An Advanced Certificate Validation Service and Architecture based on XKMS[1]

Antonio Ruiz Martínez, Daniel Sánchez Martínez, C. Inmaculada Marín López, Manuel Gil Pérez and Antonio F. Gómez Skarmeta

Department of Information and Communications Engineering
University of Murcia, 30071, Murcia, Spain
Email: {arm, danielsm, inmaml, mgilperez, skarmeta}@um.es

SUMMARY

The apparition of some laws that make the electronic signature (e-signature) legally equivalent to handwritten signature (under some circumstances) has favoured its use in different fields such as e-commerce, e-government, etc. In these fields, some signed documents have to be stored and remain valid over long periods of time. For this kind of e-signatures some formats such as CAdES and XAdES have appeared. These formats specify the information to include with the e-signature. Basically, this information comprises signer's certificates, a set of certificates up to a trust anchor, certificate validation responses, etc. That is, the information needed to determine if an electronic signature is valid. These evidences can be gathered by using different PKI-compliant protocols. However, the support of the different protocols is complex for clients. XKMS appeared with the aim of simplifying the certificate management, but XKMS only supports a simple validation mechanism that does not provide the long term information needed for the CAdES/XAdES signature. As a solution to this problem, we have extended XKMS in order to support the obtaining of long term evidences needed for CAdES/XAdES signatures. With this extension we have also defined the different components that are needed to support this kind of service. Based on the definition provided, the service has been implemented and it has been incorporated to an e-government infrastructure based on service-oriented architectures, which is able to create and verify this kind of signatures.

KEY WORDS: certificate validation, advanced electronic signatures, service oriented architecture, XKMS, validation evidences

1. INTRODUCTION

Electronic signature (e-signature) provides some interesting features such as integrity, authentication and non-repudiation. Thanks to these features, the use of e-signature can guarantee non-repudiation in electronic commerce (e-commerce), business (e-business) and government (e-government) transactions. Moreover, it can be considered an equivalent to the handwritten signature. In fact, some laws such as the directive

[1] This article is an extended and revised version of "ACVS: an Advanced Certificate Validation Service in Service-Oriented Architectures". Published in Proceedings of the Third International Conference on Internet and Web Applications and Services (ICIW'08). pp. 297-302, Athens (Greece), 2008. DOI: 10.1109/ICIW.2008.84

1999/93/EC of the European Parliament and Council [1] and the UNCITRAL Model Law [2] have appeared to guarantee that if the e-signature is developed in some circumstances, it can be considered legally equivalent to the handwritten signature. This acknowledgment has favoured the increase in the number of different electronic business/commerce/government transactions. Thus, paper documents have started to be replaced by electronic documents.

Some of the signed documents that are generated in e-commerce, e-business and e-government environments have to be stored over long periods of time due to legal requirements [3]. Some examples of these documents are e-invoices, title-deeds, etc. In order to be able to validate a signed document over these long periods we need to include some of the validation evidences that were used to validate the signature when it was stored. These validation evidences can be signers' certificates, a set of certificates from signer's certificate up to a trust anchor, Online Certificate Status Protocol (OCSP) [4] responses, Certificate Revocation Lists (CRLs) and/or Authority Revocation Lists (ARLs) [5], etc. In fact, for this purpose, some signature formats such as CMS Advanced Electronic Signature (CAdES) [6] and XMLDSig Advanced Electronic Signature (XAdES) [7] have been proposed. These formats extend Public Key Cryptographic Standard 7 (PKCS#7) [8]/Cryptographic Message Syntax (CMS) [9] and XML Signature (XMLDSig) [10] standards, respectively, in order to include these evidences within the e-signature. Their inclusion is performed in different moments in time. Some evidences can be included at the same time we are creating the signature according to CMS/XMLDSig formats. Others can be included just before creating the e-signature and validating it. Finally, some of them can be included some time after its validation (if the evidences are not available, e.g., CRLs) or periodically in order to avoid cryptographic and information weaknesses.

In the validation of an advanced electronic signature we have just mentioned, we need to perform two main processes: path construction or path discovery up to a trust anchor and path validation. This process as a whole is referred to as certification path processing [11],[12].

In order to perform the path construction, protocols such as Lightweight Directory Access Protocol (LDAP) [13] or Server-based Certificate Validation Protocol (SCVP) [14] can be helpful. In order to carry out the path validation, there exist several mechanisms that we can use, such as CRLs, OCSP responses, SCVP responses, etc. A deeper analysis of the different mechanisms can be found in [15]. To perform this process we have to support different Public Key Infrastructure (PKI) protocols with complex syntax and semantics. Furthermore, this process can suppose an important overhead for some thin clients such as Personal Digital Assistants (PDAs) or mobile devices, which have limited resources. As a solution to this problem, the XML Key Management Specification (XKMS) [16] was proposed.

The XKMS specification is based on Service-Oriented Architectures (SOA) [17],[18] and defines several protocols for registering, distributing and obtaining information about public keys. Its definition as an SOA service is due to the fact that it helps to create interoperable services. This also facilitates its integration with many services that are offered in electronic commerce and government platforms, which are migrating their services to SOA-based services.

XKMS, specifically for the purpose of validating a certificate, defines a *Validate* service. The main drawback of this service is that it only returns a simple response without any additional information about the validation protocol or the evidences used.

Therefore, the information provided by this service is not enough to recover the information needed to create long term signatures according to CAdES/XAdES formats. The only element we can recover is the signer's certificate, which, in turn, can also be recovered with the *Locate* service. However, we need to recover some other information used in the validation process such as the set of certificates up to a trust anchor, CRLs, OCSP responses, etc.

The aim of this paper is to present a solution to the problem of creating long term signatures from XKMS service in an SOA infrastructure. For this reason, we propose a system architecture with the different elements that should participate in order to cope with long-term signature (creation and validation). This solution can be useful as the basis for other standards related to Web services such as SAML or XACML, as well as for some scenarios such as e-government platforms, e-invoices, registered electronic mail, mobile platforms, grid infrastructures, identity federations, etc.

The main contribution to this architecture is the extension of XKMS *Validate* and *Locate* services to return the evidences needed. With these extensions we also define the different modules that should be included in the service to support them. This design is modular and supports the use of different PKI solutions as well as the extension with new protocols and new PKIs. Our design also takes into account some components from other proposals such as Secure client-server Architecture for the Validation of X.509 Certificates (SAVaCert) [19] and Efficient Certificate Path Validation (ECPV) [20], as well as extending them in order to provide the new functionality.

Additionally, we have also defined an authentication mode based on Payword [21] to support fast authentication in XKMS when there is a long term relationship between the client and the server.

With our proposal, we facilitate both the obtaining of these evidences and the migration of current XKMS services to this new specification. This extension and architecture have been implemented and are currently part of the e-government platform of the University of Murcia. In this paper, we present both the platform and the scenario where this service is been used.

This paper is structured as follows: in Section 2 we introduce background information related to the creation and validation of advanced electronic signatures. Thus, we describe some use scenarios of advanced electronic signatures, the e-signature validation process, the information to be included, the different advanced electronic formats and the different proposals used to create these signatures. In Section 3 we present the different components that are part of the advanced certificate validation architectures. One of the fundamental components in this architecture is the validation authority that offers the advanced validation service. Its specification is described in detail in Section 4. Then, in Section 5 we present an SOA-based advance electronic signature that is being used in an e-government infrastructure. Section 6 analyses and discusses some related work. Finally, in Section 7 we present our main conclusions and possible future work.

2. CREATION AND VALIDATION OF ADVANCED ELECTRONIC SIGNATURES

This section introduces different issues related to the creation and verification of e-signatures that remain valid over long periods of time as well as some scenarios where these signatures are useful and/or necessary. Our aim is to establish the context and

explain the different problems that arise when we want to deal with advanced electronic signatures. Later, we provide a solution to the problems commented.

Nowadays, in e-government, e-business and e-commerce systems, the use of the e-signature is fundamental to support different kinds of transactions. In general, in these transactions some documents have to be signed. The e-signature of such documents is expressed according to formats such as PKCS#7 [8]/CMS [9] or XMLDSig [10]. Some of these signed documents have to remain valid over long periods of time. For this purpose, Advanced Electronic Signature (AdES) formats have been proposed.

Basically, AdES formats, that is, CAdES [6] and XAdES [7], are based on the incorporation of some additional information in the PKCS#7/CMS and XMLDSig formats. These AdES formats can be created by both the signer and the verifier. The elements included are commented in more detail in Section 2.2. We can anticipate that, basically, these elements are the certificates used to validate the e-signature as well as the different evidences (CRLs, OCSP responses, etc.) to prove that the certificates were valid at the moment the signature was created. Furthermore, some timestamps are included to prove the existence of these evidences at a moment in time.

In order to obtain these validation evidences, we could make use of XKMS when validating the signature and the certificate before storing the signed document. Specifically, XKMS can be used to locate some certificates and validate them. However, at this moment, the information provided by XKMS, as we analyze in Section 2.3, is very simple but not enough to create AdES formats.

2.1. USE SCENARIOS OF ADVANCED ELECTRONIC SIGNATURES

These advanced formats are useful and/or needed for some kind of scenarios. An example of such scenario is the exchange of electronic invoices (e-invoices), a fundamental process in e-business/government transactions. In these e-invoices the authenticity and integrity are preserved thanks to e-signature. Without the verification of the e-signature at issuance time, the validity of the e-invoice cannot be guaranteed. However, in many cases, this is not enough. Due to legal requirements, the receiver of the e-invoice has to store the electronic invoice for some time period. During this period the Directive [22] establishes that the authenticity of the origin, the integrity and the readability must be preserved.

In order to support long term signatures, the signature stored with the e-invoice must be signed according to the CAdES/XAdES-X-L format (see more details in Section 2.2). Some governments that are working on this kind of signature are the German and Spanish Governments.

Another scenario of application is the e-mail. Nowadays, the e-mail is one of the most important tools in e-business and e-government. However, e-mail systems do not provide enough security services to be trusted. As a response to this need, some services named Registered Electronic Mail (REM) services have been proposed [23]. In this kind of systems, the goal is to provide origin authentication and proof of delivery. For this purpose, in REM the Advanced Electronic Signature is used and defined how to apply it to satisfy the goals we have just mentioned.

As a final scenario, it is worth mentioning that in some countries, such as Spain, the exchange of e-dossiers between different public agencies has been regulated. These dossiers are signed according to the XAdES signature in order to guarantee the origin, integrity and authentication of the information exchanged.

Due to these needs and the different kinds of documents where AdES formats can be used, some profiles have been defined [24],[25]. These profiles establish the specific AdES format to use in a particular type of document as well as the different elements to be incorporated.

In the rest of this section we briefly explain the different AdES formats, which information is needed to create them, how we can obtain this information, etc. In particular, we comment that XKMS can be an interesting candidate to simplify the process of recovering this information. However, as we also explain later, this protocol, with its current specification, cannot be used for this purpose and therefore we have decided to extend it.

2.2. ELECTRONIC SIGNATURE AND AdES FORMATS

Currently the standard formats that are available to e-signatures are PKCS#7/CMS and XMLDSig. The former is based on the ASN.1 data model, whereas the latter is based on the XML data model. Both formats basically allow expressing the same information: data to be signed (optional), algorithms used, certificate (optional) or a reference to the certificate; and the signature value itself.

In the e-signature verification process we do not only need to verify whether the document is correctly signed, but also whether the certificate used was valid at the moment the signature was generated or at the moment the verification is being performed. Specifically, the verification of an e-signature involves:

- The verification of the e-signature value.
- The validation of the additional data needed to validate the e-signature (set of certificates up to a trust anchor, certificate status of each certificate in the complete certification path). These certificates must be valid at the moment the signature was performed. For this purpose, we need to recover the different certificates up to a trust anchor and perform the validation of the status of each certificate in the path. In Section 2.3 this process is explained in more detail. Neither PKCS#7/CMS nor XMLDSig define how to incorporate this information.
- The verification of the existence of the signature at a specific moment in time. For this purpose, timestamps are used.

When the e-signature verification is performed at a time close to the moment when the e-signature was produced, the information previously mentioned might be recovered easily. However, if this e-signature has to be stored and remain valid over long periods, we can find some problems when verifying this signature much later: the certificates set could be unavailable, some keys could be compromised (signer’s key, CA’s key, etc.), the algorithms used to generate signatures and/or certificates could become weak, etc.

To overcome these possible problems, the use of the advanced electronic signature (AdES) formats is proposed [3]. Then, we describe the different formats defined in AdES formats and the different information incorporated in each of them.

2.2.1. AdES

AdES formats [6],[7] were proposed with the aim of defining several electronic signature formats for various types of transactions where long-term validity is

important. Thus, AdES defines electronic signature formats that support the validity of electronic signatures over long periods of time.

Next, we briefly mention the different formats and the information that is included in each format. We go from the simplest to the most complex. Each format includes the information that was defined by the previous one. For this reason, in each format we only describe the new information that is included. Basically, each format involves inserting a set of signed and/or unsigned properties over basic signature formats (CMS or XMLDSig). These formats are:

- *Basic Electronic Signature (AdES-BES).* It includes the signing certificate (or a digest of it) as a signed property. It can also include other signed and/or unsigned properties such as signing time, data object format, signer role, etc.
- *Explicit Policy-based Electronic Signature (AdES-EPES).* It is built from CMS/XMLDSig or from AdES-BES formats. It includes a signed property to indicate a reference to the signature policy used.
- *Electronic Signature with Time (AdES-T).* It adds a timestamp of the signature as an unsigned property.
- *Electronic Signature with Complete Validation References (AdES-C).* It adds to the AdES-T format the references (certificate hash, issuer and serial) to all the certificates used to validate the signature. It also includes the references to all revocation data used in the signer and the CA certificates validation.
- *Extended Signatures with Time Indication Forms (AdES-X).* It adds one or more timestamps to protect the information inserted in the previous format.
- *Extended Long Signatures with Time (AdES-X-L).* It adds the set of certificates (not the references) to all the certificates used to validate the signature, as well as all the revocation data needed to validate the signer and CA certificates.
- *Archival Electronic Signatures (AdES-A).* It adds a timestamp to protect the information inserted in the previous format. At a certain time, a new timestamp can be inserted to protect the signature from algorithms and cryptographic material weaknesses.

2.3. CERTIFICATION PATH PROCESSING

In this section we describe the process to be carried out to validate a certificate. We also mention the main existing mechanisms that can be used for this purpose and that could be used to recover the evidences needed for the AdES formats.

With the purpose of validating a certificate, the following steps must be executed:

1. Building one or more candidate certification paths between the certificate to validate and one established trust anchor. This is called *path construction* or *path discovery*. This task can be carried out by a client performing a recursive search through multiple directory protocols such as Directory Access Protocol (DAP), LDAP, HyperText Transfer Protocol (HTTP) and File Transfer Protocol (FTP). The client can also delegate this task to a server. The set of requirements that has to be taken into account to define a protocol with delegated path discovery processing is specified in Delegated Path Discovery (DPD) [26]. Some protocols which satisfy those requirements are SCVP and ECPV. This latter mechanism is

analysed in more detail in the following section because we have considered some interesting components for our validation authority architecture.

2. Checking that each certificate in the certification path is valid. This implies to check that its structure is correct and satisfies certain constraints (e.g. path length constraints, name constraints, etc.), it is within its established validity period, it has not been revoked and the issuer's signature over the certificate is valid. This step is called *path validation*. This process can be performed directly by a client with the help of several mechanisms such as CRLs (and ARLs) and OCSP responses, or even through an XKMS system. This task can be also delegated. The requirements that should satisfy a delegated server are described in Delegated Path Validation (DPV) [26]. Both SCVP and XKMS satisfy them. A deeper analysis of the validation mechanisms can be found in [15].

The process consisting of these two steps is referred to as *certification path processing* [27],[28]. In order to facilitate this complex process, XKMS has appeared. XKMS is especially useful in SOA architectures because it is specified as a Web service. XKMS is also part of the WS-* security related to protocols and can constitute the basis of other proposals such as WS-Authorization, WS-Federation, WS-Policy, WS-Trust, etc. XKMS is analysed in more depth in Section 2.3.2 because it is an essential part of our proposal.

2.3.1. ECPV

ECPV [20] is an Efficient Certificate Path Validation scheme, designed only for public key infrastructures based on X.509 certificates, in which clients can delegate the complex process of building and validating users' certificates. This scheme is based on one or more Certificate Path Validation Authorities (CPVAs). These authorities build and analyze candidate certification paths according to the trust anchors in which a relying party trusts and the policies/constraints established by it. These certification paths are called Certificate Path Validation Trees (CPVTs) which can be used later by the relying parties to carry out a local validation in an offline manner.

The ECPV architecture is composed by the following components:

- *Subject module*. Authorized clients that can request to the relying parties the validation of their user's certificates.
- *Relying party module*. It receives the requests from the clients and forwards, on their behalf, the validation requests to the corresponding CPVA. This last component provides all the needed information for the building and validation process (i.e. certificate to be validated, trust anchors and validation policies).
- *CPVA module*. The Certificate Path Validation Authority performs the process of path discovery (building the CPVTs) and validation. This module is divided into the *harvester* and *analyzer* submodules, which are in charge of gathering required information from the different PKIs (e.g. CA certificates, CRLs, OCSP responses) and building CPVTs based on relying parties' requirements, respectively.
- *CA module*. This module represents a specific underlying local PKI with its own internal components; that is, the CA, its RA, the public repository and OCSP responder.

CPVAs can operate either as autonomous entities or in a federated mode. When relying parties handle a few domains (i.e., a reduced number of CAs), autonomous entities work very well. But nowadays thousands of CAs might be managed by a single relying party, and therefore a federated architecture is necessary to harvest data about all CAs. In this case, CPVAs could be organized hierarchically to share this information among them, and thus to build the different CPVTs more quickly.

ECPV is a good distributed scheme to validate X.509-based certification paths, although it presents some important disadvantages that should be taken into account. Firstly, ECPV can only manage X.509 digital certificates. This entails an important negative consideration since other formats have to be supported, such as SPKI and attributes certificates. Secondly, this scheme does not consider that future non-repudiation checking operations could be carried out. Thus, although validation evidences might be stored, they are not used later. Finally, ECPV does not provide any mechanism by means of which clients can select the validation protocol needed for validating certificates requested by them.

2.3.2. XKMS

XML Key Management Specification (XKMS) [16] is a W3C recommendation which is not tied to specific protocols defined by the underlying PKIs, thus making clients totally independent of the complex process that those protocols require. XKMS is a Web Service which involves exchanging trust information between clients and the underlying PKIs. Clients are able to use, in a simple way, different PKI solutions (which might be based on different specifications such as PKIX, SPKI or PGP) with many different protocols which these PKIs could be using.

This means that clients can outsource the processing of key management to a dedicated server in order to reduce substantially the complexity, high processing and memory requirements of the underlying protocols. Thereby, XKMS is a very attractive solution when small devices are being used by clients. Furthermore, XKMS uses XML as exchange format to express the services for certificate management, thus being able to use the XML-Signature and XML-Encryption protocols to provide a secure information exchange.

With respect to the scope of this work, one of the main parts defined in XKMS is the XML Key Information Service Specification (X-KISS), which defines two services. The first one, called X-KISS *Locate*, resolves *<ds:KeyInfo>* elements for gathering the corresponding public key certificates in order to validate or decrypt secure messages.

The second service, called X-KISS *Validate*, performs the same function as X-KISS *Locate* and, in addition, clients may obtain an assertion specifying the status of a given public key, which will depend on a single set of validation criteria. However, X-KISS *Validate* entails some disadvantages that should be treated. On the one hand, it is not able to return any certificate evidence needed to create advanced electronic signatures such as a set of certificates (or references to them) up to a trust anchor, CRLs, OCSP responses, etc. On the other hand, clients are not able to specify which validation protocol should be used by the service to validate the certificates requested by them. Thus, the validation results could vary depending on the protocol chosen by the corresponding PKI. For example, let us suppose that X-KISS *Validate* uses a CRL mechanism to check the revocation status of the certificates. When a certificate becomes revoked it is possible that this certificate could not be considered as revoked until the

new CRL is issued by its corresponding CA. By using other validation mechanisms such as OCSP, new revoked certificates are instantly deemed not valid.

As explained above, XKMS can offer the service which is used to access an SOA-based validation authority. In order to perform the different tasks related to a validation authority, a set of modules is needed. In the following section, we analyse a proposal named SAVaCert that covers this open issue. This proposal is analysed because we have considered the different components in it in order to define the architecture of our validation authority.

2.3.2. SAVaCert

SAVaCert [19] is a Secure client-server Architecture for the Validation of X.509 Certificates. Clients can delegate only path discovery for doing an offline validation locally, or both path discovery and validation together. This architecture is comprised of several modules, which are defined in a suitable level of abstraction, in order to allow developers to choose the most appropriate protocols and validation mechanisms (both on the client and server side).

On clients' side, there exists the possibility of configuring some parameters to control and tune the behaviour of the validation process. Between those parameters we can emphasize:

- Indication of what information used during path processing must be returned.
- Set of acceptable certificate policies for the CAs in a certification path.
- Other specific parameters for the selected protocol between client and server.

Regarding the Certificate Validation Server (CVS) defined by SAVaCert, the modules involved in the validation process are the following:

- *Validation module*. This is the module in charge of building and validating the certification paths for a given certificate. Since it is based neither on a particular cryptographic library nor certificate management library, developers can freely choose the underlying protocols and mechanisms. This module can be in turn divided into various submodules:
  - *Validation Protocol module*. It manages the messages exchanged between clients and the server. Thus, this is dependent of the underlying protocol used between them, like SCVP.
  - *Path Validation module*. It is responsible for validating the status of each certificate in the certification path.
  - *Path Construction module*. It tries to build one or more candidate certification paths for the requested certificate according to the defined certificate policy constraints.
  - *Certificate Status module*. It determines the status of a certificate depending on the revocation mechanism used (e.g. CRL, OCSP).
  - *Policy Processing module*. It manages both the certificate and validation policies.
  - *Time module*. It supplies an indication of time, such as timestamps.

- *Storage module*. It is appointed to store and/or retrieve certificates, revocation data and policies.

This architecture is not complete enough since it does not consider several aspects which are essential for an advanced certificate validation service; namely, among others: an element that carries out authorisation operations to control accesses to the service, support for asynchronous operations, management of profiles (not only service profiles but also user ones) and management of service policies.

3. ADVANCED CERTIFICATE VALIDATION ARCHITECTURE

In this paper we propose the extension of XKMS in order to provide an advanced validation service that can be used to recover the evidences needed for the AdES formats. This extension, which we have called Advanced Certificate Validation Service (ACVS), defines the service that could provide a validation authority.

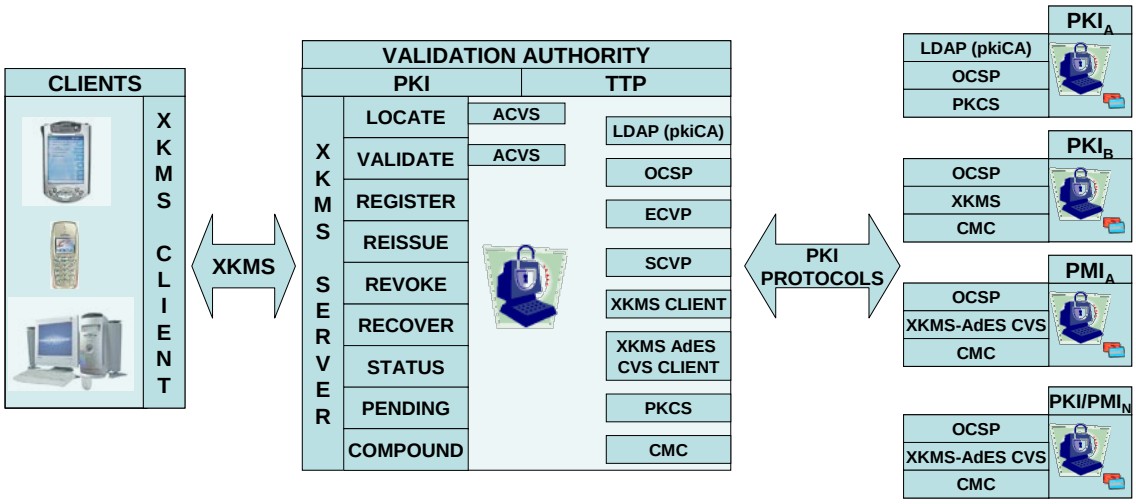

Figure 1: Architecture system

In this section we present the different elements that are part of the architecture where our service is integrated. In Figure 1 we show the elements that compose the system as well as the different relationships drawn between them.

These elements are: clients, the XKMS server that supports our extension (ACVS), the different Public Key Infrastructures (PKIs) and Privilege Management Infrastructures (PMIs), which issue the certificates to be validated and offer different validation mechanisms for this purpose. These elements communicate with each other by using different protocols like XKMS, LDAP, OCSP, etc. Each of these components is described in more depth in the following sections.

3.1. PUBLIC KEY AND PRIVILEGE MANAGEMENT INFRASTRUCTURES

PKIs (in Figure 1) are responsible for managing the complete lifecycle of identity certificates. One of the most important functions is to provide information about the status of a certificate by means of different mechanisms/protocols such as CRLs, delta CRLs, OCSP, SCVP, etcetera. We need to simplify how end users access those mechanisms.

In a similar way, PMIs are in charge of the management of attribute certificates. Usually, these certificates are issued for short periods of time and, therefore, it is not necessary to check their validation status. However, attribute certificates are sometimes issued for long periods of time, thus being required the provision of a mechanism to check them.

These infrastructures are mainly based on the use of certificates according to the X.509-based format. However, there are other proposals of certificate format, such as SPKI certificates.

3.2. CLIENTS

In this architecture, shown in Figure 1, clients contact the Validation Authority in order to check the status of a certificate (or some of them) at a given moment in time. There are two kinds of possible clients: *thin or lightweight clients* and *thick or powerful clients*.

Thin or lightweight clients are clients that have quite limited networking and computational resources (reduced memory, bandwidth and speed processor). In this category we could introduce some clients with mobile devices such as mobile phones, Personal Digital Assistants (PDAs), etc. These clients only need to know whether a certificate is valid at a given time (when the request is being performed or at a previous moment). Additionally, these clients need to know whether the certificate satisfies a specific policy at the moment when it is being verified. Thus, in their requests these clients send the minimum possible information (for example, instead of the full certificate they could send a reference to the certificate). Usually, the answer to their requests is only *valid* or *not valid*. In the event that these clients need the evidences used to validate the certificate, they could request the server to store them on their behalf. Later they could recover them from a more powerful client.

Thick or powerful clients are those that are sending their requests from powerful devices such as a personal computer, a laptop or a server. In general, these clients contact the validation authority because they need to recover the different evidences used to validate a certificate (for example, for building an AdES signature for some of the scenarios previously mentioned in Section 2). However, at the same time, they want to avoid the complexity of supporting different protocols for the tasks of certification path processing.

The clients request the certificate validation to the authority validation by using the XKMS protocol (see Figure 1). In particular, they make use of the *XKMS Validate* service. As commented in Section 2.3.2, this service only returns whether the certificate is valid or not. We have defined an extension to this XKMS service in order to recover the evidences used in the certification path processing to validate the certificate. We have defined it as an extension to maintain backward compatibility with current XKMS systems and to facilitate the process of migration to the new service we propose. This extension to the *XKMS Validate* service is presented in Section 4.

3.3. VALIDATION AUTHORITY

The Validation Authority is responsible for receiving the certification validation requests from the clients and answering them. In our proposed architecture, this entity is different from a PKI or a PMI. This entity would be a trusted third party for the clients.

Although we have established it as a separate entity, this certificate validation task could also be carried out by the corresponding PKI or PMI.

This validation authority implements an XKMS server with the services associated to this protocol (see Figure 1). In our proposal these services have been extended as we explain in the Section 4. In particular, these services are responsible for performing the certification path processing of a certificate to validate it. In order to carry out these tasks, the service needs several modules. These modules are described in the following subsection.

3.3.1. CERTIFICATE VALIDATION SERVICE MODULES

In Figure 2, we depict the different modules that are part of the architecture of this advanced validation service. This architecture is based on SAVaCert and in some interesting ideas from ECPV, such as the *Scheduler* and *Harvester* modules. Thus, we can improve the path validation. Furthermore, we have defined some new modules in order to satisfy the requirements that we need for this advanced validation services.

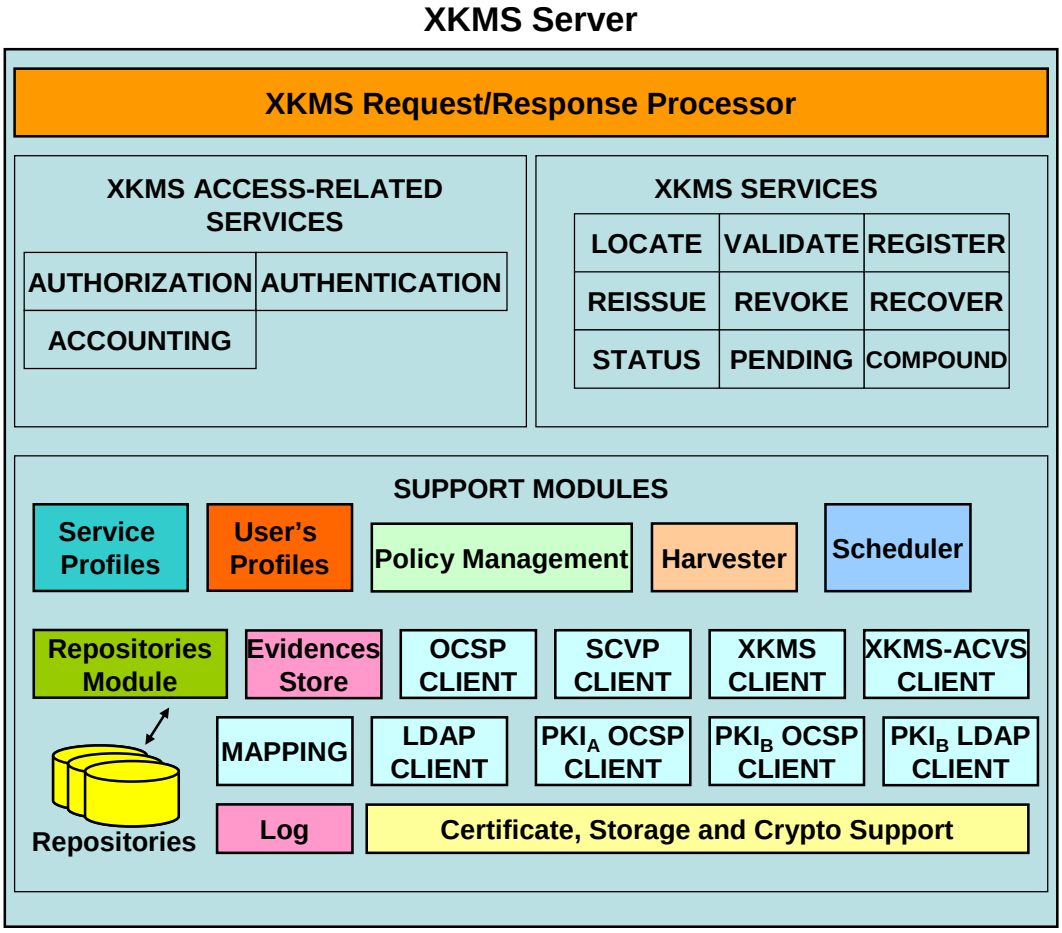

Figure 2: XKMS server service modules

Conceptually, there are four main groups of modules (each of them is marked in Figure 2):

- *XKMS Request/Response Processor*. This module is responsible for receiving the different requests from the clients and providing an answer. This module uses

XKMS access-related services to control the users' access to the service. When the request is processed, this is redirected to the specific XKMS service which is in charge of carrying out the task requested. In order to support the different tasks to perform, this module could make use of the *Support Modules*.

- *XKMS access-related services*. There are several modules that are in charge of performing different actions related to the access of the clients to the system, such as authentication, authorisation and accounting. Thus, the service could determine whether a client is able to use the service or not, the request comes from a valid client, and even it could take into account the different accesses made from the different clients in order to subsequently charge for their services (in the event of some services being charged for). The integration of these security services in XKMS could be based on SAML and XACML as proposed in [29].
- *XKMS services*. These modules are responsible for supporting the different functionality provided by the different services offered by XKMS, such as *Locate*, *Validate*, *Register*, *Reissue*, etc. The behaviour of these services is described in XKMS specification [16].
- *Support Modules*. The different modules in charge of supporting the different functionality of XKMS to carry out its tasks.

Next, we provide a clarification on the different modules that are part of the *Support Modules* group because they constitute the core modules needed to support the different functionality of the modules of the XKMS services group. These modules are:

- *Certificate, storage and crypto support*. This module offers all the functionality related to the processing of certificates, its storage, as well as all cryptographic functions (parser, verify, etc.) needed to process certificates, e-signatures, etc.
- *Service and User's profiles*. These modules are used to manage and support the different profiles supported by the server. A profile is useful to make requests simpler. The profile determines which information can be requested and should be answered in a response. For example, there could be a profile that returns the specific information needed for the XAdES format, others for the XAdES-X-L format, etc., or the information contained in the certificate from a particular PKI (user's name, passport ID, address, etc.). The profile could also determine the information to return for a specific type of client such as thin clients. In our system, to simplify requests, we have defined a profile for each AdES format (see Section 2.2 to know the evidences needed for each format). Thus, we do not need to specify the evidences explicitly to recover them. In the response, the server provides these evidences.
- *Policy Management*. This module is responsible for the different policies that can be used in the service. Specifically, a policy determines the behaviour for a specific profile.
- *Harvester module*. This module gathers from the CA repositories (e.g. LDAP servers), OCSP and SCVP responders some information needed to carry out both the path building and validation. This information comprises certificates, CRLs and OCSP responses, etc. and is stored in a repository called *harvested evidences*. Thus, we could have in advance some information which is used in the certification path validation and makes this process faster.

- *Scheduler module*. The scheduler interacts with the harvester module in order to have the validation information up to date.
- *Analyzer*. The analyzer is responsible for analysing the information obtained from the *Harvester* module.
- *Repositories*. It is the module that accesses the data base systems to store all the information that needs to be stored by the rest of the modules.
- *Evidences store*. This specific module, by using repositories modules, stores all the information needed for particular certification path validation operation. This information is stored so that the requester can subsequently have the information used in a validation and which at that specific moment could have not been returned (e.g. in the event, it is a thin client).
- *Log*. This module registers the different events that occur in the system in order to detect possible problems or attacks related to the behaviour of the different modules and its implementation.
- *LDAP (pkiCA), OCSP, SCVP, XKMS, XKMS-ACVS Clients*. These modules implement the LDAP (pkiCA), OCSP, SCVP, XKMS, XKMS-ACVS protocol according to its specification, respectively.
- *PKI_X LDAP (pkiCA), OCSP, SCVP, XKMS, XKMS-ACVS Clients*. Some PKIs, when implementing the different protocols, do not follow the specification completely or introduce some changes for authentication purposes. These modules are introduced in order to support slightly different versions of a standard protocol for a specific PKI.
- *Mapping*. This module helps to determine which module is used to perform the certificate validation. Most of the PKIs introduce the URL of the CRL distribution point, the URL of the OCSP responder, etc., in the certificate extensions. However, some certificates do not include this information. For this reason a module that allows the system to know which client should use to validate a certificate is needed. This module processes a configuration file in the server. This file indicates, for each PKI, which module has to be used to validate a certificate. This module is also used to determine the module and the URI to use in order to recover certificates from the reference to a certificate.

3.4. PKI PROTOCOLS

The PKI protocols are used by the validation certificate service to build the candidate certification paths and perform the subsequent validation.

Depending on the certificates involved in the validation process, the service requests validation information to the corresponding PKI. The protocols used depend on the mechanisms supported by every PKI. These protocols can vary from online validation protocols (OCSP, SCVP, etc.) to offline methods (CRLs, delta CRLs, etc.).

In each request to the service, this service uses the protocol that it considers the most adequate. This behaviour is modified whether the client specifies the protocols to be used in the validation. This functionality is fundamental for some scenarios, such as in e-commerce scenarios when some transactions, due to the risk to be assumed, require an online verification.

It is also important to mention that the XKMS protocol, with the extensions we propose, can also be considered a PKI protocol. Thus, our service could make a request, by means of XKMS with our proposal, and this request is processed by other entity such as a PKI or another validation entity. Thus, the delegation of the validation of some particular certificates is possible.

4. CERTIFICATE VALIDATION SERVICE SPECIFICATION

In this section we present the design of the Advanced Certificate Validation Service (ACVS) we propose. Firstly, we define the main goals the service should support in order to provide an advanced validation service. Next, we introduce the different working modes the protocol supports to interact with it. These modes are compatible with the XKMS specification. Then, we provide a detailed specification of the different extensions we have made to XKMS to support this kind of service. Finally, we provide an example of a validation flow to describe the working of the different components defined in our proposal to perform this certificate validation.

4.1. MAIN GOALS FOR AN ADVANCED VALIDATION SERVICE

This section defines the main goals and requirements that an advanced certificate validation service should fulfil. Regarding the goals, six objectives have been detected.

First, clients of a certificate validation service should be as independent as possible from the underlying protocols used during the validation process. As a second goal, legacy validation mechanisms must be supported; that is, it should be possible to perform different types of validation processes through this service (e.g. OCSP, SCVP), depending mainly on which validation mechanism has been implemented by the underlying infrastructure.

The third goal is the support of different kinds of certificates belonging to several infrastructures. At least, X.509-based certificates (both identity and attributes certificates) and SPKI certificates have to be supported.

As a fourth goal, a repository is mandatory in order to store the validation evidences for future non-repudiation checking operations. These evidences could be certificates, CRLs, OCSP and/or SCVP responses, among others. They depend on the validation mechanism chosen.

The fifth goal is related to policy and profile configuration. End clients must be able to select a specific policy and/or profile to configure the behaviour of the service. In some cases it could be interesting to perform any kind of validation, but sometimes only an online mechanism validation could be valid (e.g. in e-commerce platforms).

Finally, the last goal is the control access configuration. Thereby, we can determine what entities can use the service and how they can do it. In some organizations only certain clients are allowed (e.g. after paying a fee), whilst others have free access.

Bearing in mind the above goals, as well as the open issues discussed in Section 2, some needs and requirements are raised for an advanced certificate validation service. We next summarize those needs and requirements which a service with these features would have. The general requirements are as follows:

- The certificate validation service must fulfil the DPD and DPV requirements.

- This service must allow validating different types of certificates such as X.509 certificates, attribute certificates and SPKI certificates.
- The architecture of a validation service should be as generic and extensible as possible in order to enable the incorporation of new features in future releases.

Next, we summarize the requirements of mandatory fulfilment for the main actors involved in a validation process. On the client side, we have the following ones:

- Clients should be able to include intermediate certificates to ease the building process of the candidate certification paths.
- Clients have to be able to indicate whether they want the revocation status of the certificates included in the certification path to be verified by the server.
- Clients should be able to determine the precise time in which the validation process should be carried out.

On the server side, the following requirements should be fulfilled:

- Storing the validation data along with all information used during that process.
- Receiving the trust anchors in which clients trust to build the certification paths.
- Receiving indications about the protocol to be used in the validation process.
- Supporting service policies and profiles.
- Returning both validation information about the protocols used during the validation process and information about a particular format (e.g. XAdES).
- The server might establish that all requests are digitally signed by the clients in order to check if they are authorized to use the service or not.
- An asynchronous message exchange must be allowed.

Finally, regarding the information that needs to be exchanged between clients and server, which implements the certificate validation service, two main requirements are defined:

- Requests could indicate that the server stores information about the validation process, together with all information used by the server to carry out such a task.
- Requests must specify what information contained in the certificate should be returned in the response. Thus, this requirement eases the processing of the certificates by a client.

4.2. WORKING MODES

In this section we explain how we have extended XKMS, its services and modes in order to satisfy the goals defined in the previous section.

The basic element in a validation request is the certificate. Optionally, the client can specify a time to validate the certificate in and a validation policy. If the time is not provided, the server validates it at the reception moment. If a validation policy is not specified, the default policy is then applied. Moreover, clients can specify how the server should behave in the validation process. Thus, clients are able to select the validation mechanisms, or request additional validation information related to the

validation process, such as requests and responses of the validation mechanisms used, the path of the certificate up to a trusted point, a timestamp, etc.

There are different modes to implement this process. If the information requested can be provided in the response almost immediately or in a short interval of time, the XKMS synchronous mode is used. On the contrary, if the response cannot be provided in a short interval of time, the asynchronous mode will be used.

Additionally, in some circumstances, it could occur that the server wants to avoid denial of services attacks. For this purpose, the two-phase protocol has been provided in XKMS. In the following sections we explain how the different modes work and how we have extended the different requests and responses in order to support the goals defined in the previous section.

4.2.1. SYNCHRONOUS MODE

In XKMS the synchronous mode consists of a request and a response (see Figure 3). In this mode, the server has or can obtain, in an acceptable interval of time, the information requested by the client. Therefore. when the server obtains the information it creates a final response with the information requested.

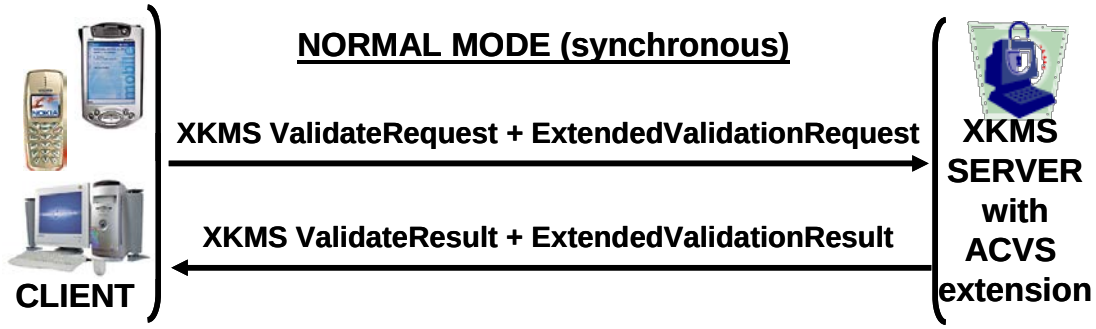

Figure 3: The *Validate* Service in the synchronous mode

In the validation of a certificate, according to XKMS, the client sends a *ValidateRequest* message and receives a *ValidateResult* message.

As for *Locate* service, the client sends a *LocateRequest* and receives the result in the *LocateResult*. All these messages (as well as all the messages of X-KRSS) extend the *MessageAbstractType* type. This last type contains an element called *MessageExtension* which was included to support the extensibility of XKMS. We mention next the elements which we defined to be included in this field in order to perform an advanced validation.

In the synchronous mode, for the *Validate* service (see Figure 3), we include a new element in the request called *ExtendedValidationRequest* into the element *MessageExtension* of the *ValidateRequest* element.

In the *ValidateRequest* we include the certificate to validate (or a reference to its public key) and, optionally, the instant of time in which we want to perform the validation. In the *ExtendedValidationRequest* we include the rest of information needed to perform the validation process (protocols to use, responses to obtain, etc.). More details about the information which is included in this element are provided in Section 4.3. As a response to this request (see Figure 3), we return a *ValidateResult* which contains an *ExtendedValidationResult* in its *MessageExtension* field. The *ValidateResult*

only returns whether the certificate is valid or not in the period of time requested. The rest of information needed to recover for an advanced validation (responses, CA certificates, etc.) is provided in *ExtendedValidationResult*. More details about the information that is included in this element are provided in Section 4.3.

4.2.2. ASYNCHRONOUS MODE

This mode is used when the server receives a request which cannot be completed immediately. In this case, the client receives a response indicating that its request is pending and that he has to wait in order to obtain it. In particular, for the advanced validation of a certificate, the process is depicted in Figure 4.

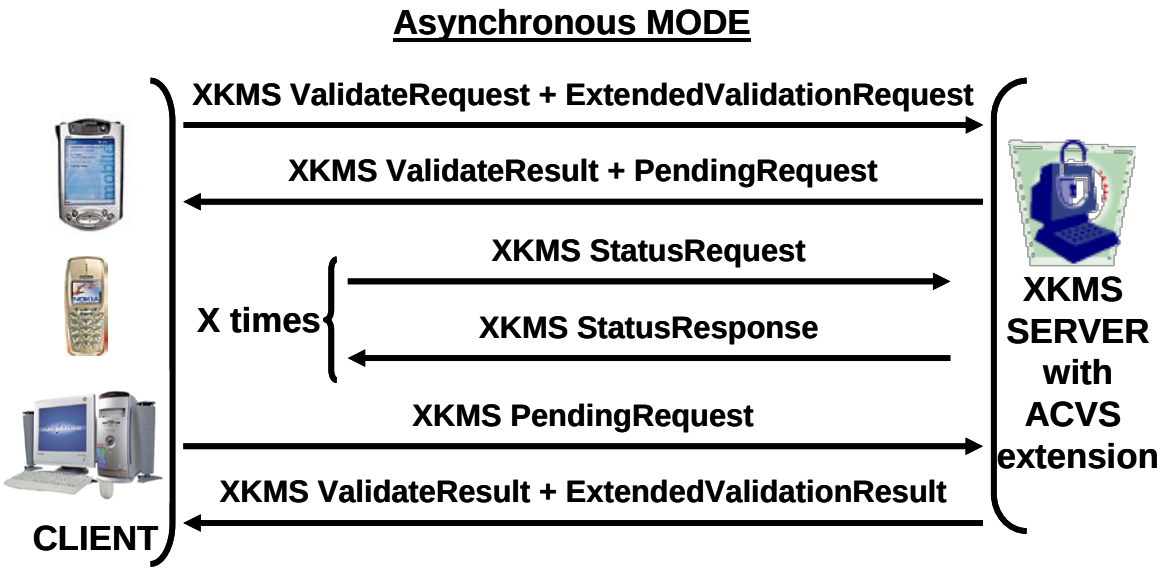

Figure 4: The *Validate* Service in the asynchronous mode

In this process the client requests the advanced validation by sending the *ValidateRequest* with the *ExtendedValidationRequest*. As a response, the server sends the *ValidateResult* indicating with the *PendingRequest* value (in the *ResultMajor* attribute) that the request cannot be processed immediately. In this response, it is also included a request identifier (*RequestId*) to subsequently check the status of this request.

From the moment the client receives the *PendingRequest* value, he has to check periodically whether the response is available or not. This checking would not be necessary whether the client in his request would have specified a notification mechanism such as the e-mail. For the purpose of checking the status of a request, in XKMS, the *StatusRequest* and *StatusResponse* messages are defined. When the response is available the server indicates it in the *StatusResult* with a *Success*.

The client sends a *PendingRequest* request to the server to recover the response that is already available. As a response the server sends a *ValidateResult* with the *ExtendedValidationResult* in the *MessageExtension* element.

The asynchronous processing for the *Locate* service would follow the same behaviour we have just described. Thus, in message 1, we use the *LocateRequest* with the *ExtendedLocationRequest*. In message 2, the server answers the *LocateResult* with the *PendingRequest*. The messages 3, 4 and 5 would be the same. Finally, in message 6, the server sends the response with the *LocateResult* and the *ExtendedLocationResult*.

4.2.3. TWO-PHASE PROTOCOL

In XKMS, the two-phase protocol has been defined to protect the service against denial of service attacks. This protocol performs a lightweight authentication just before satisfying a request. The goal of this protocol is to check that the client is able to process the responses provided by the server.

Thus, in the first phase, when the service receives a request (see Figure 5), the server replies with a response indicating that the two-phase protocol has to be executed (*Represent* value in the response) and also sends a nonce.

In the second phase, the client sends the original request with the nonce. If the nonce is the same that the one provided in the first phase by the server, then, the response is now sent. This two-phase protocol can even be combined with the asynchronous mode defined in XKMS.

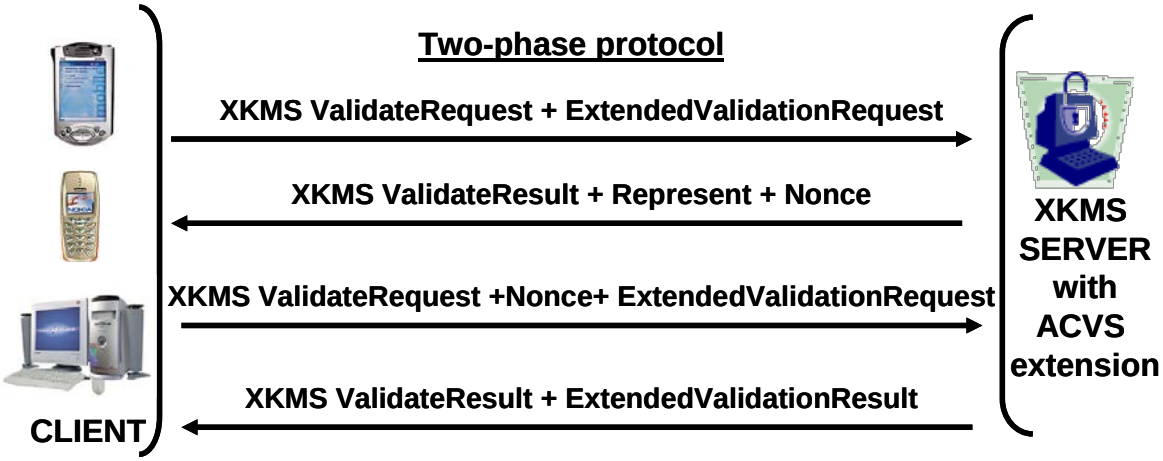

Figure 5: The XKMS two-phase protocol with our extensions

The use of the two-phase protocol with the service we have defined is the same as specified in XKMS, and there is not any problem with the extension introduced here. This is due to the fact we have extended the requests and responses defined in XKMS and which are used in this protocol. Our extensions are sent with these requests and responses as can be seen in Figure 5. However, we are going to comment some aspects of the authentication process.

The lightweight process of authentication is based on the exchange of several messages as we have just described. Therefore, when using this protocol a request has to be sent twice. A stronger way of authentication is to use an electronic signature in each message. However, this process supposes a higher overload to the server. We propose next a hybrid approach between these previous ways of authenticating the user for those cases when the client usually works with a certificate validation service.

Our proposal of authentication is based on the use of Paywords, which are chains of hash values. This proposal was introduced by Rivest in [21] to make micropayments. The idea consists in the client sending a hash value called root (w_0) to the vendor. This root value is the result of executing *n* times a hash value over a starting value (w_n). Thus, $w_0=h_n(w_n)$. This root value is signed and sent to the server. Thus, each time the client has to access the service, the client reveals a new value of the chain, from the first value (w_1) to the n-value (w_n) in turn. The service could know that the hash value comes from the client because in a hash function the reverse operation is impossible to perform

if we suppose perfect cryptography. That is, it is very difficult to calculate the value used to make the hash from the hash value.

We propose that when the service needs to authenticate the client in order to avoid denial of service attacks, that client signs the request without a root value of a Paywords chain. This root value is used as a nonce. Then, the server verifies the signature and stores this nonce. In subsequent requests the client sends directly the following Payword. Thus, we avoid that the service has to send us a response with a nonce indicating that an authentication is required.

With this mechanism, in the first authentication, the number of messages exchanged between client and service is the same that the two-phase protocol proposed by XKMS. Subsequent times the server does not need to execute the two-phase protocol, and therefore the number of messages to exchange is reduced. As for the security, we can say that it is the same as in the previous proposal because both proposals are based on the use of nonces.

4.3. EXTENSIONS

According to XKMS specification, there exist two kinds of messages in the *Validate* service: *ValidateRequest* and *ValidateResult*; similarly, there exist two types of messages in the *Locate* service: *LocateRequest* and *LocateResult*. All these messages extend the *MessageAbstractType,* which includes the element *MessageExtension* that was thought as an extension mechanism for XKMS messages.

Taking this into account, in order to reach the goals outlined in Section 4.1, and as we have mentioned in the previous section, we have extended XKMS *Validate* and *Locate* services by defining four new elements that suit the *MessageExtension* field perfectly. These new elements are called *ExtendedValidationRequest* for the *ValidateRequest* message, *ExtendedValidationResponse* for the *ValidateResult* message, *ExtendedLocationRequest* for the *LocateRequest* message and *ExtendedLocationResult* for the *LocateResult* message. The following subsections describe each of these new elements in more detail.

4.3.1. *ExtendedValidationRequest*

When a client wants the service to validate a certificate, he sends the service a *ValidateRequest* message. This message includes information regarding that certificate within the element *QueryKeyBinding*; optionally, this element also includes within the *TimeInstant* subelement, the time at which the certificate should be validated. Furthermore, if the client is interested in obtaining information associated to the validation process (such as the specific validation mechanism that should be used or the indication that the complete certification path validation should be accomplished) and/or related to the information used in the validation process that should be retrieved (such as the complete certification path) and stored (in order to be used later, as a proof of existence of such validation evidences), the *ValidateRequest* message includes the *ExtendedValidationRequest* field. This field is composed of several elements, as shown in Figure 6.

As we can see, the *ExtendedValidationRequest* field has two attributes: *Profile* and *ReturnCertificate*. The former allows a user to indicate the server a specific profile to be used in the validation process. A profile usually establishes a context that requires a predefined behaviour, thus simplifying the number of parameters indicated in a

message. The latter indicates if the client wants the validated certificate to be returned within the response, as it is not mandatory to include the complete certificate within the request (it can include a reference to the certificate, such as an URI or its *IssuerAndSerial*). Additionally, the *ExtendedValidationRequest* consists of several subelements.

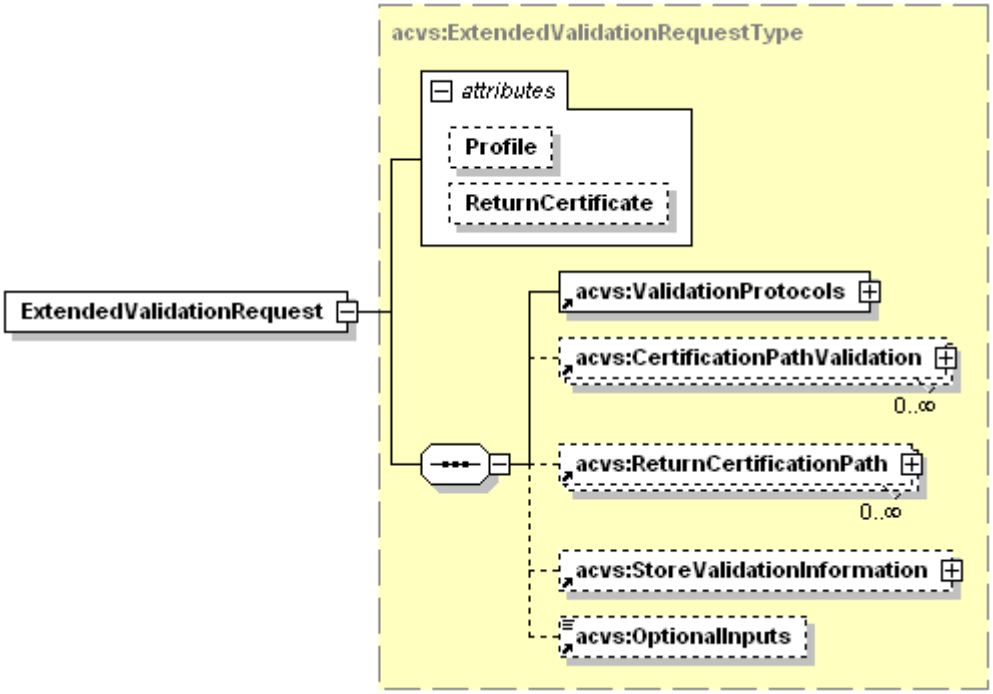

Figure 6: *ExtendedValidationRequest* element

The *ValidationProtocols* subelement defines the validation mechanisms that should be used in the validation process. The server can use the first available mechanism or even all of them. It is composed of a set of elements of type *ValidationProtocol* that includes the mechanism identifier, the priority which mechanism should be applied with, the validation response returned by this mechanism, the details regarding that response, etc.

The subelement named *CertificationPathValidation* indicates that the server should accomplish the certification path validation from the target certificate up to one or several trust anchors. It also indicates whether validation evidences should be returned. Additionally, the client can provide intermediate certificates to ease the building process of candidate certification paths.

The subelement named *ReturnCertificationPath* indicates that the server should return one or several certification paths up to one or more trust anchors. These certification paths can be different from those validated by the server. Furthermore, this subelement allows the client to indicate if these certification paths should be validated by the server, although no evidences will be returned (certification path validation evidences could only be returned if specified in *CertificationPathValidation* subelement).

The subelement named *StoreValidationInformation* expresses that the evidences used in the certificate validation process should be stored in the own XKMS server or in an external secure server. By default, when we include this element, the information is

stored in the XKMS server. We also provide the possibility of specifying the information needed for the external server.

A secure external server that would be used is an LTANS-based server. As the use of this kind of server is a suitable proposal, and LTANS [30],[31],[32] might become standardised, we have defined the information that we would need to specify with this proposal. Furthermore, we have defined an element for extensibility purposes which would enable it to support other future proposals that could appear. Thus, according to LTANS, we allow the client to specify the access address to that LTANS server, the client identifier to gain access to that server, the LTANS server identifier to store the validation evidences in, the set of policies to be applied when archiving the validation evidences, the validation evidences that should be archived, etc.

This element supports that clients (especially thin clients) can store validation information that could be used and recovered subsequently. Thus, a client could indicate that he is interested in performing the validation. However he does not want to obtain validation evidences now but he wants to store them to recover them later. For this purpose, once the information is stored the service provides a stored evidence identifier in the response. Subsequently, when the client wanted to recover the evidences, he would send a request with the certificate and the stored evidence identifier in the *OptionalInputs* subelement.

Finally, the *OptionalInputs* subelement has been defined for extensibility purposes and can contain any type of element. Currently, this field is used to request stored evidences as we have just described.

4.3.2. *ExtendedValidationResponse*

When a client sends a *ValidateRequest* message, the service responds by sending a *ValidateResult* message. This message indicates whether the specified certificate is valid or not at the moment requested. If the *ValidateRequest* message sent to the service included the *ExtendedValidationRequest* field, the service would send, as a response, a *ValidateResult* message (as the one aforementioned) with the element *ExtendedValidationResponse*. In this case, the existence of this element is mandatory. The *ExtendedValidationResponse* field is used to carry the data asked in the request by means of the *ExtendedValidationRequest* element, such as the complete certification path, the validation mechanisms responses etc. This field is composed of several elements, as shown in Figure 7.

As we can see, the *ExtendedValidationResponse* field has one attribute, called *Profile*, which indicates the profile used by the server when validating the specified certificate. Generally, the value of this attribute is equal to the corresponding attribute in the *ExtendedValidationRequest* field within the *ValidationRequest* message.

The subelement named *CertificateValidity* includes the validated certificate as well as information related to the certificate validation process, the specific validation mechanism used, the validation responses and the reference to the certificates within the certification paths from the specified certificate until one or several trust anchors.

The subelement *CertificationPaths* can include one or more certification paths from the specified certificate up to one or several trust anchors. It can also include validation information regarding every certificate within a certification path. Finally, the *OptionalOutputs* subelement has been defined for extensibility purposes.

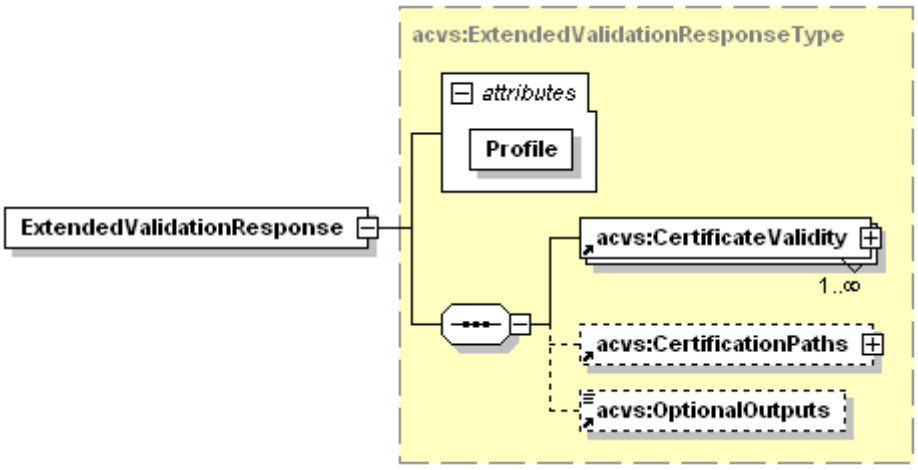

Figure 7: *ExtendedValidationResponse* element

4.3.3. *ExtendedLocationRequest*

When a client wants the service to locate a certificate, he sends the service a *LocateRequest* message. Similarly to the *ValidateRequest* message, it includes information regarding the certificate within the element *QueryKeyBinding* and it also includes, optionally, the time the certificate should be located at, within the *TimeInstant* subelement. Furthermore, if the client is interested in obtaining information included within the specified certificate (such as the name of the subject, the identification number of the subject, the identification of the certificate issuer, etc.), the *LocateRequest* message will include the *ExtendedLocationRequest* field. This field is composed of several elements, as shown in Figure 8.

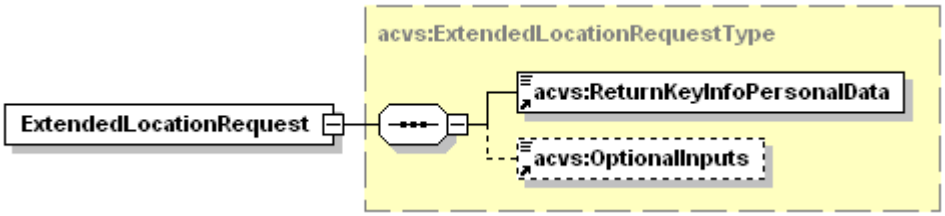

Figure 8: *ExtendedLocationRequest* element

The subelement named *ReturnKeyInfoPersonalData* indicates that the client is interested in recovering information related to the certificate or the certificate owner. The *OptionalInputs* subelement has been defined for extensibility purposes.

4.3.4. *ExtendedLocationResponse*

When a client sends a *LocateRequest* message, the service responds by means of a *LocateResult* message. This message mainly provides the complete certificate that was referenced within the *LocateRequest*. If the *LocateRequest* message sent to the service included the *ExtendedLocationRequest* field, the service would send, as a response, a *LocateResult* message (as the aforementioned one) with the element *ExtendedLocationResponse*. In this case, the existence of this element is mandatory. The *ExtendedLocationResponse* field is used to carry data asked in the request, specifically, within the *ExtendedLocationRequest* element. This field is composed of several elements, as shown in Figure 9.

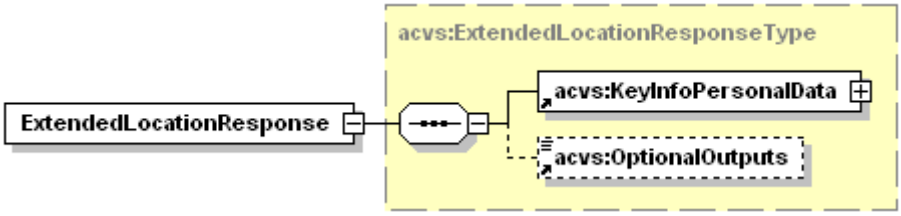

Figure 9: *ExtendedLocationResponse* element

The subelement called *KeyInfoPersonalData* includes information related to the specified certificate (for example, its description), the certificate owner (for example, its identification number, name, surname, email, date of birth, etc.), the entity represented by the certificate owner (for example, the entity identifier, the entity name, the entity domain, etc.) or the existing relationship between the certificate owner and the entity (for example, the owner position within the entity, the entity unit, etc.). This subelement includes the field named *Other* for extensibility purposes. The subelement called *OptionalOutputs* has been also defined for extensibility purposes.

4.3. VALIDATION FLOW EXAMPLE

In this section we show the interaction and the main operations which take place when a user using a (thin) client wants to validate a certificate. Let us suppose that a thin client wants to validate a certificate and obtain the evidences associated to this certificate for, subsequently, creating an AdES signature. In this example, we suppose that a thin client provides the validation authority with the reference to the certificate and wants to obtain the certificate, CA certificate, its validation evidences by means of OCSP and he also wants to store all this information in a LTANS server. For the shake of simplicity we also suppose that the client is able to make the request without authentication and using the two-phase protocol. The steps that would take place appear in Figure 10.

In step 1, the thin client creates a *ValidateRequest* indicating he wants the validation of the certificate which is referenced in the request. He also provides the information needed to store the associated information validation in a LTANS server.

The server checks (in step 2) that the client is able to make the requests. The request is processed in order to determine the kind of request (step 3). Once the request is a *ValidateRequest*, it is forwarded to its associated module (step 4) to process it.

The request specifies the validation in the terms we have explained at the beginning of this section. Thus, the server queries (in step 5), in the harvested evidences repository, whether the evidences requested (certificate and CA certificate) had been obtained in an anticipated way for the *Harvester* module. In this example, we suppose these evidences are not stored in this repository. Thus, the server orders the *Scheduler* module (step 6) to schedule in order to obtain these evidences for future validations (creating also the CPVTs trees, see Section 2.3.1).

As the evidences are not stored, the server proceeds to recover them. For this purpose, it checks the module and the URI in the *Mapping* module to obtain the evidences for the certificate requested (step 7). In this case, the module used to recover them is the LDAP protocol. With the LDAP protocol (step 8), the server obtains the certificate and the CA certificate from the corresponding PKI.

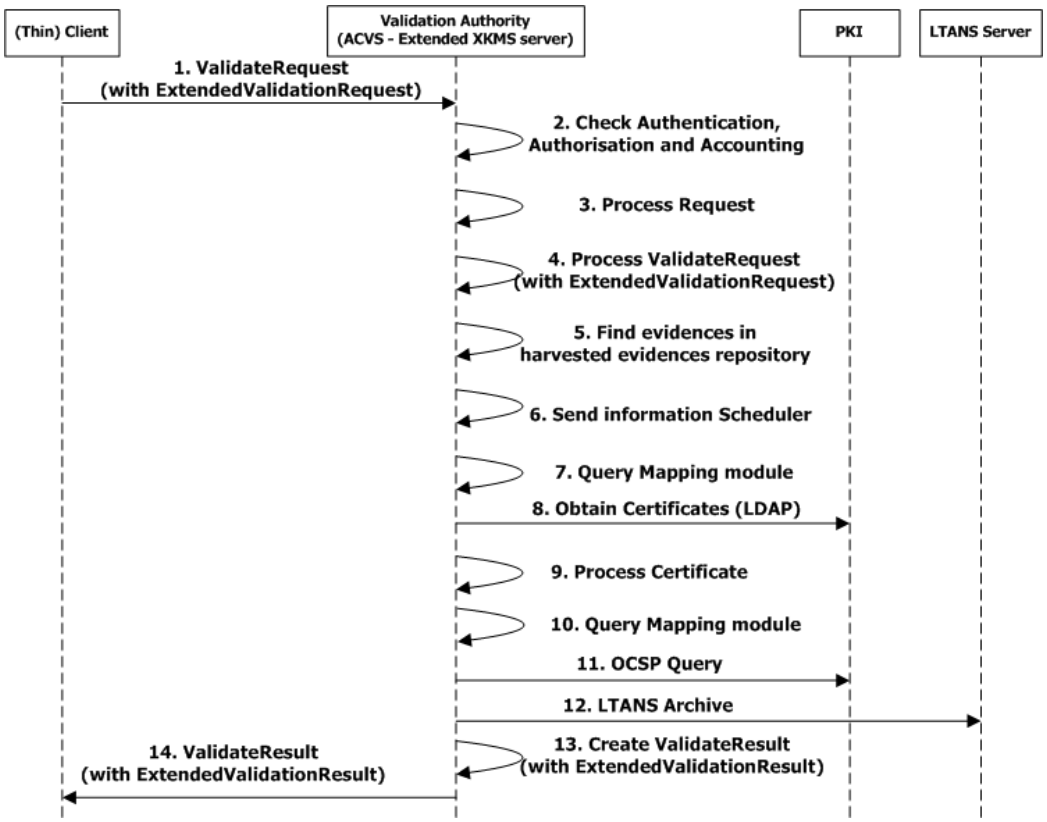

Figure 10: Validation example flow

To check the validity of the certificate, the server processes the certificates to be validated (step 9). In these certificates the OCSP responder to be used is specified. The server additionally checks in the *Mapping* module whether a specific OCSP module should be used. In this case, we suppose that this is not necessary since the PKI offers this service according to the standard and without any special authentication. Thus, the server makes use of the OCSP client to check the status of the certificates (step 11). All the evidences obtained (certificates and OCSP responses) are stored in the LTANS server specified by the user (step 12). Thus, these evidences could later be recovered by the user from a more powerful client. Finally, the *ValidateResult* (with the *ExtendedValidationResult* providing a stored evidences identifier is provided) is created (step 13) and sent (step 14) to the thin client.

5. AN SOA-BASED ADVANCED ELECTRONIC SIGNATURE ARCHITECTURE

In this section we describe the main elements that compose the SOA architecture of the University of Murcia for the e-government processes where the advanced electronic signature is involved. Next, we focus on a real scenario for the university community, such as the generation and signing of electronic mark certificates.

5.1. DESIGN OF THE ARCHITECTURE

The e-government SOA Infrastructure of the University of Murcia (Figure 11) is composed of several elements. Some of them are hosted by the University and the rest by trusted third parties such as certificate service providers. We introduce next those

which will be necessary to understand the scenario of use presented in the following section.

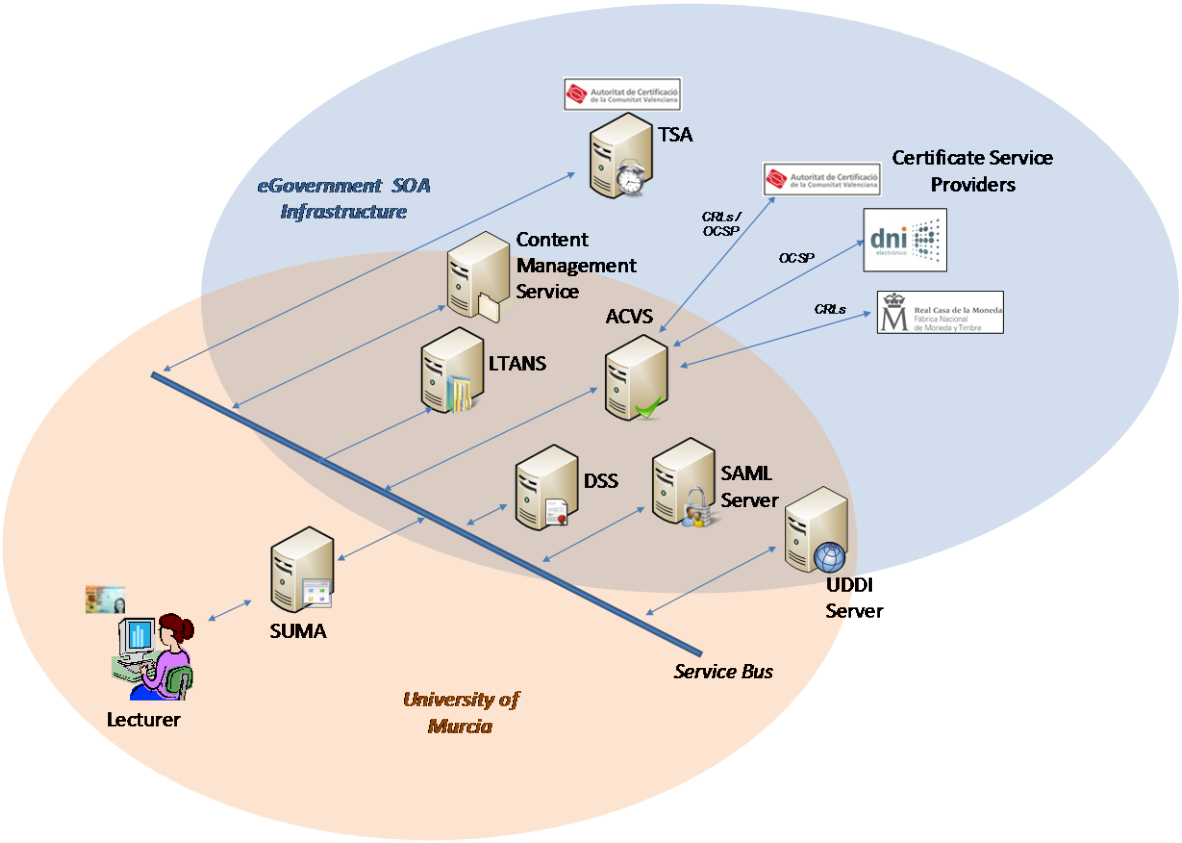

Figure 11: e-government SOA infrastructure

The Digital Signature Services (DSS) server is the pivotal element of the infrastructure. This server is able to sign and verify signatures on different formats (PKCS/CMS, XMLDsig, PDF, AdES formats) and support different profiles of use (profiles for different options of PKCS/CMS/XMLDSig/AdES formats, different parameters, extensions to include, etc.). The service has been developed according to the OASIS DSS [33] specification that defines it as a Web service. This service is invoked by all the different applications and services which require the incorporation of electronic signature processes in the University, either for validating any kind of e-signatures and timestamps or for producing signatures on behalf of the University. Some of these applications are the virtual campus (also known as SUMA), the official electronic noticeboard system, the electronic registry and the annual university enrolment application. In the e-signature processes, our most commonly used profile is able to perform the verification process, add a timestamp to the verified e-signature and store it in a secure archive system based on LTANS.

One of these applications is SUMA [34]. SUMA is the virtual campus of the University of Murcia. It provides to lecturers and students a set of tools to improve the learning of students as well as it facilitates this process can be carried out everytime, everywhere. In SUMA students can see contents and syllabuses, chat with lecturers, ask questions related to the subject, query their marks, etc. It also offers additional services to the university community, some of them specifically for lecturers, such as the generation and signing of electronic marks certificates.

The Security Assertion Markup Language (SAML) [35] server is the element that performs all the authentication and authorization processes through the infrastructure. This server exchanges standard XML assertions with applications and services in the university, for the performing of access control decisions.

The Advanced Certificate Validation Service (ACVS) is the element that carries out the validation of all the certificates used in the electronic signature processes. In the context of the University of Murcia, it is able to perform a validation operation against three main certificate services providers. The first one is the Dirección General de Policía (DGP) -Spanish state police direction- for the validation of the Spanish National Identity Card through the OCSP protocol. The second and third ones are the Fábrica Nacional de Moneda y Timbre (FNMT)-the Spanish state certification service provider-, which is the main certificate service provider in Spain and provides CRLs for performing the validation process, and, finally, the Agencia de Certificación de la Comunidad Valenciana (ACCV) -a regional certificate service provider-, which provides both CRLs and OCSP validation services. The University of Murcia has an agreement with the ACCV not only for the provision of digital certificates, but also for consuming its timestamp service. This service, which is developed according to the RFC 3161 [36], is offered through its Time Stamping Authority (TSA), which is qualified by the Spanish Government.

The Long-Term Archive and Notary Services (LTANS) server is the element in charge of archiving and preserving the electronic documents over long periods of time. The LTANS server periodically timestamps the archived documents in order to prevent weakness of the cryptographic material. It also follows the recommendations of the IETF working group for the archive protocol and the data structures [30],[31],[32].

5.2. SCENARIO OF USE: ELECTRONIC MARKS CERTIFICATE

Based on the previous elements, we describe now how they are used in a scenario of use: the signing process of a mark certificate. This process, which is depicted in Figure 12, is performed by a lecturer within SUMA and the eGovernment SOA Infrastructure of the University. It takes place at the end of the term when the lecturer wants to publish the official marks that are sent to the secretariat of the University.

At the beginning of the process, the lecturer signs the electronic marks certificate by means of a qualified certificate of the Spanish National Identity Card (step 1). This signed document is received by the SUMA application, which initiates the verification of the electronic signature and the archive of this document (step 2). At this point, the DSS server performs several tasks in order to carry out this process.

As part of these tasks, the DSS server requests an authorization to the SAML server (step 3). This server checks the attribution of the SUMA application for invoking a verification and archive operation. Next, DSS checks the revocation state of the electronic certificate used for signing the document through the ACVS server (step 4). The ACVS server is able to establish an online connection with the certificate service provider for performing the validation process with a standard mechanism, like OCSP (step 5). Once the verification is performed, the DSS server requests a timestamp token to the TSA (step 6) with the aim of guaranteeing the validation time. Thus, the DSS server creates a signature according to AdES-T format (including the timestamp) to guarantee the non-repudiation of the document and the date it was produced.

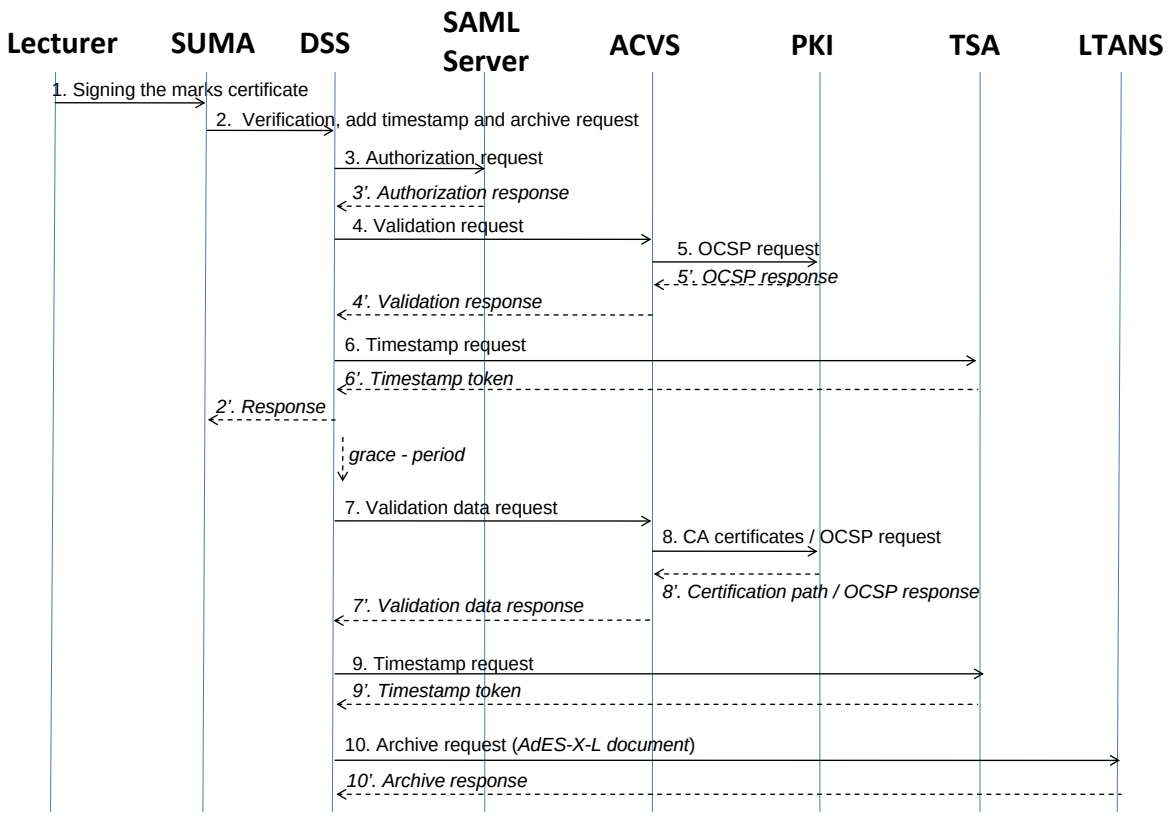

Figure 12: Electronic mark certificate process

AdES is a standard proposal of the ETSI which defines several profiles and formats for the advanced electronic signature, in terms of European Directive 1999/93/EC. More details on AdES formats were described in Section 2.2. The University of Murcia has adopted this specification for the different electronic signed documents that must be archived and preserved. AdES also introduces the grace-period concept, as the period of time that the verification system must wait for until the validation evidences of the electronic document are available.

After this grace-period, DSS begins the second validation process (step 7). In this process, the recovering of all the validation data of the digital certificate is essential. For this reason, a second communication with the certificate provider is performed (step 8). In this connection the trusted certificate chain of the lecturer certificate, and a signed OCSP response which results from the checking about its revocation state, are provided.

Then, DSS requests a new timestamp token to the TSA (step 9) for protecting the certificate chain obtained in this second validation process. Finally, an extended AdES format (AdES-X-L) is generated and sent to the LTANS server for the archiving and long-term preservation of this electronic information (step 10). LTANS securely stores this document and the process finishes.

5.3. DETAILS OF THE IMPLEMENTATION

In this section we provide a description of the different elements and components that we needed to develop the architecture and the scenario presented in previous sections. It is worth mentioning that our developments are based on open source code and mainly

based on the Java programming language because it is object-oriented and the code can be used in several platforms.

The different Web servers involved in the architecture are based on the OAS 10g application server. We have based the support of Web services used in the different servers (DSS, LTANS, SAML, ACVS) on Apache Axis library, which provides an implementation of the SOAP server as well as several tools and APIs for the development of Web service applications.

For the development of the different software we have made use of the Eclipse Open Source IDE. In our developments, we have used Bouncy Castle Crypto APIs as cryptographic library. This library is used for the basic cryptographic operations (encryption and signature) and algorithms as well as supporting the main ASN.1 standards formats such as PKCSs, CMS, RFC 3161, OCSP, CRLs, etc. For XML formats related to cryptography we have used Apache XML security library. Based on these libraries we have developed the DSS service based on OASIS specification. The DSS service also uses the iText PDF library for PDF documents. The development of the SAML server is based on OpenSAML.

The development of ACVS, which is presented in this paper, is based on the OpenXKMS library [37] and the extensions we have developed for this library. Our implementation uses OpenLDAP for the access to LDAP servers, Bouncy Castle for CRLs and OCSP, and our own implementation of SCVP [38]. It is worth mentioning that most of PKIs are totally compatible with these standards. We also make use of a specific library for the access to the certificates provided by the FNMT because the method it offers is not completely based on LDAP. The reason is that they provide a restricted access to this service. Thus, this service is not available to citizens. FNMT only provides access to public organisms (following an agreement). The LTANS service has been developed according to the Internet draft [31],[32]. SUMA is developed by using J2EE technologies.

Finally, as database for storing evidences, information of the different applications such as SUMA, ACVS, LTANS, etc. we are using Oracle 10g.

6. RELATED WORK

This section analyses some related work with respect to the different components of the architecture presented in Section 3 and the XKMS solutions presented in current literature. As we have commented above, the architecture presented here is based on SAVaCert [19] and some modules (mainly the *Scheduler* and *Harvester* modules) from ECPV [20]. Our proposal relies on these architectures for the validation phase of certificates, and we have extended their main core to include some modules and features that we need to support the XKMS-based advanced validation required to build AdES signatures as well as to store the validation evidences associated to this process. Furthermore, we have presented a complete implementation of this proposal, which is being currently used at the University of Murcia.

On the other hand, regarding the current XKMS solutions, many have been the implementations that have appeared since the stable and mature release of XKMS (version 2.0) was recommended in May 2005. Almost all participants and contributors to the design stage of this specification have developed their own prototypes, or even a complete reference implementation of it. These companies are VeriSign, as a leading

editor, Microsoft Corporation, RSA Security, SQLData Systems and DataPower Technology, among others.

VeriSign's implementation [39] allows key functionalities of PKI to delegate trust decisions for authentication purposes. The XKMS Web service where Web servers can send theirs requests is located at [40]. On the other hand, Microsoft has developed an XKMS client and server on ASP.NET [41] with the aim of providing SOAP message-based accesses to PKI services. SQLData Systems provides a live test server for demonstrations and interoperability tests [42]. A client side implementation of XKMS 2.0 is also provided by SQLData Systems, which is written in C/C++ as a COM object to be used in VBScript and/or ASP pages. Finally, DataPower Technology provides XKMS support to its XS40 XML Security Gateway [43], which can act as an XKMS client during the certificate validation phase with the aim of increasing the security in Web services transactions.

Apart from previous implementations, other interesting commercial implementations can be found. For instance, the TrustFinder XKMS Server [44], developed by Ascertia, provides an XML-based server for locating and validating certificates. The building and validation of the candidate certification paths can help the TrustFinder SCVP Server and/or the TrustFinder OCSP Server, both also developed by Ascertia.

Regarding open source implementations of XKMS 2.0, only OpenXKMS [37] has appeared. This WS-XKMS solution, widely explained and detailed in [45], defines a complete design and implementation of this specification. Moreover, in [45] the authors include a new module to the XKMS engine, called *PKIForXKMS*, which is a connector used to add new specific underlying PKI implementations. The XKMS engine will choose the proper *PKIForXKMS* connector, which is in charge of interacting with the final PKI solution, according to the protocols implemented by this PKI. We have used and extended this solution for our purposes, explained throughout this paper, since this solution is open source, quite complete in its definition and implementation, and it is based on Java so that it can be executed under several operating systems. A limitation of this solution is the fact that OpenXKMS associates to a concrete domain the use of a specific module (the proper *PKIForXKMS* connector) manually. Our proposal is more generic by looking for the access information to the OCSP responders, CRL distribution points, etc. directly in the certificate extensions through a PKI-independent module. In case this information is not defined, a specific module is used as OpenXKMS proposes.

Finally, it is worth mentioning that there exist a lot of XKMS prototypes that can be used by the research community for interoperability testing purposes. Among others, we emphasize Wings of Hermes [46], which provides an online XKMS server accessible through different URLs.

7. CONCLUSIONS AND FUTURE WORK

In e-commerce, e-business and e-government, AdES signatures are fundamental in order to guarantee the integrity and non-repudiation of a signed document over long periods of time. To build these signatures we need to incorporate some validation evidences. These evidences are needed so that the e-signature and the certificate used in this signature remain valid for a long time after their creation.

These evidences could be obtained by means of XKMS when we validate the certificate. However, XKMS only provides a simple response indicating whether the certificate is valid or not. We have extended XKMS in order to able to recover

validation evidences. Our extension is based on the extensibility mechanisms provided by XKMS. Thus, we facilitate the incorporation of our ACVS extension to the current implementations of XKMS. Furthermore, we have defined the system architecture and the different modules that a server providing this functionality should support. The functionality provided has divided clients into two kinds: thin clients, with limited capabilities; and thick clients with powerful capabilities. We have also proposed a lightweight authentication to facilitate long-term relationships between clients and the server. Finally, our proposal is extensible and could be improved with new features.

Our proposal has been developed and is being used by supporting different PKIs. This development has also been incorporated in the e-government platform developed in the University of Murcia. In this platform, as shown previously, this component is fundamental for the generation of AdES signatures and is related to the signature service.

Finally, as regards future work, we are studying how to improve the infrastructure for the access to the services provided in the platform with Single Sign On. We are also studying how to define a federation between different validation authorities based on XKMS servers.

REFERENCES

[1] European Parliament. Directive 1999/93/EC of the European Parliament and the council of December 1999 on a Community framework for electronic signatures. In Official Journal of the European Communities. 2000.

[2] United Nations. UNCITRAL Model Law on Electronic Signatures with Guide to Enactment. In United Nations Publications. 2002.

[3] Comité Européen de Normalisation - European Committee for Standardization (CEN). E-invoices and digital signatures. In CEN Workshop Agreement (CWA) 15579. 2007.

[4] Myers M, Ankney R, Malpani A, Galperin S, Adams C. X.509 Public Key Infrastructure: Online Certificate Status Protocol - OCSP. IETF RFC 2560, June 1999.

[5] Cooper D, Santesson S, Farrell S, Boeyen S, Housley R, Polk W. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. IETF RFC 5280, May 2008.

[6] Pinkas D, Pope N, Ross J. CMS Advanced Electronic Signatures (CAdES). IETF RFC 5126, February 2008.

[7] Cruellas JC, Karlinger G, Pinkas D, Ross J. XML Advanced Electronic Signatures (XAdES). W3C Recommendation, February 2003. http://www.w3.org/TR/XAdES [6 August 2009]

[8] RSA Laboratories. PKCS #7: Cryptographic Message Syntax Standard. An RSA Laboratories Technical Note. Version 1.5, November 1993.

[9] Housley R. Cryptographic Message Syntax (CMS). IETF RFC 3852, July 2004.

[10] Word Wide Web Consortium (W3C). XML-Signature Syntax and Processing. In W3C Recommendation. February 2002.

[11] Cooper M, Dzambasow Y, Hesse P, Joseph S, Nicholas R. Internet X.509 Public Key Infrastructure: Certification Path Building. IETF RFC 4158, September 2005.

[12] Lloyd S. Understanding Certification Path Construction. PKI Forum White Paper, September 2002.

[13] Wahl M, Howes T, Kille S. Lightweight Directory Access Protocol (v3). IETF RFC 2251, December 1997.

[14] Freeman T, Housley R, Malpani A, Cooper D, Polk W. Server-Based Certificate Validation Protocol (SCVP). IETF RFC 5055, December 2007.

[15] Perlines Hormann T, Wrona K, Holtmanns S. Evaluation of Certificate Validation Mechanisms. Computer Communications, 29(3):291-305, February 2006. DOI: 10.1016/j.comcom.2004.12.008.

[16] Hallam-Baker P, Mysore SH. XML Key Management Specification (XKMS 2.0). W3C Recommendation, June 2005. http://www.w3.org/TR/xkms2 [6 August 2009]

[17] Jiang M, Willey A. Service-Oriented Architecture for Deploying and Integrating Enterprise Applications. In Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05), pp. 272-273, 2005. DOI: 10.1109/WICSA.2005.60

[18] Papazoglou MP, Traverso P, Dustdar S, Leymann F. Service-Oriented Computing: State of the Art and Research Challenges. Computer, 40(11):38-45, November 2007. DOI: 10.1109/MC.2007.400

[19] Berbecaru D, Lioy A. Towards Simplifying PKI Implementation: Client-Server based Validation of Public Key Certificates. In Proceedings of the 2nd IEEE International Symposium on Signal Processing and Information Technology (ISSPIT'02), pp. 277-282, December 2002.

[20] Halappanavar M, Mukkamala R. ECPV: Efficient Certificate Path Validation in Public-key Infrastructure. In Proceedings of the 17th Annual Working Conference on Data and Application Security (DBSec'03), pp. 215-228, 2003.

[21] Rivest RL, Shamir A. PayWord and MicroMint: Two Simple Micropayment Schemes. Proceedings of the International Workshop on Security Protocols, Lecture Notes in Computer Science, pp. 69-87, 1997.

[22] Council of the European Union. Council Directive 2001/115/EC of 20 December 2001 amending Directive 77/388/EEC. 2001.

[23] European Telecommunications Standards Institute (ETSI). Electronic Signatures and Infrastructures (ESI); Registered Electronic Mail (REM). In ETSI Technical Specification (TS) 102 640. October 2008.

[24] ETSI. Electronic Signatures and Infrastructures; Profiles of CMS Advanced Electronic Signatures based on TS 101 733 (CAdES). In ETSI TS 102 734. February 2007.

[25] ETSI. Electronic Signatures and Infrastructures; Profiles of XML Advanced Electronic Signatures based on TS 101 903 (XAdES). In ETSI TS 102 904. February 2007.

[26] Pinkas D, Housley R. Delegated Path Validation and Delegated Path Discovery Protocol Requirements. IETF RFC 3379, September 2002.

[27] Hesse PM, Lemire DP. Managing Interoperability in Non-Hierarchical Public Key Infrastructures. Proceedings of Network and Distributed System Security Symposium (NDSS'02), February 2002.

[28] Department of Information Security and Electronic Signature, Slovakian National Security Authority. Certificate Path Validation v1.4. No. 1891/2006/IBEP-011, November 2006.

[29] Kim J, Kim S, Moon K. Design of Integration Security System using XML Security. In Proceedings of World Academy of Science, Engineering and Technology (WASET'05), Vol. 8, pp. 136-140, October 2005.

[30] Blaič AJ, Klobučar T, Jerman BD. Long-term Trusted Preservation Service using Service Interaction Protocol and Evidence Records. Computer Standard & Interfaces, 29(3):398-412, March 2007. DOI: 10.1016/j.csi.2006.06.004

[31] Gondrom T, Brandner R, Pordesch U. Evidence Record Syntax (ERS). IETF RFC 4998, August 2007.

[32] Jerman Balzic A, Sylvester P, Wallace C. Long-term Archive Protocol (LTAP). IETF Internet-Draft version 08. July 2009.

[33] OASIS. Digital Signature Service Core Protocols, Elements, and Bindings Version 1.0. OASIS Standard. April 2007.

[34] University of Murcia. SUMA Campus Virtual. https://suma.um.es/suma/sumav2 [6 August 2009]

[35] Cantor S, Kemp J, Philpott R, Maler E.Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. OASIS Standard, March 2005.

[36] Adams C, Cain P, Pinkas D, Zuccherato R. Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP). IETF RFC 3161, August 2001.

[37] OpenXKMS, Open Source Implementation of XKMS 2.0, http://xkms.sourceforge.net [6 August 2009]

[38] UMU-PKIv6 SCVP API. Public Key Infrastructure with IPv6 support. University of Murcia. http://pki.inf.um.es/SCVP/ [6 August 2009]

[39] XML Trust Services - XKMS, VeriSign Inc. http://www.verisign.com/developer/xml/xkms.html [6 August 2009]

[40] VeriSign's XKMS Web service, http://interop-xkms.verisign.com/xkms/Acceptor.nano [6 August 2009]

[41] Dillaway B. Implementing XML Key Management Services Using ASP.NET. ASP.NET Technical Articles, Microsoft Corporation, January 2002.

[42] SQLData Systems, Inc., http://www.sqldata.com [6 August 2009]

[43] DataPower Technology Corporation, http://www.dptia.com [6 August 2009]

[44] TrustFinder XKMS Server, http://www.ascertia.com [6 August 2009]

[45] Alcaraz Calero JM, López Millán G, Martínez Pérez G, Gómez Skarmeta A.F. Towards the Homogeneous Access and Use of PKI Solutions: Design and Implementation of a WS-XKMS Server. Journal of Systems Architecture, 55(4):289-297, April 2009. DOI: 10.1016/j.sysarc.2008.10.004

[46] XKMS Prototype Server, Wings of Hermes, http://www.wingsofhermes.org/xkms.html [6 August 2009]