Using the XML Key Management Specification (and breaking X.509 rules as you go)

Stephen Farrell[1] and José Kahan[2]

[1] Distributed Systems Group,
Department of Computer Science,
Trinity College, Dublin 2, Ireland,
stephen.farrell@cs.tcd.ie,
https://www.cs.tcd.ie/Stephen.Farrell/
[2] W3C / ERCIM,
INRIA Rhône-Alpes,
ZIRST, 655 av. de l'Europe, Montbonnot,
FR-38334 ST ISMIER CEDEX, France,
jose.kahan@w3.org,
http://www.w3.org/People/Jose/

Abstract. Implementing X.509 based public-key infrastructure requires following a complex set of rules to establish if a public key certificate is valid. The XML Key Management Specification has been developed as one way in which the implementation burden can be reduced by moving some of this complexity from clients and onto a server. In this paper we give a brief overview of the XML key management specification standard, and describe how, in addition to the above, this system also provides us with the means to sensibly break many of the rules specified for X.509 based public key infrastructure.

1 Introduction

In this paper we will describe how the XML Key Management Specification (XKMS) [1, 2] can be used as a kind of intranet "front-end" to an X.509 based Public Key Infrastructure (PKI). Such PKIs mainly try to follow the rules specified in most detail in RFC 3280 [3].

Once we have seen how an XKMS responder can be used in such a situation we will then examine a number of ways in which the responder can offer better service to clients, by breaking the rules of X.509!

2 The XML Key Management Specification

The XML Key Management Specification (XKMS) [1] is a W3C Recommendation designed to ease the costs of PKI deployment without sacrificing its benefits. It is suitable for use in conjunction with the XML-Signature [4] and XML-Encryption [5] W3C Recommendations as well as in other application contexts,

such as email. As stated, XKMS is a W3C Recommendation, the pinnacle of the W3C standards process: it has been reviewed by W3C Members and other interested parties and there exists enough implementation and interoperability proof to validate its concepts. The XKMS W3C Recommendation was published on 28 June 2005 though the original W3C Note on XKMS dates from 2001.

We will now give a brief overview of XKMS - a companion paper [6] describes XKMS in more detail and also reports on interoperability status. XKMS consists of two different parts: the XML Key Information Service Specification (X-KISS) and the XML Key Registration Service Specification (X-KRSS). X-KISS defines a protocol to support the delegation by an application to a service of the detailed processing of key information associated with an XML-Signature, XML-Encryption, or other usage of the XML-Signature `<ds:KeyInfo>` element. X-KRSS defines a protocol for the registration of a public key by a key pair holder, with the intent that the key subsequently be usable in conjunction with X-KISS or a PKI. XKMS is designed to be protocol independent and it proposes bindings [2] over SOAP/1.2 as well as plain HTTP.

2.1 X-KISS

Reducing the complexity of applications using XML-Signature is one of the key objectives of the protocol design. X-KISS clients are relieved of the complexity of the underlying PKI used to establish trust relationships. These relationships may be based upon a different specification, such as X.509/PKIX, or PGP[7].

In addition, sometimes the information provided by a signer can be insufficient for performing cryptographic verification or to be able to decide whether to trust a signature. Alternatively, the information provided by the signer may be in a format that is not supported by the client. In these cases communication with an X-KISS service can be useful as a way to get that "missing" information.

Examples where the key information could be insufficient for the client include:

- The key may be specified by a name only.
- The key may be encoded in an X.509 certificate that the client cannot parse.
- In the case of an encryption operation, the client may not know the public key of the recipient (e.g., just having a name).

X-KISS works via two different services: `Locate` and `Validate`.

`Locate` resolves a `<ds:Keyinfo>` element but does not require the service to make an assertion concerning the validity of the data in the `<ds:Keyinfo>` element. `Validate` does all that `Locate` does, but in addition, the client obtains an assertion (at that time, according to that responder) specifying the status of the binding between the public key and other data, for example a name or a set of extended attributes. Furthermore the service represents that each of the other data elements returned are bound to the same public key.

2.2 X-KRSS

X-KRSS handles the registration and subsequent management of public key information. An X-KRSS service may bind information such as a name, an identifier or other attributes, to a public key, on reception of a client request. The key may be generated by the client or by the service on request. The Registration protocol may also be used for subsequent management operations including recovery of the private key and reissue or revocation of the key binding. The protocol provides ways of authenticating the requester and the possession of a private key. Additionally it provides a means of communicating the private key to the client in the case that the private key is generated by the registration service.

The operations constituting X-KRSS are:

- `Register`: Information is bound to a public key through a key binding. Generation of the key pair may be performed by either the client or the Registration service.
- `Reissue`: A previously registered key binding is updated. It is similar to the initial registration of a key and the principal reason a client would make a Reissue request is to cause the registration service to generate new credentials in the underlying PKI, e.g., X.509 Certificates.
- `Revoke`: A previously registered key binding may be revoked. A revocation request need only contain sufficient information to identify the key binding to be revoked and the authority for the revocation request.
- `Recover`: The private key associated with a key binding is recovered. The private key must have been previously escrowed with the recovery service, for example by means of the X-KRSS registration of a server generated key.

2.3 Using XKMS as a PKI front-end

PKIs aim to allow every user and every application to verify the identity of everyone with which they communicate and to ensure that the counter-party identity is appropriate for the transaction and also that the identity/key binding is still valid (not revoked). Unfortunately, the infrastructure needed to support this places such burdensome demands on application developers that it can be difficult to develop a secure application that achieves all of these goals simultaneously.

In order to verify a given signed document, a party must locate the corresponding public-key certificate, verify its validity, and parse it to extract the corresponding public-key. Traditionally with PKI, these operations are carried out by a (PKI) client application. This requires complex configuration settings, e.g., rules/configuration for mapping application identities to X.500 names. Moreover, as different PKIs can have different conventions, this can complicate the integration of PKI with applications.

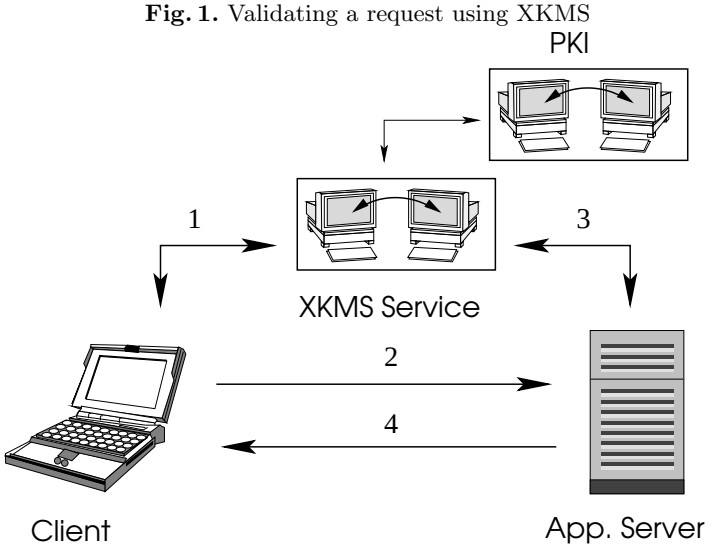
In XKMS, these trust decisions are delegated to a common server, so that they can be centralized and applied consistently across platforms. The only configuration information that an XKMS client needs is the URL of the server and

the public key the server will be using to sign its replies. Different trust models can be supported by using different server URLs.

Figure 1 gives an example of one way in which XKMS can validate requests between a client and an application server (we assume that both the client and application server have previously used X-KRSS to register their public keys in the XKMS service). The workflow is as follows:

1. The client uses the `Validate` request to get a public key for the server.
2. The client sends a signed request to the application server; the request includes a copy of the client's public-key certificate and is encrypted with the server's public key.
3. The application server decrypts the request, and forwards the client's public key certificate to the XKMS service using a `Validate` request, asking to validate it and to extract the public-key contained within.
4. The application server processes the client request.

Fig. 1. Validating a request using XKMS



Note that in the preceding example, the XKMS service does not sign or verify the signature of the client's request. These operations have to be done locally. Also note that the PKI that is behind the XKMS responder is completely transparent to the client - it could be based on X.509 or something else. Likewise, this transparency allows for the exchange of one PKI with another one with minimal change for the client and application server code. Finally, the client and the application server could contact different XKMS services; in this case, one XKMS server could act as client to get the required information from the other server.

In addition to acting as a relatively straightforward "front-end" for the usual X.509 based PKI, XKMS is also intended to be usable in other contexts, in particular where PGP [7] based data formats are used instead of X.509 based ones or where a PKI is built from scratch based on "native" XML formats like the `<ds:KeyInfo>` structure defined in XML-Signature [4].

3 Breaking X.509 rules using XKMS

in this paper we are interested in considering how an XKMS responder, placed in front of an X.509 PKI can usefully break the rules of that PKI in order to provide better service to the responder's clients. We first consider the types of rule which can be broken, and then give a number of examples where breaking each type of rule is of benefit.

Many of the rule-breaks, or "cheats", we outline below might be considered to be features of a PKI. For example, if an XKMS responder uses a different name for an entity, someone could respond that the PKI could have done the same thing via the use of another SubjectAltName extension. However, even if some PKI could achieve the same effect, the point is that for a given deployed PKI, changing the name of the entity is cheating, at least in PKI terms. Whether this indicates that X.509 based PKIs are too rigid, is a topic potentially worth discussing though not one we address here.

Note that many, but not all, of the "cheats" here could also be implemented by a *non-conformant* implementation of the SCVP protocol [8], however a conforming XKMS implementation can do all of these things, as well as acting as a more "traditional" PKI front-end when circumstances warrant.

3.1 A classification of the rules-to-be-broken

RFC 3280 [3] describes in detail the contents of certificates and related data structures and (mainly in Section 6) describes an algorithm which can be implemented in order to check the validity of certificate paths. The rules which are explicitly and/or implicitly specified therein could be broken down into the following classes:

- **Certificate content**. Rules as to how to interpret the content of a certificate.
- **Certificate status checking**. Rules stating whether or not a previously issued certificate remains valid.
- **Valid path constraints**. Rules which valid certificate paths follow (many of these are derived from the validation algorithm).

For each of these classes we will give examples of how breaking related rules can be useful. Note that the classes themselves are not really significant, but they do help to organize our presentation and also help us to find additional ways in which we can break X.509 rules!

3.2 Certificate content related rules

We begin by breaking the most basic rules of X.509 based PKI. We will go into more detail on the first rule-break, for later ones we leave the details of the XKMS messaging required as an exercise for the reader.

Changing the name of the certificate holder . In many cases organizations will have gone to much trouble to ensure that the certificate holder (subject and/or subjectAlternateName) fields contain unique values, for example, via the inclusion of employee numbers or other serial numbers. Non-human entities (like applications) may have similarly constructed names for consistency. This makes these names cumbersome and unsuitable for many applications, for numerous reasons, not least the fact perhaps the applications didn't actually exist when the PKI naming debate was raging within the enterprise. An XKMS responder can therefore usefully maintain its own translation of names, perhaps via a set of tables and/or some algorithm. In this way, an entity known to the PKI as `"L=Internet;O=Example Org; CN=Joseph User+SN=123456798"` might sensible be mapped to `"joe"`. In terms of XKMS processes, the steps that occur could be as follows:

1. Application received XML-Signature produced by Joe, with Joe's X.509 certificate in a `<ds:KeyInfo>`.
2. Application does a Validate containing that `<ds:KeyInfo>`, but without ever looking "inside".
3. Responder does PKI things to validate the certificate and then maps from the subject field (above) and the name of the application (or other context) to `"joe"`.
4. Responder returns a binding containing the appropriate ds:keyValue and a `<ds:KeyName>` containing simply the string `"joe"`.
5. Application correctly accepts that XML-Signature is from Joe and does further XML processing on the relevant documents.

In a variation on the above, the responder could simply *invent a pseudonym* for the certificate holder each time it sees a new XKMS-client/application pair (or following many other algorithms). This would make the application less likely to cause privacy problems since application state would contain the pseudonym and not a real identity. It would also make it harder to correlate the same user's actions over multiple applications.

Another related "cheat" would be where the responder uses some *name resolution* service (like DNS), and maps from the requested name to one present in a certificate (or vice-versa). This could be useful to handle load-balancing and other cases where the names currently in X.509 certificates don't match those that the application requires. Of course, this means putting some "trust" in whatever resolver is used by the XKMS responder, but that may well be as secure, and cheaper than, frequently getting new X.509 certificates.

Hidden key escrow. In response to a query from an application which is about to do an encryption operation (e.g. an S/MIME enabled mail user agent), an XKMS responder could produce a response which contains a public key for which the corresponding private key is known to some other application (e.g. an outbound SMTP server). In the example, the mail client will (unknown to it!) encrypt for the SMTP server, which can decrypt, presumably then apply some useful policy checks, and subsequently re- encrypt for the intended recipient. Say that now the SMTP server uses the XKMS responder to find the intended recipient's key. Either via configuration, or even some visible and/or hidden values in the KeyBinding returned (e.g. a proprietary X.509 extension), to the mail client, the XKMS responder can know how to respond.

Ignoring expiration. X.509 certificates expire, unfortunately. For many applications, there is no real need to update certificates other than to handle the fact that certificates expire. XKMS can rescue us here by simply checking the notBefore and notAfter values in a certificate and running the RFC 3280 validation algorithm for some time in that interval, that is, the responder can ignore the certificate's validity period entirely. In this way, Joe can go on using the same certificate indefinitely possibly saving money for the organization (if certificate renewal is a chargeable service). For any application which has its own concept of account revocation or expiry this mode of operation is entirely sufficient (i.e. the notAfter field of the X.509 certificate adds no value in such cases).

Creating an entirely new certificate. In response to a locate or validate request a responder could do the standard PKI operations to validate some certificate for the public key in question, but then create a new certificate, issued by the responder (or at least signed using a key under the responder's control) which will be more likely to satisfy the application requirements. Basically, this allows the responder to include any of the above "cheats" in an X.509 certificate, which is useful if the ultimate X.509 relying party will trust the responder-issued certificate. This is slightly different from normal PKI models, in that the responder may for example change the subject field to better match the application requirements.

This "cheat" also allows the responder to *handle potential cryptographic weaknesses*, for example related to hash-function robustness. If the responder has a local repository of certificates, then it can perhaps be confident that those certificates retain their integrity even if the hash functions used therein are currently considered weak. The newly created certificate can then use better hash functions or else countermeasures like long, random serial numbers in order to create a certificate which will be acceptable to the X.509 relying-party.

If the responder is creating new certificates for some application where the certificate is only used for a short period, then the responder could *re-use serial numbers* in order to make it simpler to handle revocation of such certificates. For example, if the XKMS client application uses the certificate for a CMS [9] based confidentiality service, but throws away the certificate shortly after use, then the responder could use a small set of serial numbers for all certificates. The

net effect is that CRLs never get long enough to cause problems, while relying party applications can do standard revocation checks. This "cheat" shows that even the most fundamental "rule" of X.509 (that an issuer MUST NOT re-use serial numbers) can usefully be broken!

Covert channel. Where a responder creates new certificates there are many X.509 fields which can be used as a kind of covert channel between the XKMS responder and intermediaries who see the certificate or perhaps the private key holder (if the certificate is carried end-to-end in an encrypting application). Such a covert channel could be used to carry (or link to) authorization information or any other application data. The fields that could be used in this way without affecting the use of the public key contained in the certificate include the serial number, unique identifiers, issuer, subject, algorithm identifiers (both inner and outer), and many of the standard extensions. The version number could also potentially be used, though probably at the expense of suffering decoding errors in intermediaries. Note that if CMS is used for encryption then the serial number covert channel gets through to the private key holder even if the certificate does not. Although a similar trick can be done using short-lived X.509 certificates, this "cheat" differs in two respects: the certificates are likely to be even more short lived since an XKMS client will more frequently contact its responder, hence the covert channel has higher bandwidth. Secondly, the XKMS protocol inherently supports us in doing this, whereas in an X.509 context a modified LDAP server might be the best way to implement this.

3.3 Certificate status related rules

Handing revocation of certificates has always been a really problematic area for X.509 based PKIs. In this section we will examine ways in which we can help by breaking the rules of X.509.

Entirely ignoring revocation. The most obvious way to solve the revocation problem is to get rid of it. This is straightforward enough - the responder simply runs the validation algorithm but skips the certificate status checking stages entirely. It may be useful to for the responder to keep its own, application-specific blacklist as well as a global blacklist and to use that as the basis on which it decides the Status of KeyBindings.

Ignoring business motivated revocations. If an enterprise pays a service provider to operate a PKI on their behalf then that service provider might create a CA for that purpose. When the customer no longer wants to use that service, the service provider might revoke that CA's certificate, thus potentially invalidating all of the end entity certificates issued by that CA. An XKMS responder is in a useful position to ignore this revocation in order to provide business continuity where the customer is switching from one service provider to another, or to an in-house PKI. More generally, one could perhaps argue that leveraging the flexibility of XKMS makes many business transitions easier, when compared with X.509 compliant solutions.

Ignoring authority revocation. Access to the required CRLs or OCSP responders for end entity certificates may be possible in many circumstances. However, for some PKIs it will be hard to find the relevant CRLs or OCSP responders in order to ensure that no CA on the path has been revoked. An XKMS responder could usefully maintain its own list of revoked CAs and therefore not have to attempt to access these ARLs or equivalent.

Ignoring revocation timing. An XKMS responder could be configured so as to ignore revocation timing information (e.g. the next update field etc.) and could simply periodically access certificate status information, presumably from a fairly reliable source. This amounts to having the XKMS responder (administrator) choose the level of acceptable risk, in terms of potential missed revocations. In many circumstances the XKMS responder will be in a much better position than the X.509 certificate issuer to properly evaluate this risk - thus leading to a more easily deployed system.

3.4 Path validation related rules

X.509 based PKIs define a range of certificate extensions which are aimed at controlling which paths are valid and which are not. There are many cases where we may want to disagree with the constraints that CA's would like to impose and XKMS allows this to be done easily.

Ignoring all certificate policy checks. Certificate policies, even if initially sensible, may well not last as long as certificates, for example introducing new applications into a bridge-CA structure may make many issued certificates useless. An XKMS responder could therefore effectively "strip" out all certificate policy handling, and in particular all policy mapping.

Ignoring basic constraints. A responder could totally ignore the basic constraints extension (or its path length constraint) and treat an end entity as a CA. This could be useful to integrate with some quasi-standard or less frequently seen PKIs such as those used in some grid computing environments [10].

4 Conclusions

We have given a brief outline of XKMS and one of its main use cases: use as a locally trusted intranet server. However, by showing ways in which XKMS can usefully be used to break restrictive X.509 rules, we have also demonstrated that XKMS can be used as more than just a front-end for X.509 based (or other) PKIs.

It should also be clear that we could have given many more examples - in fact one could possibly construct an interesting "cheat" for almost all certificate and CRL fields, and for almost all MUST or SHOULD statements in RFC 3280 [3]. However, we are, of course, not recommending that XKMS implementers

implement these “cheats” - they each only make sense in specific contexts. We do expect that over time, some of them (or others) will be found to be useful enough to warrant inclusion in XKMS implementations - perhaps at some stage some might even find their way back into the X.509 related standards!

Finally, the fact that there are so many potentially useful ways in which the X.509 PKI rules can be broken, given the opportunity offered by XKMS, may indicate that those rules are somewhat too onerous, at least when an on-line trusted server is available as is the case with XKMS (and, whenever it is finally completed, with SCVP).

References

1. P. Hallam-Baker and S. H. Mysore (eds.): XML Key Management Specification (XKMS 2.0). Recommendation, W3C (2005) http://www.w3.org/TR/2005/REC-xkms2-20050628/.
2. P. Hallam-Baker and S. H. Mysore (eds.): XML Key Management Specification (XKMS 2.0) Bindings. Recommendation, W3C (2005) http://www.w3.org/TR/2005/REC-xkms2-bindings-20050628/.
3. R. Housley, W. Polk, W. Ford, D. Solo: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC, IETF (2002) http://www.ietf.org/rfc/rfc3280.txt.
4. D. Eastlake et al (eds.): XML-Signature Syntax and Processing. Recommendation, W3C (2002) http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/.
5. D. Eastlake and J. Reagle (eds.): XML Encryption Syntax and Processing. Recommendation, W3C (2002) http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/.
6. G. Alvaro, S. Farrell, T. Lindberg, R. Lockhart, Y. Zhang: XKMS Working Group Interoperability Status Report. In: to appear in Proceedings of EuroPKI 2005, Univerrsity of Kent, Canterbury, England. (2005) http://www.europki.org/.
7. P. Zimmermann: Pretty Good Privacy (PGP), PGP User’s Guide. Technical report, MIT (1994)
8. T. Freeman, R. Housley, A. Malpani, D. Cooper, T. Polk: Simple Certificate Validation Protocol (SCVP). Internet Draft, IETF (2005) http://www.ietf.org/internet-drafts/draft-ietf-pkix-scvp-18.txt.
9. R. Housley: Cryptographic Message Syntax. RFC, IETF (2004) http://www.ietf.org/rfc/rfc3852.txt.
10. S. Tuecke, V. Welch, D. Engert, L. Pearlman, M. Thompson: Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile. RFC, IETF (2004) http://www.ietf.org/rfc/rfc3820.txt.