XML Security: The XML Key Management Specification

XKMS helps make security manageable

Manish Verma

January 27, 2004

The XML Key Management Specification (XKMS) outlines an easy mechanism for accessing and integrating with Public Key Infrastructure (PKI). In this article, Manish Verma explains the objective behind XKMS and then offers a step-by-step guide to using the XKMS service to register and retrieve information related to a public and/or private key.

View more content in this series

With an ever-increasing number of people and businesses relying on the Internet to exchange confidential and sensitive information, security has become a hot issue. Two security-related topics have gained significant importance:

- **Ease of management:** Making the security infrastructure's usage and integration with applications easy so that its adoption becomes widespread.
- **Portable trust:** After a trust relationship has been established with an entity, having a standard mechanism to transfer that trust to another cooperating entity. Single SignOn is a typical example of portable trust. After a user has been authenticated with a particular Web site, a standard mechanism passes that information to other cooperating sites that require the user's authentication information. This allows those sites to transparently share information about an entity without the need to request the same information from the entity again and again. For single sign-on to work, the entities must recognize each other's credentials.

The XML Key Management Specification (XKMS) allows for easy management of the security infrastructure, while the Security Assertion Markup Language (SAML) makes trust portable. SAML provides a mechanism for transferring assertions about authentication of entities between various cooperating entities without forcing them to lose ownership of the information.

This article discusses the role that XKMS plays in managing the security infrastructure, and provides a step-by-step guide to using XKMS.

The problem of manageability

XML encryption and digital signatures rely on Public Key Infrastructure (PKI) to help encrypt, decrypt, sign, and verify various documents. For an application to use XML encryption and/or
digital signature, the application must use or integrate with a PKI solution. Various PKI solutions are available, such as X.509 (the most widely used), Pretty Good Privacy (PGP), Simple Public Key Infrastructure (SPKI), and Public Key Infrastructure X.509 (PKIX). The question is, which is the right one to use or integrate with?

Unfortunately, the answer is not as simple as choosing one of the many available PKIs and integrating with it. The real issue is how many PKIs -- along with their syntax and semantics -- must the application be aware of for it to be able to talk to other applications that may send or expect documents encoded or signed using other available PKI solutions. For example, if organization A uses an X.509 PKI solution and sends encrypted documents to organization B, which uses an SPKI PKI solution, then organization B won't be able to decrypt and use the document sent by A. For A and B to work together, one of them has to understand the other's PKI solution. If you extrapolate this scenario to a situation where multiple partners are involved, it becomes clear that all of the partners will have to be aware of each other's PKI solution, thus increasing each application's complexity many times.

In rest of this article, I will demonstrate how XKMS provides an answer to this problem.

**XML Key Management Specification (XKMS)**

XKMS allows easy management of the PKI by abstracting the complexity of managing the PKI from client applications to a trusted third party. The trusted third party hosts the XKMS service while providing a PKI interface to the client applications.

**XKMS objectives**

The primary objectives of XKMS are:

- Create an abstract layer between the application and the PKI solution. This allows the application to plug in different PKI solutions based on the need, without requiring any modification of the application itself.
- Eliminate the need for the application to understand complex PKI syntax and semantics by providing a simple XML-based protocol for processing key information through the XKMS service.
- Move complexity from the client application to the infrastructure level, thereby allowing the application to remain simpler and smaller. This allows even small footprint devices to take advantage of PKI.
- Implement XKMS such that it is platform-, vendor-, and transport protocol-neutral.

**XKMS overview**

XKMS is implemented as a Web service that allows a client application to access PKI features, thereby reducing the client application's complexity. The client application need not be concerned about the syntax of the underlying PKI, which could be any of the following:

- X.509 (the most widely used)
- Pretty Good Privacy (PGP)
• Simple Public Key Infrastructure (SPKI)
• Public Key Infrastructure X.509 (PKIX)

XKMS specifications are made up of two specs, one that relates to registration of the public keys -- XML Key Registration Service Specification (XKRSS) -- and one that's concerned with the retrieval of information based on key information -- XML Key Information Service Specification (XKISS).

Another specification, X-Bulk, has been released by the Worldwide Web Consortium (W3C); it addresses the issue of registering key pairs in bulk. I will explain X-Bulk after discussing XKRSS.

**XML Key Registration Service Specification (XKRSS)**

This part of the XKMS deals with the mechanism for registering a key pair with a service provider. You can register keys with an XKMS service in two ways:

• The client generates a key pair and provides the public key, along with other information, to the service provider for registration.
• The XKMS service generates a key pair for the client, registers the public key of the pair with itself, and sends the private key of the pair to the client for its use. The client can also tell the XKMS service to keep the private key as well. The private key is kept with the XKMS service in case the client loses its private key.

An XKRSS service specification defines four operations:

• **Register:** Information is bound to a key pair through key binding. During registration, either the client provides the public key, along with some proof of possession of the corresponding private key, or the service provider generates the key pair for the client. The service provider may request more information from the client before it registers the public key (and optionally the private key as well).
• **Reissue:** A previously registered key binding is reissued. New credentials in the underlying PKI are generated using this operation. While there is no lifespan for the key binding information used by XKMS, the credentials issued by the underlying PKI occasionally do have a time span that must be renewed periodically.
• **Revoke:** This operation allows clients to destroy the data objects to which a key is bound. For example, an X.509 certificate that's bound to an XKMS key is destroyed when this operation is called.
• **Recover:** This operation allows clients to recover the private key. For this operation to be meaningful, the private key must have been registered with the service provider. One of the ways in which the service provider may have the private key is when the key pair is generated at the server rather than the client. The recover operation adds a lot of value in cases where the private key of the pair is used to encrypt data; if the private key were lost, the encrypted data would also be lost. When this happens, it makes sense to have the private key generated at the server and have the service provider keep the private key. That way the private key can be recovered.

If the private key that's lost is only used for digitally signing the documents, a new key may be generated without affecting the validity of the existing signed documents. Hence, keys used
for such a purpose can very safely be generated by the client, and the private key may never
be registered with the XKMS service.

An XKMS service implementing XKRSS service specifications may choose to offer some, all, or
none of these operations. The XKRSS service specification does not make it mandatory for the
XKMS service to implement any of the operations.

**Bulk registration of key pairs (X-Bulk)**

X-Bulk handles registration of multiple key pairs in one request message, unlike XKRSS which
addresses registration of one key pair at a time. The X-Bulk specification defines a batch element
that can have multiple registration requests or multiple server responses, all of which follow the
XKMS message format. Like XKRSS, the X-Bulk service supports all four operations -- Register,
Reissue, Revoke, and Recover. X-Bulk can handle bulk registrations of client- as well as server-
generated key pairs.

**XML Key Information Service Specification (XKISS)**

This part of XKMS addresses the mechanism that allows client applications to authenticate
encrypted/signed data.

The client authenticates the encrypted/signed data by passing the corresponding key information
to the service provider. The service provider then responds with "true" or "false." "True" indicates
that the public key corresponding to the private key used for signing does belong to the entity that
claims to have signed the docs.

The XKISS service specification defines the following two operations:

- **Locate:** Locate resolves a `<ds:KeyInfo>` element that may be associated with XML
  encryption or XML signature, but it does not prove the validity of data binding in the
  `<ds:KeyInfo>` element.
- **Validate:** This operation does all that locate does, plus more. The locate service finds a
  key based on the `<ds:KeyInfo>` element, but does not assure the trustworthiness of the key
  binding information. The validate operation not only searches the public key corresponding to
  the `<ds:KeyInfo>` element, but also assures that the key binding information that it returns is
  trustworthy.

**The XKMS process**

In this section I will take you through the step-by-step process of registering and retrieving the
keys.

The registration process that's based on XKRSS allows registration of both client-generated and
server-generated keys. In client-generated key registration, the client generates the keys and then
requests that the XKMS service register the public key of the pair by sending the public key and
providing the private key as proof of possession. In server-generated key registration, the XKMS
service generates both the private and public keys, then registers the public key and sends the
private key to the client.
Listing 1 demonstrates how a client-generated key is registered. In order to use the code in the listing, you will have to contact some XKMS service provider that has a demo or test XKMS service running. Obtain from the service provider the information that you will need to register keys with their demo or test XKMS service. I use Verisign as my XKMS provider and use the shared secret and the key name that they provided me for registering my key.

Listing 1. XKMS key registrations

```java
String sharedSecret = "passcode";
String passPhrase = "passPhrase";
KeyPair rsakey = getKeyPair();
String keyName = "http://xkms.verisign.com/keyname?jurisdiction=d7ea68c518b2602ca4bbca895826a7dd&" + "mail_email=mverma2@xkms.org&corp_company=Nano";
XKMSKeyData data = new XKMSKeyData(rsakey, new XKMSKeyName(keyName));
XKMSAuthInfo authInfo = new XKMSAuthInfo(passPhrase, sharedSecret);
XKMSRegister register = new XKMSRegister(data, authInfo);
XKMSRegisterResponse resp = null;
String serviceurl = "http://interop-xkms.verisign.com/xkms/Acceptor.nano";
URL url = new URL(serviceurl);
XmlTransportSOAP transport = new XmlTransportSOAP(url);
resp = register.send(transport);
System.out.println("Response status is " + resp.getStatus());
if (resp.getStatus()) {
    System.out.println("The key name is " + resp.getKeyName());
    System.out.println("The public key is " + resp.getPublicKey());
}
```

**Step 1:** Get the passcode (shared secret) from the service provider where the key pair is to be registered. The service provider may also give the key name to be used with the passcode.

**Step 2:** Generate the key pair at the client using `getKeyPair()`. The code for `getKeyPair()` is part of the source code that you can download with this article (see Resources).

**Step 3:** Specify the key data required in XKMS key registration. Create the `XKMSKeyData` object. Provide the key pair and the key name as the input. Of the key pair, the public key is registered and the private key is used as proof of possession. The key name used in `XKMSKeyData` has to follow a particular format specified by the provider.

**Step 4:** Specify the registration authorization data required for XKMS key registrations. There are two ways to provide the authorization information:

- **Shared secret:** The registration request is accompanied by the shared secret that is given by the registering authority to the client out-of-band.
- **Registration authority (RA) authentication:** The X.509 certificate of the RA and private key of the RA are used. In Listing 1, I demonstrate the use of the shared secret authentication mechanism.

**Step 5:** Create an instance of the `XKMSRegister` object by providing the XKMS key data and authorization information.

**Step 6:** Create an `XmlTransportSOAP` object. Provide the URL as input where the registration service is running.
**Step 7:** Send the request to the **XKMSRegister** object using the `send` method to which the **XmlTransportSoap** instance is passed.

The send method registers your key and returns a **XKMSRegisterResponse** object. This object can be queried to get the key name and other key information. The key name that you get on querying the **XKMSRegisterResponse** object is different from the key name that you gave at the time of creating the **XKMSKeyData** object in step 3. Later, you will use this key name to locate your key.

Next, I will demonstrate how to locate a key that has been registered with the service provider. (The mechanism demonstrated in Listing 2 for locating a key uses the key name; I will explain that in more detail after the listing.)

**Listing 2. Locating the key**

```java
// The key name that you got from the XKMSRegisterResponse
// Object
String keyNameToLocate =
    "http://xkms.verisign.com/key?company=VeriSign&department=XKMS Test&"+
    "CN=MANISH VERMA6&issuer_serial=73efeeea87e159e6d7fc70ef01c6fe?";
// An array of response strings to indicate what key
// information is requested
String[] xkmsResponse =
    {XKMSLocate.KeyName,XKMSLocate.KeyValue,XKMSLocate.X509Cert,XKMSLocate.X509Chain};
```

**Step 1:** To locate a key, you can use any of the following information for querying:

- Key name (This should be the key name previously returned in an **XKMSRegisterResponse** object. The key name used with **XKMSKeyData** during registration may not be used here.)
- Public key value
- X.509 certificate

In Listing 2, I use the key name to query the key.

**Step 2:** Specify an array of response strings whose values are of interest.

**Step 3:** Create the **XKMSLocate** object with the key name and the array of response strings.

**Step 4:** Create an **XmlTransportSOAP** object. Provide the URL as input where the registration service is running.
Step 5: Send the request to the \texttt{XKMSLocate} object using the \texttt{sendRequest} method to which the \texttt{XmlTransportSoap} instance is passed.

Step 6: Get the \texttt{XKMSLocateResponse} object. From this object, you can obtain a list of \texttt{XKMSKeyInfo} objects. These objects are iterated to get the information corresponding to the response strings.

Security of the XKMS service itself

You are not done with XKMS yet. You should be aware of few things before you entrust the security of your documents and applications to the XKMS service. The issues that I touch upon in this section relate to any Web-based service, but it's important to ensure that XKMS is reliable and available.

- **Prevent replay attacks**: Ensure that the reply that XKMS service sends for a request is not captured by an unauthorized third party. This captured response can again be sent by the third party as a response to some other request and masquerade as the XKMS service itself. You can deal with such a situation in various standard ways, such as timestamping the responses.

- **Prevent denial of service attacks**: Using XKMS to get key information is useful if it is available when needed and within a reasonable amount of time. The service must be implemented such that it can withstand denial of service attacks and hence remain highly available.

- **Verify a sound, robust key recovery policy**: Make sure that the XKMS service provider has a sound and robust key recovery policy as it has a direct impact on signed document repudiation and encrypted document authenticity.

Conclusion

Often, good technologies fall by the wayside because they are cumbersome to use; in such cases, only a small devoted set of developers continue using the technology without it ever being adopted by average IT departments. PKI has been around for many years, but has not yet made it into typical IT departments. Now, XKMS provides an easy mechanism for using and integrating PKI with applications. In this article, I have explained the objectives of having an XKMS abstraction layer on various PKI solutions, and demonstrated how easy it is to use the XKMS service for registering and locating your key. In my next article, I will focus on explaining how to make this trust portable using SAML.
## Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x-seclay3_XKMSCode.zip</td>
<td>1MB</td>
</tr>
</tbody>
</table>

© Copyright IBM Corporation 2004

**Trademarks**
(www.ibm.com/developerworks.ibm/trademarks/)