASP.NET, C++, etc. But there are two main important issues regarding these XKMS solutions. Firstly, they are only based on the X.509 standard to offer certification services. Secondly, they are integrated with the security services products offered by these companies.

Other implementations, have tried to give a more open approach of XKMS, defining an open architecture which offers advanced certification services. In

[25] a WS XKMS architecture is presented, and it is integrated with an advanced version of the OCSP protocol for the definition of a light on-line certificate validation service. [19] proposes a similar and very basic modular architecture of a U-KMS (Unified-KMS), able to connect different certification services, but making use of X.509 services. In this case, how to manage different PKI technologies through a single XKMS server is not described. Finally, a proposal for the integration of SPKI services with XKMS is provided in [31].

In contrast to the current solutions for XKMS, the work proposed in this paper presents an open, full and modular architecture, which, through a WS interface, allows the integration of different PKI specifications (X.509, PGP, SAML, Kerberos, etc.), and which also allows the integration of different certification services offered by each specification. For example, using a X.509 PKI infrastructure, this solution would be able to support, in an easy way and adding the required modules, a validation service based on OCSP or SCVP, or a certification service based on CMC or PKCS#10 protocols.

Finally, this work also presents specific data about the impact in the performance of security infrastructures by introducing a new certification proxy, the XKMS server. Some interesting data can be found in [31] related to digital signatures and encryption time costs, but not involving the whole process.

## 6 Conclusions and future work

This paper has been presenting the main issues behind the design of an XKMS server characterised for being modular, extensible and reusable in different scenarios. It also presents details of an implementation of an XKMS client and server (named Open XKMS) and some specific performance data of this implementation. This standard is considered as a key component for providing WS-based secure and homogeneous access to trust management infrastructures based on different standards such as X.509 or PGP.

As future work, this research will be evolving to cover the effective management of the shared secrets needed to start authenticating valid XKMS clients. Also P2P and Bridge CA multi-domain scenarios will be tested to identify open issues that either the XKMS specification or the Open XKMS implementation are not covering.

## References

[1] CIM-Common Information Model, DMTF Standard, DMTF,

http://www.dmtf.org/standards/cim/ (2008).

[2] E-Commerce PKI, http://www.ecommercepki.com (2008).

[3] European Bridge-CA EB-CA, http://www.bridge-ca.org/html/eb-ca.html (2008).

[4] The InCommon Federation web site, http://www.incommonfederation.org/ (2008).

[5] The SWITCH Federation web site, http://www.switch.ch/aai (2008).

[6] G. Alvaro, XKMS Assertions and Test Collection, W3C Recommendation, W3C, http://www.w3.org/2001/XKMS/Drafts/test-suite/CR-XKMS-test-suite.html (Feb 2005).

[7] Apache Foundation, Apache SOAP Web Service Implementation, http://ws.apache.org/axis/ (2006).

[8] Apache Foundation, Apache AXIS2 Web Service Implementation, http://ws.apache.org/axis2/ (2008).

[9] J. P. Bigus, D. A. Schlosnagle, J. R. Pilgrim, W. N. Mills-III, Y. Diao, ABLE: A toolkit for building multiagent autonomic systems, Artificial Intelligent 41 (3) (2002) 350.

[10] S. Cantor, J. Kemp, Liberty Protocols and Schema Specification Version 1.2, liberty Alliance Project (Jan 2003).

[11] S. Cantor, J. Kemp, R. Phipott, E. Maler, Assertions and Protocols for the OASIS Security Assertion Language (SAML) v2.0, OASIS Specification, OASIS, http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf (Mar 2005).

[12] M. Cooper, Y. Dzambasow, P. Hesse, S. Joseph, R. Nicholas, Internet X.509 Public Key Infrastructure: Certification Path Building, RFC 4158, IETF, http://tools.ietf.org/html/rfc4158 (Sep 2005).

[13] J. C. Cruellas, G. Karlinger, D. Pinkas, J. Ross, XML Advanced Electronic Signatures (XAdES), W3C Recommendation, W3C, http://www.w3.org/TR/XAdES/ (Feb 2003).

[14] B. Dillaway, Implementing XML Key Management Services Using ASP.NET, ASP.NET Technical Articles.

[15] D. Eastlake, D. Solo, J. Reagle, XML-Signature Syntax and Processing, W3C Recommendation, W3C, http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/ (Feb 2002).

[16] M. Halappanavar, R. Mukkamala, ECPV: Efficient Certificate Path Validation in Public-key Infrastructure, in: Seventeenth Annual Working Conference on Data and Application Security (DBSec), IFIP TC-11 WG 11.3, 2003.

[17] P. Hesse, D. Lemire, Managing Interoperability in Non-Hierarchical Public Key Infrastructures, in: Network and Distributed System Security Symposium (NDS), Internet Society, 2002.

[18] Java Developer Community, Java API for XML Web Services (JAX-WS), https://jax-ws.dev.java.net/ (2008).

[19] J. Kim, K. Moon, Design of Unified Key Management Model using XKMS, in: 7th International Conference on Advanced Communication Technology, (ICACT05), vol. 1, IEEE, 2005.

[20] C. Lambrinoudakis, S. Gritzalis, F. Dridi, G. Pernul, Security requirements for e-government services: a methodological approach for developing a common PKI-based security policy, Computer Communications Journal 26 (16) (2003) 1873–1883.

[21] R. Martinez-Pelaez, C. Satizabal, F. Rico-Novella, J. Forne, Efficient Certificate Path Validation and Its Application in Mobile Payment Protocols, in: Proceedings of the Third International Conference on Availability, Reliability and Security (ARES) 2008, IEEE, 2008.

[22] Microsoft Developer Network, Web Services Security Specifications Index Page, http://msdn2.microsoft.com/en-us/library/ms951273.aspx (Oct 2007).

[23] T. Moses, OASIS eXtensible Access Control Markup Language (XACML), OASIS Specification, OASIS, http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf (Feb 2005).

[24] S. H. Mysore, P. Hallam-Baker, XML Key Management Specification (XKMS 2.0), W3C Recommendation, http://www.w3.org/TR/xkms2/ (Jun 2005).

[25] N. Park, K. Moon, J. Jang, S. Sohn, Development of XKMS-Based Service Component for Using PKI in XML Web Services Environment, in: Proceedings of the International Conference on Computational Science and its Applications (ICCSA04), vol. 3043, Springer, 2004.

[26] RSA Inc., RSA Security, http://www.rsasecurity.com (Mar 2005).

[27] T. Scavo, S. Cantor, Shibboleth Architecture. Technical Overview, working Draft 02 (Jun 2005).

[28] University of Murcia, UMU-PKIv6 (University of Murcia Public Key Infrastructure with IPv6 support), http://pki.dif.um.es/ (2008).

[29] University of Murcia and Berne University of Applied Sciences, Open XKMS, http://xkms.sourceforge.net/ (2008).

[30] Verisign Inc., XML Trust Services - XKMS, http://www.verisign.com/developer/xml/xkms.html (Mar 2005).

[31] M. Wangham, E. R. de Mello, J. da Silva Fraga, D. da Silva Bger, A Model to support SPKI Federations management through XKMS, in: IEEE International Conference on Web Services (ICWS07), vol. 9, IEEE, 2007.