FFI-rapport 2008/00278

XML Based Certificate Management

Anders Fongen

Norwegian Defence Research Establishment (FFI)

21st February 2008

FFI-rapport 2008/00278

1086

ISBN 978-82-464-1346-4

Keywords

XML

Informasjonssikkerhet

Kryptografi

Electronisk Signatur

Offentlige nøkler

Sertifikathåndtering

Approved by

Anders Eggen	Project manager
Vidar S. Andersen	Director

English summary

The need for key management in SOA-based systems is addressed in this report. The report starts with a presentation of key management principles and a discussion of the problem posed by the PKI (Public Key Infrastructure) technology. An XML-based specification for key management over a Web Service protocol, XKMS, will be discussed in depth. The report further presents an XKMS interoperability experiment conducted by the World Wide Web Consortium in order to test the precision of the specification.

Keymanagement in mobile networks is a particularly complex problem since the absence of reliable network connectivity inhibits the certificate validation from relying on on-line resources. The report outlines possible solution for key management in mobile network.

Sammendrag

Behovet for administrasjon av digitale nøkler og tilgjengelig teknologi for dette er temaet i denne rapporten. Rapporten starter med en presentasjon av gjeldene prinsipper for nøkkelhåndtering og en diskusjon av de problemene som oppstår i tilknytning til PKI-teknologi (Public Key Infrastructure). En spesifikasjon for nøkkeladministrasjon over Web Service-protokoll, kalt XKMS, vil bli grundig diskutert. Rapporten fortsetter med en presentasjon av et W3C-styrt eksperiment for å vurdere interoperabilitetsegenskapene til XKMS.

Nøkkeladministrasjon i mobile nettverk er et spesielt komplisert tema, fordi mangel på pålitelig nettverksforbindelser gjør det umulig å la sertifikatvalidering bero på online ressurser. Rapporten skisserer noen mulige løsninger for nøkkelhåndtering i mobile nettverk.

Contents

Preface 7

1 Basic principles of cryptography 9

1.1 Symmetric cryptography 9

1.1.1 Assurances of symmetric cryptography 9

1.1.2 Key management of symmetric cryptography 9

1.2 Asymmetric cryptography and Public Key Infrastructure 10

1.2.1 Assurances of asymmetric cryptography 11

1.2.2 Forged public keys, man-in-the-middle-attack 11

1.2.3 Digital certificates 12

1.2.4 Assurances of digital certificates 13

1.2.5 Trust model of digital certificate chains 14

1.2.6 Revocation of certificates 15

1.2.7 Cross domain certificates 15

1.2.8 Key management of asymmetric cryptography 15

2 Principles of a PKI Repository 16

2.1 Introduction 16

2.2 Overview of the X.500 Directory System 17

2.2.1 Data Hierarchy 18

2.2.2 Service Model 19

2.2.3 The X.400 background 19

2.2.4 Adaption to the Domain Name System 20

2.2.5 The X.509 Digital Certificate 21

2.3 Overview of the LDAP protocol 24

3 Criticism of a worldwide public PKI 25
3.1 Introduction 25
3.2 Certificate Revocation List 26
3.3 Cross domain certificates 26
3.3.1 Transitive trust 26
3.3.2 Multiple certificate paths 27
3.3.3 Distribution of cross domain CRLs 28
3.3.4 Which name? 29
3.3.5 Which directory? 29

4 XML-based Key Management 30
4.1 Introduction 30
4.2 A short presentation of XKMS 2.0 32
4.2.1 Locate and Validate operations 32
4.2.2 XKMS Request Attributes 33
4.2.3 Adaption between abstraction layers 33
4.2.4 Protocol alternatives 34
4.3 Example XKMS transactions 35
4.3.1 Locate 35
4.3.2 Validate 36
4.4 An XKMS demonstrator server/client 37
4.5 An XKMS interoperability experiment 38
4.6 The W3C interoperability experiment 39
4.7 Concluding remarks on XKMS experiments 40

5 Key management for mobile networks 40
5.1 Introduction 40
5.2 Security vs. Latency 41
5.2.1 Certificate revocation 41
5.2.2 Certificate expiration 42
5.2.3 A hybrid configuration for mobile networks 43
5.3 The role of XKMS in a mobile network 44

6 Conclusion 46

Preface

The FFI project 1086 named “secure pervasive SOA” investigates how SOA principles can be applied to a military information system. The openness of SOA raises lots of security concerns with regard to integrity, authenticity, confidentiality and access control of services and of the information being processed by these services.

These concerns are presently being addressed on a large scale by the industrial community, and several standards have been established on how the security mechanisms may be represented by XML constructs. The basic mechanisms for signing and encrypting information is in place as well as frameworks for authorization and certificate management.

In a military environment, these standards may represent a challenge for the part of the communication infrastructure which has poor connectivity and low bandwidth. The protocols designed for a stable and high-speed network become too costly both in terms of transport volume and the number of necessary protocol interactions. Computational cost may also be a matter of concern, since the computing nodes on the mobile network are likely to include portable units with limited resources.

The purpose of this report is to investigate the need for certificate management, which is an important building block in a distributed security framework. The basic principles of certificates (and why they are needed) will be explained, and the most common architecture for certificate management, termed PKIX, will be presented.

The report will present XKMS, an XML-based key management service, and arguments on why there should be a loose coupling between the application software and the services for certificate management. The report will also present experiences from an experimental implementation of a XKMS server.

Finally, the report will discuss issued concerning XKMS in a mobile environment.

1 Basic principles of cryptography

1.1 Symmetric cryptography

The traditional type of cryptography uses the same key for sender and receiver, it is therefore called *symmetric* cryptography. Symmetric cryptography (or simply crypto) can be formally expressed as:

$$C = E(P, K) \tag{1.1}$$

$$P = D(C, K) \tag{1.2}$$

Where C denotes the cipher text (unreadable), P denotes the plain text, K denotes the common key and E and D denotes the functions for encryption and decryption, respectively.

Received information C (which can be decrypted with key K to meaningful information P) can only have been made by an actor with knowledge of K, and cannot be altered during transport. Others may receive or eavesdrop C but will not be able to understand the content unless they know the crypto key K. Consequently, symmetric crypto may guarantee integrity, confidentiality and authenticity to the extent described by the assurances of symmetric cryptography.

1.1.1 Assurances of symmetric cryptography

Symmetric crypto offers the following guarantees:

- The encrypted information C cannot be understood by anyone
- C can only be transformed to plain-text P with the knowledge of the crypto key K
- If C is modified, the resulting decrypted $D(C, K)$ will be unintelligible. Modification of C will therefore not remain undetected.
- It is *computationally infeasible* to compute K through observation of C or P

In symmetric crypto the shared key represents an authorization to take part in "trusted" communication. Symmetric crypto offers a form of authentication as well, since a valid P only can be made by someone who possesses the corresponding C.

1.1.2 Key management of symmetric cryptography

No trusted communication can take place until the parties have agreed upon a shared key, which must be generated and distributed to the members through a secure ("out of band") channel.

Keys should be replaced regularly and also when they may be compromised. New keys must be distributed using secure separate channels (one cannot use old keys to distribute new keys). When a member leaves a group that uses a shared secret key, a new key must be generated and distributed to the new set of group members. The cost of this process scales with the square of group size[1].

Key management poses the biggest problem in symmetric crypto. Every user or node must possess a secret key for every (pairwise or group wise) trusted communication channel. New keys must be generated and distributed often and impose a potential large cost (e.g. a courier service).

1.2 Asymmetric cryptography and Public Key Infrastructure

Asymmetric cryptography uses different keys for the encryption and decryption process. The two keys are mathematically related and are denoted *public* and *private* key.

$$C = E(P, K_{pub}) \tag{1.3}$$

$$P = D(C, K_{priv}) \tag{1.4}$$

It is computationally infeasible to find K_{priv} given P, C, and K_{pub}. K_{pub} can therefore be known by everyone in order to encrypt plain text P into cipher text C. The cipher text can only be decrypted through knowledge about K_{priv}, which is assumed to be secret to the owner of the key pair.

Anyone can now initiate a protected conversation with a person given his/her public key. The resulting cipher text can only be decrypted by the person's private key (which is secret to the person).

If the following equation is true,

$$P = E(D(P, K_{priv}), K_{pub}) \tag{1.5}$$

i.e. the plain text is first decrypted with the private key, then the resulting data encrypted with the public key getting the plain text back, we have an arrangement for *electronic signatures*. Based on the assumption that only the owner of the key pair is able to produce $D(P, K_{priv})$ anyone can verify that this person is the originator of a message that can be transformed to plain text through encryption with the user's public key.

The RSA algorithm [14] meets this requirement and can therefore be used both for encryption and signature. Other algorithms with key pairs can only be used for signature (e.g. DSA).

The E and D operations are computationally expensive, so for efficiency reasons the full plain text is not subject to encryption or signing. A hashing algorithm $H(P)$ generates a *message digest* which is subsequently signed:

$$Signature = D(H(P), K_{priv}) \tag{1.6}$$

[1]In addition, the number of groups may grow linearly with the number of users, leaving symmetric key management as an $O(n^3)$ problem.

The receiver of the signature produces a new message digest $H(P)$ and compares it with $E(Signature, K_{pub})$.

In a similar manner, a temporary session key K_{sess} is used to encrypt the plain text with a symmetric (and computationally cheap) crypto algorithm. K_{sess} is then encrypted with the receiver's public key and added to the message sent:

$$C = E_{symm}(P, K_{sess}) + E_{asymm}(K_{sess}, K_{pub}) \tag{1.7}$$

1.2.1 Assurances of asymmetric cryptography

On the condition that the users possess the correct public keys of each other and the private keys are kept secret, the users can safely assume that:

- Content encrypted with the public key of a person can only be read by that person
- Content encrypted with the public key of a person can not be modified without detection
- Plain text content decrypted with the private key of a person can only have been made by that person

It is important to realize that an electronic signature not necessarily bears legal or contractual implications. The legal consequences of an electronic signature must be defined by a contract between the parties.

1.2.2 Forged public keys, man-in-the-middle-attack

Unless care is taken to ensure that a public key of a person is safely distributed to others, a so-called "man-in-the-middle-attack" is possible. The attack happens if an attacker intercepts and modifies the communication between the sender and the receiver. During a man in the middle attack the attacker diverts the client (the one who initiates a session) to the attacker's computers instead of the correct server, e.g. through false DNS information[2].

Figure 1.1 shows a sequence of events where an attacker has successfully intercepted a communication between sender and receiver. The attacker then presents to the sender a forged public key, to which the attacker holds the private key himself. In the example shown an encrypted message is decrypted by the attacker before it is passed on to the receiver using the receiver's correct public key. The message is delivered as expected, so the attack may remain undetected. Not only will the content be exposed to the attacker, the attacker may even modify the message before it is passed on.

A signature operation would likewise be corrupted by this type of attack. The attacker may modify the message which would still be considered as correctly signed.

[2]DNS - Domain Name System, the service that translates Internet names to IP addresses.

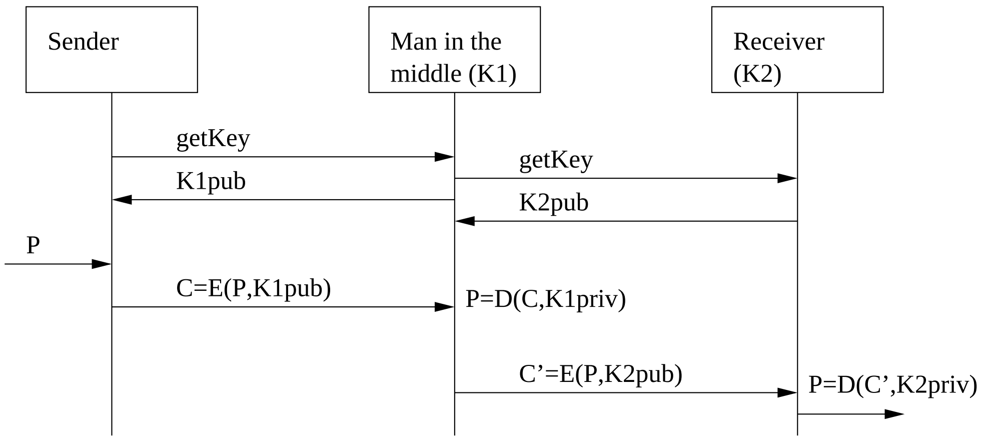

Figure 1.1: Man in the middle attack with forged keys

The man-in-the-middle vulnerability creates a need to assure the validity of public keys. A public key must be associated with a *principal* and the correctness of that association must be verifiable by everyone. *Digital certificates* will now be introduced as a means for such verification.

1.2.3 Digital certificates

The most common method for public key verification is through digital certificates. Every user on the network is assumed to be able to trust the validity of the public key of the *Certificate Authority* (CA). This trust can be accomplished by pre-installation of the key during e.g. software installation.

User1 may provide proof of his/her identity to the *Registration authority* (RA). The RA can then instruct the CA to generate a public key pair (public and private key) and send the keys to User1 using a secure channel (e.g. a storage medium sent by courier mail). User1's identity is a name that is meaningful within the domain of the application.

The CA will then issue a digital certificate of User1's public key with the following structure:

- User1's public key $U1_{pub}$
- User1's identity information (globally unique name, e-mail etc.) $U1_{id}$
- Validity period for the certificate
- Description of the authorized use of the key.
- CA's signature on the information above
- Technical info (CA's name, chosen crypto algorithms etc.)

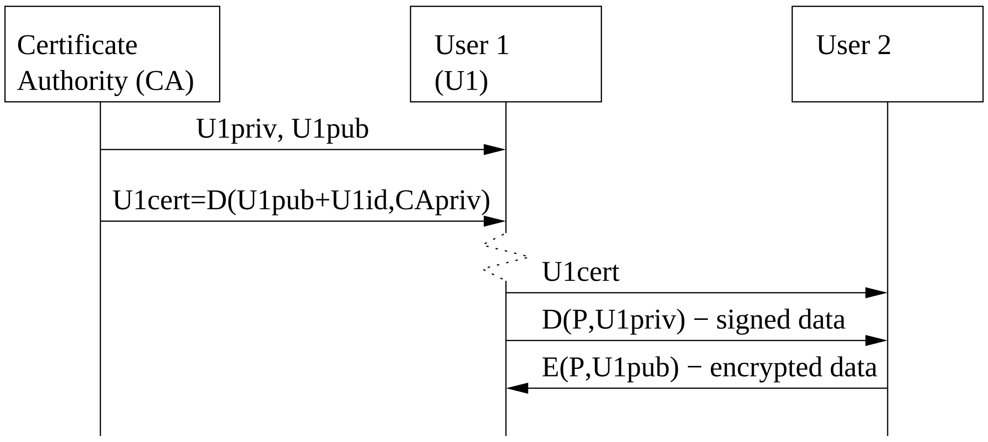
Certificate Authority (CA)
User 1 (U1)
User 2
U1priv, U1pub
U1cert=D(U1pub+U1id,CApriv)
U1cert
D(P,U1priv) − signed data
E(P,U1pub) − encrypted data

Figure 1.2: Certificate Authority issues a certificate on U1's public key

The digital certificate of User1's key is denoted $U1_{cert}$ and is expressed by the following formula:

$$U1_{cert} = D(U1_{pub} + U1_{id} + Validity + Authorizations, CA_{priv}) + \text{techinfo} \tag{1.8}$$

$U1_{cert}$ may be distributed to anyone, it is *not* a confidential document. It can be made available in the CA's certificate repository, disseminated by User1 or by any other practical means.

Alternatively, the user may generate the key pair herself, and pass to the CA a *self-signed certificate*, which proves to the CA that the user possesses both the private and the public key. The CA (or RA) may then verify the identity of the user (using any method) and issue a certificate on the user's public key. The advantage of this method is that the private key never needs to be transferred over a network link.

1.2.4 Assurances of digital certificates

Anyone that possesses User1's digital certificate may safely assume that:

- The association between the public key and the identity given in the certificate has been verified by the Certificate Authority
- The private key has been transferred to the key owner in a way that maintains the secrecy requirements
- The identity or the public key cannot be modified undetected

on the condition that:

- The CA has a trustworthy way of establishing the identity of a principal
- The private key of the CA is kept secret

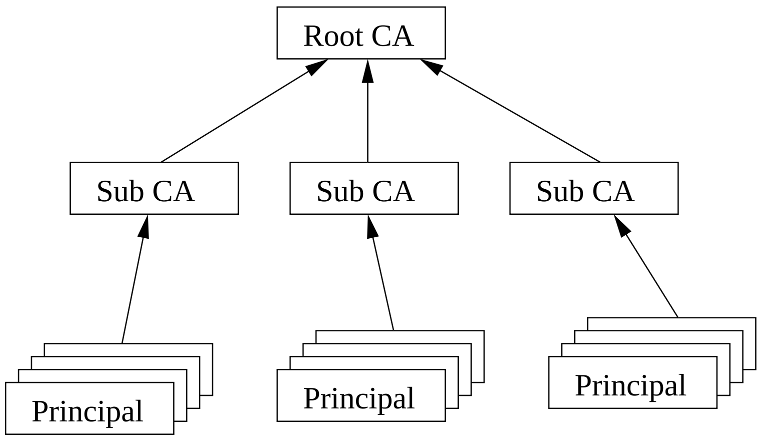

Figure 1.3: Certificate paths from principal to root CA

1.2.5 Trust model of digital certificate chains

A recipient of a digital certificate trusts its validity based on trust in the CA. The CA is trusted to keep its private key secret and to implement a policy to establish the identity of the principals before a key pair is generated and a certificate issued.

This is called a *trust chain*: I trust the certificate of User1 because I trust the CA. In the future I can trust another certificate signed by User1 if I trust User1's certificate issuing policy[3]. Certificates should not normally be issued by normal users, but by Certificate Authorities. The intended use of trust chains is therefore one where one CA delegates its authority to a sub-CA "branch office". The sub-CA is equipped with its key pair and a certificate indicating the authority to issue certificates (called a CA certificate). The sub-CA could be authorized to issue certificates with a limited validity period, certificates with limited use (e.g. no CA certificates), or to principals with identities within a given name space.

Figure 1.3 shows how principals possess certificates issued by a sub-CA. The sub-CAs possess certificates issued by the root CA. The certificates contain the identity of the issuer (indicated by the arrow), thus forming a *certificate path* from any certificate to the the root CA. The certificate of the root is self-signed (using the root CA's own key) and becomes the end of the path.

The root CA is called the *trust anchor* and is (by definition) trusted by everyone in the domain. In order to *validate* a certificate, each certificate on the path to the trust anchor must be inspected and approved. These certificates may be sent to the validating party together with the leaf certificate, the validating party can request them from a repository, or they may reside in a local cache. The details of certificate path validation will be discussed in more detail later in this report (Section 2.2.5).

[3]Anyone can issue a certificate signed with their own private key, but it is at the receiver's discretion to trust it.

1.2.6 Revocation of certificates

Although certificates have an expiration date, circumstances may require that they are revoked before they expire (e.g. the person owning the key pair is being reassigned). Below is mentioned three strategies for dealing with revoked certificates:

1. **Certification Revocation List (CRL):** The CA maintains a list which contains the revoked certificates which have not yet expired. A validating party can inspect the CRL during certificate validation. The CRL must be kept updated and widely disseminated, which is a resource-consuming process. The high demand for resources created by a CRL service is raising concern, and the entire concept of CRL is somewhat controversial, which will be discussed in Section 3.2.

2. **Partitioned CRLs:** The total CRL volume can be split into several lists, each representing a specified range of certificates. A revoked certificate is being listed on “its” CRL partition. Each certificate is annotated with information about the list which holds its revocation status. A client validating a certificate can download this CRL partition if not already in cache. Partioned CRLs may reduce the total volume of CRLs transferred over the network, but require a “pull”-model for distribution where clients always initiate the CRL transfer.

3. **Online Status Checking:** All revoked certificates are registered with a server, and a validating party may request the revocation status of a certificate using a client/server protocol. Online checking relieves the system from distribution of CRLs, but requires constant availability of network connections between the status server and the clients.

1.2.7 Cross domain certificates

When *Principal 2* wishes to present its certificate to *Principal 1* (see Figure 1.4) who has not the issuing *CA1* as its trust anchor, the certificate cannot be validated unless there exist a *cross domain certificate* between the respective CAs. A cross domain certificate is issued by *CA1* and binds the identity of *CA2* to its public key. This certificate will extend the certificate path from *CA2* to *CA1*, so certificates issued by *CA2* may be validated by parties having *CA1* as their trust anchor.

CA2’s certificate is (by definition) self-signed, so there is no reference to the cross domain certificate issued by *CA1*. Validation involving cross domain certificates must therefore use other means to locate them. This process is called *certificate path discovery*, which is an active field of research [13].

1.2.8 Key management of asymmetric cryptography

In a large community of principals/users, a facility is required for distribution and management of key certificates. Key management involves these tasks:

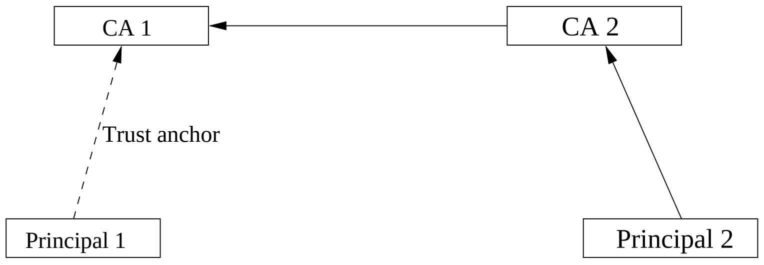

Figure 1.4: Certificate paths between CAs (cross domain)

- Generate key pairs and certificates
- Renew certificates of existing public keys
- Revoke certificates on invalidated keys
- Serve client requests for stored certificates and revocation lists

A set of services which issues, stores and disseminates certificates is called a *Public Key Infrastructure* (PKI).

In order to disseminate certificates (and revocation lists) among a large population of clients a large-scale distribution and storage service is required. This service should respond to interactive requests for certificate retrieval and validation and requests for lists of revoked certificates. The service must offer low latency and high availability and is therefore likely to employ replicated storage and distributed servers. This service is called a *PKI repository* and will be described in the next section.

2 Principles of a PKI Repository

In this section a brief introduction to certificate storage will be given. The emphasis will be put the service called *PKI repository*.

2.1 Introduction

If public key cryptography is being used on a large scale, certificate management becomes a critical issue. An agent who will be able to communicate with “anyone” needs a service that provides certificate retrieval, fast, correct and up to date. The service should also publish a Certificate Revocation List (CRL). There are several architectural alternatives for such a service:

Centralized repository A centralized solution ensures that the service always provides updated certificates, and that a revoked certificate never is wrongly validated. In general, a centralized solution provides the best consistency properties, but is also a scalability bottleneck and a single point of failure.

Local repository Each agent can keep a repository of all (necessary) certificates. They may be loaded into the agent during an initialization stage. A local repository is network efficient, but requires much storage space and fails completely to cope with the dynamics of new and revoked certificates.

Distributed repository A distributed repository consists of server replicas where each holds a portion of the certificates. The distributed repository manages the replication process, and make sure that all certificate requests are directed to a server that is able to serve the request. A client of a distributed repository is likely to keep a cache of frequently used certificates.

Peer to peer Certificates may be exchanged on demand in a peer-to-peer fashion. A certificate may be included in a signed message, or a certificate may be requested from a peer prior to the transmission of an encrypted message. CRLs may also be disseminated with the same mechanisms, although this is not seen in practice.

Any of the aforementioned techniques may be used in any combination where needed, driven by demands for consistency, latency[4], reliability, scalability, flexibility and security.

In particular, there exists a contradiction between latency and scalability. A centralized repository will offer poor scalability, poor reliability but zero latency, whilst a distributed repository will suffer some latency due to the frequency of replication procedures and the validity period of cached data.

In practice, a PKI repository is often constructed as an *X.500 Directory System* and with the use of LDAP protocols. X.500 and LDAP will be described in the following sections. An implementation of a PKI using X.500 Directory System, LDAP protocol and X.509 digital certificates (Section 2.2.5) is termed "PKIX". The distinction between the terms "PKI" and "PKIX" may seem unclear. In this report the term PKI refers to the principles and services, while the term PKIX refers to a set of IETF recommendations for the implementation of a PKI or instances of those recommendations.

It is a little unfortunate that there is such a tight coupling between the service principle of PKI and a specific implementation technology. A separation of the service specification from the service implementation may introduce a loose coupling between the two. An approach which follows this principle will be presented and analyzed in section 4.

2.2 Overview of the X.500 Directory System

The original mission of the X.500 Directory System was to offer a federated (possibly worldwide) directory of users, servers, peripheral units etc. so that service endpoints, e-mail addresses, certificates etc. may be retrieved. For reasons of scalability and availability, the architecture of an X.500

[4]The term "latency" is used to describe the time between the PKI is updated and the client retrieves the updated value.

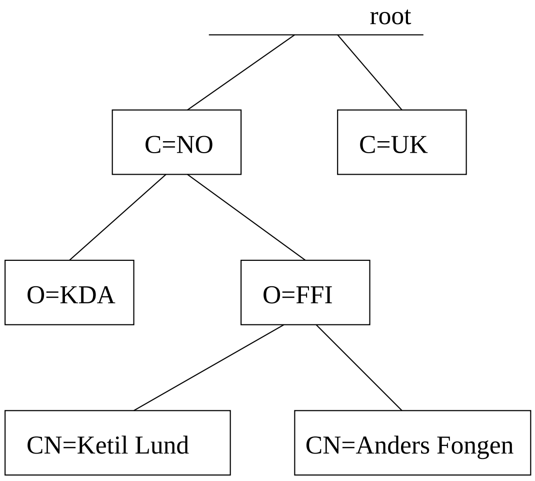

Figure 2.1: The data hierarchy of an X.500 directory system. The objects are labeled with their Relative Distinguished Names (RDN)

Directory System may employ replication and distribution; the database may be split on several servers, and data may be replicated and available in more than one server.

NATO has adopted the X.500 standard through the ACP 133 standard for military directories: "Common Directory Services and Procedures".

2.2.1 Data Hierarchy

The X.500 system stores *data objects* in a *hierarchical structure* under a *retrieval key*. The data objects consist of sets of $(name, value)$ attributes. The retrieval key is called a *Distinguished Name* (DN) and must be unique for the directory system.

The DN is an unordered set of $(name, value)$ attributes, e.g. `CN=Anders Fongen, O=FFI, OU=II, C=NO`[5]. The attributes that distinguish an object from its parent object (the attributes that it does not have in common with its parent object) are called *Relative Distinguished Names* (RDN)[6]. The names of the attributes (CN, O, OU, C) are not chosen freely, but defined in the X.521 recommendation and used to group objects in a hierarchical structure.

The objects in a directory system are organized in a tree structure based on the attributes they have in common and the hierarchical ordering of the attributes (see Figure 2.1).

The most commonly found attribute names in a DN are:

[5]the string representation of DNs are shown in accordance with RFC 1779.

[6]attributes are not completely unordered; if there are several attributes with the same name, then their relative ordering counts.

C	Country
ST	State or province
O	Organization
OU	Organizational Unit
E	E-mail
CN	Common Name

2.2.2 Service Model

The service model of the X.500 Directory System offers the operations necessary to search for, insert and delete object. The model maintains a concept of *context* for an operation, which is an object in the tree (e.g. “c=no,o=ffi”) on which subtree the operation will take place[7]. The available operations include:

Set context Input: the DN of the object which will serve as a context for subsequent operations.

Lookup Input: the RDN of the desired object. Together with the DN of the current context, the desired object’s DN is constructed and a retrieval operation on the data structure takes place. All attributes of the object are returned[8].

Search Input: a logical expression of the attribute values (e.g. “email=anders.fongen@ffi.no”). All objects in the subtree starting at the context object which evaluates the expression to `true` will be included in the returned result set. The search operation is potentially expensive, and the configuration of the directory system may limit search operations to lower levels in the directory tree.

Enter Input: an RDN and a set of attribute values. Based on the current context, an object is created and inserted into the directory tree. Restrictions may apply on which attributes are mandatory or legal.

Delete Input: The RDN of the object which should be deleted.

Some or all of the operations may be subject to access control and user authentication.

2.2.3 The X.400 background

The original DN form was inspired by the X.400 Mail system, where messages were routed in a hierarchical manner from national service providers (telecom companies) downwards to private enterprises, organizations, organizational unit and individuals. In an X.400 system, a DN designates

[7]Think of the context as a “current directory”.

[8]The LDAP protocol has an optional input parameter to select the attributes that are returned, but this parameter is left out of many APIs (e.g. Perl and Java JNDI).

a mailbox, and a typical form would be: `C=NO, ADMD=TELEMAX, PRMD=SAGA, O=IKT, GN=Anders, SN=Fongen`. The names of the RDNs are to be interpreted as follows:

C	Country
ADMD	Administrative Management Domain
PRMD	Private Management Domain
O	Organization
GN	Given Name
SN	Surname

Several other RDN names may be used as well, but they are left out for now.

The X.500 Directory System was designed to be the a directory of X.400 addresses (although other applications were identified), so the DN form was consequently used for retrieval operations, and the hierarchical structure of the X.400 system was reflected in the design of the X.500 Directory System.

2.2.4 Adaption to the Domain Name System

It is straightforward to add the DNS or E-mail address of an entity as an attribute in an X.500 object. What is not so straightforward, is to know that the e-mail address of “Anders Fongen” is found in the object with the DN “CN=Anders Fongen, O=FFI, OU=II, C=NO”. The same problem occurs for a receiver of an E-mail from “anders.fongen@ffi.no” who wishes to retrieve the sender’s X.500 object for additional information. There is no intuitive mapping between DNS or E-mail addresses and Distinguished Names.

For this reason, the DN may be given a form that accommodates DNS names in a straightforward manner. The RFC 2247[11] standard gives guidelines on how to do this and proposes the use of a “DC” attribute (Domain Component) in the DN which contains the list of DNS name components. According to RFC 2247, the email address “anders.fongen@ffi.no” could be mapped into the DN value “CN=anders.fongen, DC=ffi, DC=no” [9]. Since there are several DC attributes in the DN, their order is significant; they should be listed in the same order as they appear in the corresponding DNS form.

The DN form recommended by RFC 2247 appears to gain popularity due to the lack of coordination of the X.521 name system. The name space offered by the RFC 2247 recommendation mirrors the DNS name space and may therefore be governed by the DNS name authorities (e.g. Norid in Norway).

[9] Strictly speaking, the RFC 2247 document does not mandate the use of the CN attribute, only the DC attribute.

2.2.5 The X.509 Digital Certificate

For the purpose of PKI operation, the directory system would be used to store public key certificates. The X.509 standard provides the required definition of a certificate. In today's operation of a PKI, the certificate specification given in RFC 3280 [9] would be used instead, and the following presentation of digital certificates will be based on the requirements set by RFC 3280 (even though they are still called X.509 certificates in the RFC document).

In order for certificate operations to be interoperable, the *format* and the *processing rules* must be specified in detail. The X.509 certificate is coded in binary format based on an ASN.1 specification and is not directly readable. Several programming languages and command line utilities can be used to parse and present an X.509 certificate, however. The ASN.1 specification is slightly similar to a BNF (Backus-Naur Form) syntax specification or an XML DTD (Document Type Definition) specification, and would be readable for persons with some background in these languages.

The structure of a digital certificate is outlined in Section 1.2.3, and the X.509 certificate has a similar structure: It contains a *signed* and an *unsigned* part. The unsigned part contains information necessary to verify the integrity of the signed part, and is not protected against modification[10]. The signed part contains the parts which need integrity protection.

- Unsigned part (unprotected integrity)
  1. Algorithm identifier: Used to decide which algorithm that should be used during signature verification
  2. Signature value: The signature made by the certificate issuer to protect the integrity of the signed part
- Signed part (integrity protected)
  1. Version: the version of the specification used to construct this certificate. Should be 3 when RFC 3280 is used.
  2. Serial number: The certificate issuer must ensure a unique number of all certificates
  3. Algorithm identifier: Same value as in the unsigned part
  4. Issuer name: Distinguished name (DN) of the certificate issuer
  5. Validity: two date values which indicates the validity period of the certificate
  6. Subject name: DN of the subject who is associated with the public key in this certificate
  7. Public key algorithm: identifies the cryptographic algorithm which is to be used with the public key (e.g. RSA, DSA or Diffie-Hellman)
  8. Public key value: The value of the public key bound to the subject name

[10]Modification of the unsigned part of the certificate will only destroy the certificate, it is not feasible to forge a certificate by altering the unsigned part.

9. Subject unique ID (optional): A unique number for this subject. The use of a unique subject id allows reuse of DNs, but this not recommended by RFC 3280
10. Issued unique id (optional): Same as above, but for reuse of issuer DNs.
11. Extensions (optional): A set of properties used to guide the validation process of the certificate. Extensions are explained in the next paragraph.

The certificate extensions are attributes which may be added in order to provide additional information for the certificate validation process. The extensions may be either critical or non-critical. A critical extension *must* be recognized by the validator, otherwise the validation process should fail. The most commonly used extensions are:

Basic Constraint Indicates whether the owner of the certificate is allowed to issue certificates (e.g. is a CA) and how long a certificate chain can extend from this certificate

Subject Key Identifier Identifies this particular certificate in case several certificates are issued on the same subject name

Authority Key Identifier Identifies the issuing certificate in case the issuer has several certificates under the same issuer name

Key Usage Restrict the usage of the public key. The extension attribute is a bit string which indicates if the certificate can be used for signing, encipherment, certificate signing etc.

Subject Alternative Name In case several names need to be bound to the public key (e.g. DNS address, E-mail address or URL), this attribute can used to store these names

Issuer Alternative Name Same as above

Name Constraints In case the certificate belongs to a CA (which issues new certificates) this attribute can be used to limit the name space of issued certificates. The attribute value may constrain the name space to a subtree in the X.500 name tree or to a particular DNS domain

Subject Information Access A set of URLs which refers to services related to this certificate, e.g. the service endpoint of the relevant PKI repository.

CRL Distribution Points This attribute value contains the address (URL) of the service point where the certification revocation list (CRL) regarding this certificate can be downloaded.

Below is a text dump from a X.509 certificate, made by the command:

```
$ openssl x509 -text < example1.cer
```

The screen output shows the values of the different fields that have been explained in this section. At the bottom of the output there is a binary version of the certificate in Base64 format, so that the certificate can be extracted from a file in this format.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            b8:21:c2:20:c4:b2:73:c8
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=no, O=ffi, CN=FFI sub CA-2
        Validity
            Not Before: Sep  6 15:17:03 2007 GMT
            Not After : Sep  5 15:17:03 2008 GMT
        Subject: C=no, O=ffi, CN=Ingar Bentstuen
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:e1:17:e4:66:11:45:4c:18:ab:61:6e:84:ef:03:
                    84:dd:22:be:59:3f:fb:d5:9b:c7:ef:4b:65:14:c8:
                    af:23:18:24:9f:43:ef:6f:4a:86:72:8d:3d:46:4e:
                    ff:51:e2:27:92:5b:69:39:0e:56:bf:49:7c:30:00:
                    40:b7:50:74:36:1e:93:ac:c8:6d:9b:2d:bd:1c:12:
                    34:3f:18:f5:43:49:1b:c7:a8:cc:e5:94:56:37:88:
                    ee:fb:67:a5:23:15:db:6d:07:90:7e:fd:56:11:dd:
                    a5:dd:83:da:44:54:ac:45:ff:5e:85:53:36:23:f2:
                    b0:2a:4d:2f:c8:47:0d:aa:e9
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                25:B7:FA:9F:F2:3D:34:87:7C:99:62:0A:6C:06:22:F4:
                BA:64:A9:67
            X509v3 Authority Key Identifier:
                keyid:32:B4:13:49:5F:B7:50:7E:D2:29:0A:38:D7:F1:
                42:F7:D5:BF:F7:E3

    Signature Algorithm: sha1WithRSAEncryption
        58:28:4d:65:77:b8:5d:f5:d6:b5:f4:33:8a:2e:1c:e5:2b:4d:
        15:2b:18:97:77:00:1e:c8:6d:a5:30:19:68:84:ec:66:dc:6f:
        6e:9e:28:de:bc:89:f9:09:9c:1c:0d:eb:7a:f7:fd:f0:24:12:
```

```
        7d:22:75:36:a9:3b:ac:4d:d9:cb:9e:2f:ec:7a:8b:d9:c7:31:
        c4:bd:8d:80:27:2c:e1:2b:04:05:46:cf:1e:eb:6f:e2:a0:82:
        a1:f3:18:10:84:c9:24:12:f5:4a:0d:c8:92:f7:b5:95:cf:dc:
        30:c4:38:d2:2c:ea:06:37:53:e4:98:6b:82:53:d0:f7:15:3f:
        29:fc
```

```
-----BEGIN CERTIFICATE-----
MIICYDCCAcmgAwIBAgIJALghwiDEsnPIMA0GCSqGSIb3DQEBBQUAMDIxCzAJBgNV
BAYTAm5vMQwwCgYDVQQKEwNmZmkxFTATBgNVBAMTDEZGSSBzdWIgQ0EtMjAeFw0w
NzA5MDYxNTE3MDNaFw0wODA5MDUxNTE3MDNaMDUxCzAJBgNVBAYTAm5vMQwwCgYD
VQQKEwNmZmkxGDAWBgNVBAMTD0luZ2FyIEJlbnRzdHVlbjCBnzANBgkqhkiG9w0B
AQEFAAOBjQAwgYkCgYEA4RfkZhFFTBirYW6E7wOE3SK+WT/71ZvH70tlFMivIxgk
n0Pvb0qGco09Rk7/UeInkltpOQ5Wv0l8MABAt1B0Nh6TrMhtmy29HBI0Pxj1Q0kb
x6jM5ZRWN4ju+2elIxXbbQeQfv1WEd2l3YPaRFSsRf9ehVM2I/KwKk0vyEcNqukC
AwEAAaN7MHkwCQYDVR0TBAIwADAsBglghkgBhvhCAQ0EHxYdT3BlblNTTCBHZW5l
cmF0ZWQgQ2VydGlmaWNhdGUwHQYDVR0OBBYEFCW3+p/yPTSHfJliCmwGIvS6ZKln
MB8GA1UdIwQYMBaAFDK0E0lft1B+0ikKONfxQvfVv/fjMA0GCSqGSIb3DQEBBQUA
A4GBAFgoTWV3uF311rX0M4ouHOUrTRUrGJd3AB7IbaUwGWiE7Gbcb26eKN68ifkJ
nBwN63r3/fAkEn0idTapO6xN2cueL+x6i9nHMcS9jYAnLOErBAVGzx7rb+KggqHz
GBCEySQS9UoNyJL3tZXP3DDEONIs6gY3U+SYa4JT0PcVPyn8
-----END CERTIFICATE-----
```

2.3 Overview of the LDAP protocol

The original access protocol which was proposed for use with the X.500 Directory System was called Directory Access Protocol (DAP). This protocol was originally based on the OSI protocol stack, but was later adapted to the TCP/IP protocol stack. The DAP protocol turned out to be too heavy for many applications. Consequently, a Lightweight Directory Access Protocol (LDAP) was developed and standardized through the IETF organization. LDAP is now described in a series of RFCs and Internet drafts[11].

The LDAP protocol is based on the TCP/IP protocols, and the protocol data units (PDU) used by LDAP is formatted in ASN.1 (an OSI presentation protocol syntax). LDAP is widely supported by the standard library of most programming languages, and the required programming methods are well documented.

Directory Systems that only supports the LDAP protocol are frequently termed “LDAP servers”, even though the data and service model is defined by the X.500 series of recommendation, not the IETF RFC documents. RFC 2251 [17] clearly states that an LDAP server must implement the X.500 data and service model.

[11]see `http://www.mozilla.org/directory/standards.html` for the full list of documents [7 Dec 2007].

The LDAP protocol is a synchronous client-server protocol[12] with operations that reflect the operations in the service model. The basic operations are:

Bind Establish an authenticated session with the server

Search/Compare Return the set of objects that match a certain filter criteria. The same protocol operation is used for "lookup" operations and "search" operations. The context (Section 2.2.2 is provided as a parameter (not in a separate service call). All attributes of the objects will be returned unless a selection parameter specify otherwise

Modify Change (add/delete/replace) one or more attributes in an object, identified by the object's DN

Add Adds a new object to the directory. The DN of the object and all attributes are given as parameters

Delete Delete an object from the directory, identified by its DN

Modify DN Change the DN of the object. The operation may potentially move the object to a different subtree, including its subordinate objects.

An LDAP operation may return a *referral*, which indicates that the server cannot complete the operation, but another server might be able to. A server may return a referral if it does not have the necessary data for a read operation, or has a read-only copy of the data on which a modify operation cannot be performed. A client who receives a referral in return from a service call should proceed to call the same operation again on the server indicated in the referral.

3 Criticism of a worldwide public PKI

3.1 Introduction

The original idea behind the PKI concept was probably that all users inside the domain of an application should use the same trust anchor and the same certificate repository service. If the application domain is Internet e-mail, the idea of a worldwide CA and PKI repository raises concerns over scalability, interoperability and authority.

The rest of this chapter will present some of the controversy over the idea of global PKI and the use of CRL in particular. Most of the arguments are based on Peter Gutmann's tutorials and papers [7, 6], and Ronald Rivest's work in [15].

[12] Asynchronous protocol elements exist, but are poorly supported in high-level programming languages.

3.2 Certificate Revocation List

CRLs does not answer the question “is this certificate valid?”, but “is this certificate revoked?”. There are several reasons why these two questions does not cover the same situation: If information about a revoked certificate does not make it to the PKI repository, or the system fails to serve the CRL retrieval request for any reason, the answer to the second question would be “no”, even for an invalid certificate. A CRL attempts to answer the validation request through a negation, which is not a complementary situation.

The idea of CRLs is the same as “Credit Card Blacklists”, which the banks abandoned many years ago because it did not serve its purposes: The lists were never timely updated and the checking process was to costly[6].

One advantage of CRLs is that it allows for semi-offline certificate validation: If a certain update latency is allowed, the CRLs may be downloaded when situation allows it. CRL checking does not depend on the availability of a repository service at all times.

Scalability is another concern of CRLs. Since a CRL must contain all certificates revoked by this CA until they expire, the size of the CRL is likely to be large. The CRLs are annotated with an expiration date, after which every receiver of the CRL will try to download a new version, effectively generating a Distributed Denial of Service attack [7].

A third concern is that a revocation may include a date when the certificate became invalid, i.e. the revocation may say that a certificate has been invalid for the last couple of days! This feature violates the semantics of transaction processing, which says that a result of an operation is final and permanent once the transaction is committed. It also violates the legal principle of non-repudiation, although the legal consequences of backdated revocation may be mitigated through a contract.

3.3 Cross domain certificates

Cross domain certificates (described in section 1.2.7) is a nice way of extending transitive trust between domains, but only when used in a small scale. In a large scale, they raise several problematic issues:

3.3.1 Transitive trust

In the example from section 1.2.7, the certificate path was extended from CA2 to CA1. If CA2 issues a certificate to CA3, the certificate path extends from CA3 to CA1, forcing all receivers with CA1 as their trust anchor to accept certificates issued by CA3. This may not be CA1’s policy.

X.509 certificates may specify a limited path length, but since they do not distinguish between cross-domain certificates and sub-CA certificates (discussed in section 1.2.5), they may inadvertently block sub-CA delegation.

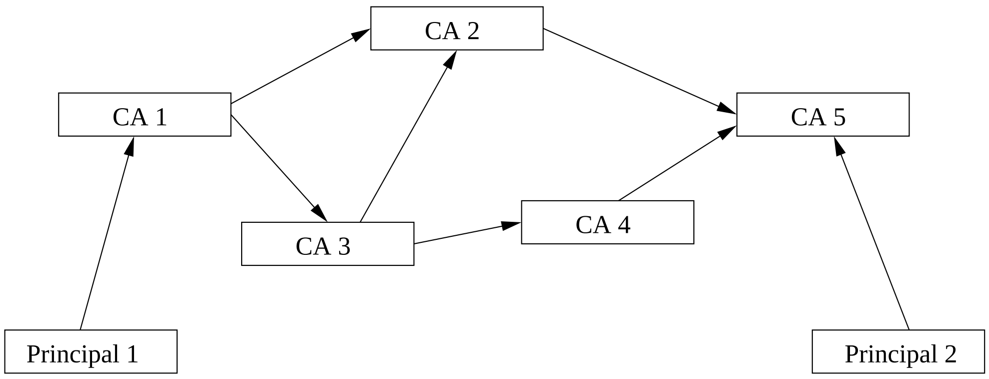

Figure 3.1: Redundant certificate paths between CAs

X.509 certificates may also specify *name constraints* which limits the name space of the certificates along this path (Section 2.2.5). Given the example in section 1.2.7, CA1 may certify CA2 in such a way that only CA2's certificates within a naming sub-space can be validated. This may prevent CA1 from trusting CA2's rogue certificates, but will also introduce an undesired coupling between the name space management of CA1 and CA2.

Transitive trust is not generally true.[4] In real life, no "blindfolded delegation" of authority or responsibility takes place. Trust is transitive only in well-defined context and limited in time, domain, and depth of sub-delegation [10]. Trust is likely to be even more constrained *between* organizations than *inside* an organization, where work contracts form a legal framework for the conduct of the delegates. The X.509 certificate extensions and rules for path validation [9] are unlikely to meet the complexity of trust transition in real life.

3.3.2 Multiple certificate paths

When several CAs issue cross-domain certificates to each other, their trust relationships (represented by certificates) form a graph with potentially multiple paths between a given pair of CAs. During certificate validation, any path may lead to a successful validation, depending on restrictions expressed as certificate extensions.

Figure 3.1 shows an arrangement of certificates between CAs. Certificate paths are indicated by the arrows, i.e. *CA2* has issued a certificate for *CA1*, *CA5* has issued a certificate for *CA4* etc. *Principal 1* now presents his certificate to *Principal 2*, who has *CA5* as his trust anchor. There are several paths from *Principal 1* to *CA5*: *(CA1,CA2,CA5)*, *(CA1,CA3,CA2,CA5)* and *(CA1,CA3,CA4,CA5)*. There are redundant paths in the graph formed by the cross-domain certificates: *CA2* can e.g. revoke the certificate of *CA1* (to indicate the termination of trust from *CA2* to *CA1*) without the desired effect, since the transitive trust in the path *(CA1,CA3,CA2)* may be used for verification of *CA1*'s certificates.

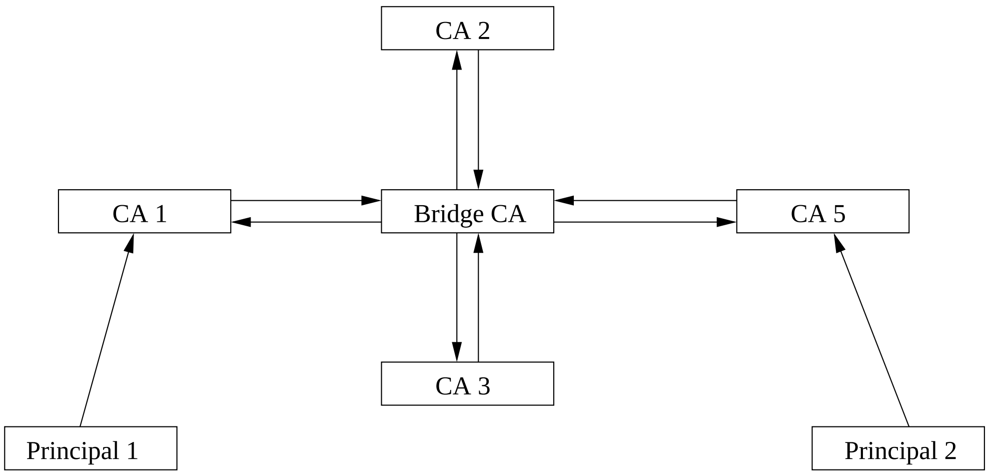

Figure 3.2: Bridge topology for cross domain certificates

Since CAs are assumed to be autonomous entities, management if this trust graph becomes impossible. There is little help in restricting the trust expressed by one certificate if there is another path bypassing the certificate with less restrictions. Revoking a certificate has no effect unless it results in a partitioned graph. With unrestricted cross domain certification, the “web of trust” becomes a “spaghetti of doubt”[7].

As a result of this problem, a restricted graph topology has been proposed. *The Bridge* (Figure 3.2) turns the general graph structure into a star shape, and the center of the star is a CA with no other duties than to issue cross-domain certificates to the connected CAs. All CAs must also issue their certificates to The Bridge, thus forming a two-way trust relationship between any pair of CA with a single path. The Bridge CA now has full control over the certificate paths and the effect of certificate revocations.

There are no scalability problems with bridges, since no actual data is transferred through the bridge, they are simply a set of certificates that may be fetched from a repository when needed.

A possible problem with bridges is that of *authority*. The bridge CA will be the entity that approves the CAs that wish to connect to the star, and will have to set requirements with regard to operational and managerial policy etc. Among a group of peer organization (e.g. NATO countries) there may not be a natural candidate for this role.

3.3.3 Distribution of cross domain CRLs

Since the use of cross-domain certificates broadens the domain of principals who may exchange their certificates, a larger set of CRLs need to be disseminated. In principle, a potential validator will need CRLs from every CA from which there exist a certificate path to the trust anchor. The scalability problems often associated with CRLs becomes worse as a result of this.

3.3.4 Which name?

The digital certificate is bound to the identity of a principal, and the identity is therefore required to be unique within the domain of the application. With the possible use of cross domain certifications in mind, the names should be *globally* unique as well. There are several naming systems which have the potential to be globally unique, e.g. the X.500 DN or RFC 822 e-mail addresses.

When retrieving a certificate from a PKI repository, the name (identity) of the certificate will often be the retrieval key. The name should therefore be on a form which is easy to remember (although search and browse functions may be offered). E-mail addresses are on a form which is familiar to many, but cannot be used as retrieval key from an X.500 directory system. X.500 allows only DNs (Distinguished Names) as retrieval key for data object.

The certificate retrieval process thus relies on a representation that is unfamiliar for most users[13]. It may lead to confusion if a validator receives a message from `anf@ffi.no` but will need to retrieve the sender's certificate using the DN form `C=NO, ORG=MIL, OU=FFI, CN=Anders Fongen`, since there is no obvious mapping between the two name forms. Some signature forms, like in XML-SIG and S/MIME, bundle the identifier together with the signature value, which facilitates the certificate retrieval operation.

If the PKI service is based on a technology different from X.500/LDAP, e.g. a Web Service or a SQL database, the problem is alleviated, since then other name forms can be used as a retrieval key.

3.3.5 Which directory?

Separate CA domains will most likely be served by separate PKI repository services. So, in order to retrieve or validate the certificate of "John Smith", where do we look? If there is a well-organized system of name spaces, where the different repository services have sub-spaces assigned to them, the question becomes a matter of looking up the service endpoint for that sub-space.

Among the name spaces that have this property are the RFC 822 email address, where the different DNS domain names can map to separate servers, and the X.500 Distinguished Name (DN), which is a hierarchical name system where subtrees can map to repository services.

If the validator already possesses an X.509 certificate, which should be validated through the use of the PKI service, this becomes a simpler situation. An X.509 certificate can be annotated with an extension called "Subject Information Access" which should contain the URI of the validation service for this certificate[14]. The RFC 3280 document is rather vague on the matter of message content for this service, e.g. how the respond messages should be encoded, and how error situations should be reported.

[13]In X.400-based electronic mail, (STANAG 4406) the DN is used for message addressing.

[14]The certificate may likewise contain the URL of a CRL distribution service regarding this certificate.

During configuration of a large-scale PKIX with many PKI service endpoints, these problems may consequently arise:

- The name to use as a retrieval key may not be known
- The name will not map to a service endpoint unless the name space is well planned
- The "Subject Information Access" extension does not contain sufficient information to avoid interoperability problems between PKIX implementations.

4 XML-based Key Management

4.1 Introduction

The services of a PKI if often implemented by a X.500 Directory system, which is a storage/retrieval system of data objects. X.500 (and its associated access protocol LDAP) offers storage, searching and retrieval services, but no application services like certificate validation.

When a X.500/LDAP based PKI is in use (called PKIX), certificate validation is supposed to be done by the client. Certificate validation includes the following tasks:

- Retrieving the principal's certificate
- Retrieving the chain of CA certificates all the way to the trust anchor
- Locating and retrieving necessary cross-domain certificates
- Parsing the certificate chain, looking for correct signatures, validity intervals, extension attributes for name constraints and usage constraints

This is a complicated process, but most CAs offer client libraries that solve this task in the client's software. There are, however, differences in the structure and interpretation of the certificates between CAs that may lead to interoperability problems[16], e.g:

- What protocols will be used for the retrieval of certificates and CRLs?
- How will cross-domain certificates be located and retrieved?
- Which certificate extensions are in use and how are their precise interpretation?

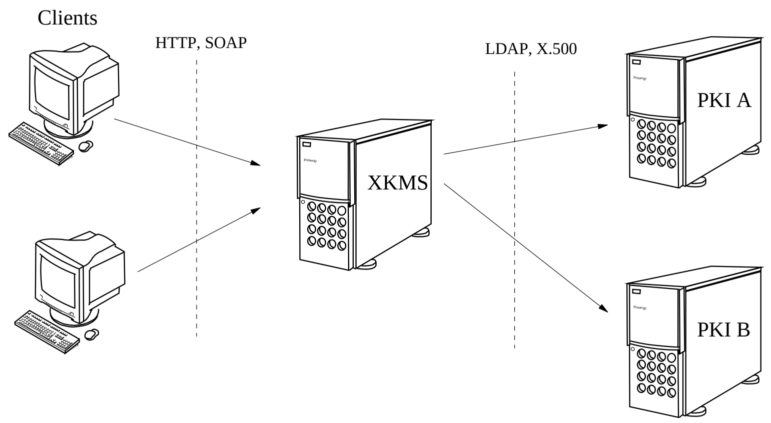

Figure 4.1: The XKMS server as an application server between the clients and the PKIX instances

There is thus a possibility that the client will need to have several brands of the client software installed, one for each CA which have signed certificates that the client wishes to validate. If this number is multiplied with the number of computing platforms that are present in the population of client computers, the emerging system administration bottleneck becomes evident.

A possible remedy to the problem is an *application server* that reduces the volume of installed software in the clients. The application server should offer application oriented services to the clients (e.g. locate and validate public keys) and execute the necessary repository retrieval and path validation operations. The application server becomes a *trust delegate* and a point of *centralized complexity*. The application server also becomes the client's new trust anchor, and the connection between the application server and the clients must be authenticated and integrity-assured.

Any RPC (Remote Procedure Call) mechanism can be used for this purpose. The *Server-based Certificate Validation Protocol*[5] (SCVP) is a relatively recent standard for remote validation of certificates. SCVP is related to PKIX and focused on the use of X.509 certificates.

This report will present an implementation based on web services protocol and with a wider scope, called XKMS (XML Key Management Specification)[8]. It will validate normal key values in addition to certificates. An implementation based on web services protocol becomes particularly important due to the emerging military applications based on web services.

XKMS offers basic key management services through a Web Service interface. A possible configuration where the XKMS server encapsulates a set of PKIX servers and offers a simplified service to clients is shown in Figure 4.1.

The XKMS standard is published by the W3C (World Wide Web Consortium), and the rest of this section will present an overview and some details about XKMS.

4.2 A short presentation of XKMS 2.0

XKMS offers key management services from a trusted server over a web services protocol. There are two sets of services offered:

KISS Key Information Service Specification. The service used by clients that need to obtain and validate certificates and public keys.

KRSS Key Registration Service Specification. The service used by clients to enter and obtain their own key pair, and by administrators for key administration purposes.

In this section, only the KISS service will be presented, since the pilot implementation (described in Section 4.4) only implements the KISS protocol.

The XKMS standard does not only describe an intermediary WS server for PKIX repository operations on X.509 certificates, but mediates operations on any storage system and a wide range of keys and certificates (S-MIME, PGP, etc.)

The XML syntax used for transport of keys and certificates is not defined by XKMS but by XML Signature (XML-SIG)[3].

The principle of KISS is quite simple: The client supplies identity information in full or partial form, and specifies which items of information it needs on this identity. The client may supply the public key and request the name of the owner, or the other way around. It can also send the identity name (e-mail address, X.500 DN etc.) and request the corresponding certificate. The scheme is very flexible, and all options will not necessarily be supported by a server implementation.

4.2.1 Locate and Validate operations

The KISS protocol offers *Locate* and *Validate* operations. The difference between the two operations is that a Validate operation offers a “verified” binding between the identity and the keying material (key or certificate), while the Locate service offers only “unverified” information. The difference between the two does not appear to be important in configurations where the keying material is stored in a centralized repository like a database, since the stored keying material is likely to be correct and verified by definition (keys are removed when expired or revoked). In a distributed environment where there is a latency from the moment the key is declared invalid until it is not longer retrievable by clients, the difference becomes important. In this case the Locate operation will return the information from the nearest and fastest storage service, whilst the Validate operation will consult more authoritative sources, study CRLs, inspect certificate paths etc. in order to provide verified information. Besides, a Validate operation would likely require an authenticated server response.

4.2.2 XKMS Request Attributes

The XKMS request message is formatted as a SOAP (Simple Object Access Protocol) message which uses the XML syntax. The message contains a set of request parameters (in the form of XML elements), some of which are optional and some are mandatory. The parameters are:

Application (mandatory) Identifies the application domain of the keying material. Possible values are: PKIX, PGP, S/MIME, TLS, IPSEC etc.

Key usage (optional) Possible values: Encryption, Signature,Key exchange

Identifier (optional) Identity of the key owner (e.g. E-mail address or X.500 DN)

Key Information (optional) A set of XML elements that supplies info (parts or all)about the keying material of interest.

Respond with (mandatory) A list of Keyinfo elements which should be supplied in the response from the server, e.g. "Public key", "X.509 Certificate", "KeyUsage".

All optional elements cannot be left out: Either the "Key Information" or the "Identifier" information must be provided in order for the server to locate the corresponding keying material.

4.2.3 Adaption between abstraction layers

The operations offered by XKMS contribute to a lightweight client configuration since they are application oriented rather than architecture-oriented. They allow a range of key management needs to be easily solved over a familiar protocol. A verified binding between an identity and a public key is basically what is needed for:

- Verification of document signatures
- Encryption of documents before sending
- Authentication of parties in client/server transactions

The traditional architecture for identity management is based on PKI with X.500 Distinguished Names, LDAP protocol and X.509 certificates (PKIX), which offers services at a much lower abstraction level. It is therefore necessary to study how the XKMS interface may be adapted to a PKIX service, and how the service parameters given in an XKMS request can be mapped onto a set of parameters given to the certificate validation algorithm. In general, PKIX operations have a richer set of parameters for selection and validation than XKMS.

In particular, the `KeyUsage` element and the `Application` attribute (of the `UseKeyWith` element) in the XKMS specification need a non-obvious mapping to the validation algorithm.

The `KeyUsage` element is a request parameter which indicates the intended use of the key. Its possible values are *(Signature, Encryption, Exchange)* and the element can occur $0..3$ times.

The RFC 3280 KeyUsage extension is a 9-bit vector that associates a public key with any of 9 usage categories. Those 9 categories are, with one exception, different variants of encryption and signature. Encryption is covered under the values “keyEncipherment”, “dataEncipherment” and “encipherOnly”, and the RFC document does not give precise instructions on how these values should be applied to use cases.

The chosen mapping between XKMS and PKIX parameters is shown in the table below. During the processing of a KISS request, the KeyUsage element is converted into a mask, which selects the certificate candidates with a logical AND operation on the KeyUsage extension. If either the KeyUsage element or the KeyUsage extension is absent, all candidate records are returned.

XKMS KeyUsage element	**RFC 3280 KeyUsage extension**
Signature	digitalSignature $\wedge$ nonRepudiation $\wedge$ keyCertSign $\wedge$ cRLSign
Encryption	keyEncipherment $\wedge$ dataEncipherment $\wedge$ encipherOnly $\wedge$ decipherOnly
Exchange	keyAgreement

In a similar manner, the *Application* attribute of the XKMS *UseKeyWith* element provides an option for the requester to indicate the intended application domain for the keying material. The attribute does not have a counterpart in the PKIX recommendations, but the demonstrator implementation requires that this element, if present, must have the value “PKIX”. Other values causes a null-operation.

XKMS Application attribute	**PKIX response**
PKIX	Normal operation
other values	no operation

4.2.4 Protocol alternatives

XKMS offers a few alternative mechanisms for service invocations:

Two-phase invocation Intended as a countermeasure against “Denial of Service” attacks. During the first phase, the server replies with a random number (a “nonce”) which is subsequently included in the next request phase, which then completes the service.

Asynchronous invocation The client annotates the request with an address to which the response can be “pushed back” at a later instant, e.g. with SMTP or HTTP protocol.

Compound requests Several requests can be bundled together and marked with different Id-attributes. The server will process the parts individually and return a bundled response.

4.3 Example XKMS transactions

4.3.1 Locate

A client wishes to obtain an encryption key bound to `bob@example.com`, so it can be able to send an encrypted mail to Bob. The client secure email format is S/MIME. The processing mode is synchronous. The resulting set of messages will consist of a Locate Request to the server and the Locate Result returned.[15] SOAP headers are not shown.

Message Request:

```
<?xml version="1.0" encoding="utf-8"?>
<LocateRequest Id="..." Service="..."
    xmlns="http://www.w3.org/2002/03/xkms#">
  <RespondWith>http://www.w3.org/2002/03/xkms#KeyName
  </RespondWith>
  <RespondWith>http://www.w3.org/2002/03/xkms#KeyValue
  </RespondWith>
  <QueryKeyBinding>
    <KeyUsage>http://www.w3.org/2002/03/xkms#Encryption
    </KeyUsage>
    <UseKeyWith Application="urn:ietf:rfc:2633"
      Identifier="bob@example.com" />
  </QueryKeyBinding>
</LocateRequest>
```

Message Response:

```
<?xml version="1.0" encoding="utf-8"?>
<LocateResult xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
      Id="..." Service="..."
      ResultMajor="http://www.w3.org/2002/03/xkms#Success"
      RequestId="..."
      xmlns="http://www.w3.org/2002/03/xkms#">
  <UnverifiedKeyBinding Id="...">
    <ds:KeyInfo>
    <ds:KeyName>...</ds:KeyName>
      <ds:KeyValue>
        <ds:RSAKeyValue>
```

[15]example taken from `http://www.w3.org/2001/XKMS/Drafts/test-suite/CR-XKMS-test-suite.html#XKISS-T`

```
          <ds:Modulus>...</ds:Modulus>
          <ds:Exponent>...</ds:Exponent>
        </ds:RSAKeyValue>
      </ds:KeyValue>
      <ds:X509Data>
        <ds:X509Certificate>...</ds:X509Certificate>
      </ds:X509Data>
    </ds:KeyInfo>
    <KeyUsage>http://www.w3.org/2002/03/xkms#Encryption
    </KeyUsage>
    <UseKeyWith Application="urn:ietf:rfc:2633"
      Identifier="bob@example.com" />
  </UnverifiedKeyBinding>
</LocateResult>
```

4.3.2 Validate

A client wishes to check whether a certificate supplied by a sender (Alice) in a message is valid or not, so he sends the certificate chain to the XKMS service. The processing mode is synchronous. The certificate is valid and it has not been revoked. The resulting set of messages will consist of a Validate Request to the server and the Validate Result returned reporting that the key binding has successfully been checked.[16]

Message request:

```
<?xml version="1.0" encoding="utf-8"?>
<ValidateRequest Id="..." Service="..."
    xmlns="http://www.w3.org/2002/03/xkms#"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <QueryKeyBinding>
    <ds:KeyInfo>
      <ds:X509Data>
        <ds:X509Certificate>...</ds:X509Certificate>
        <ds:X509Certificate>...</ds:X509Certificate>
      </ds:X509Data>
    </ds:KeyInfo>
    <KeyUsage>http://www.w3.org/2002/03/xkms#Signature
    </KeyUsage>
    <UseKeyWith Application="urn:ietf:rfc:2633"
       Identifier="alice@example.com" />
```

[16]example taken from `http://www.w3.org/2001/XKMS/Drafts/test-suite/CR-XKMS-test-suite.html#XKISS-T2`

```
  </QueryKeyBinding>
</ValidateRequest>
```

Message response:

```
<?xml version="1.0" encoding="utf-8"?>
<ValidateResult xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
      Id="..." Service="..."
      ResultMajor="http://www.w3.org/2002/03/xkms#Success"
      RequestId="..."
      xmlns="http://www.w3.org/2002/03/xkms#">
  <KeyBinding Id="...">
    <ds:KeyInfo>
      <ds:X509Data>
        <ds:X509Certificate>...</ds:X509Certificate>
      </ds:X509Data>
    </ds:KeyInfo>
    <KeyUsage>http://www.w3.org/2002/03/xkms#Signature
    </KeyUsage>
    <UseKeyWith Application="urn:ietf:rfc:2633"
       Identifier="alice@example.com" />
    <Status StatusValue="http://www.w3.org/2002/03/xkms#Valid">
      <ValidReason>http://www.w3.org/2002/03/xkms#Signature
      </ValidReason>
      <ValidReason>http://www.w3.org/2002/03/xkms#IssuerTrust
      </ValidReason>
      <ValidReason>http://www.w3.org/2002/03/xkms#RevocationStatus
      </ValidReason>
      <ValidReason>http://www.w3.org/2002/03/xkms#ValidityInterval
      </ValidReason>
    </Status>
  </KeyBinding>
</ValidateResult>
```

4.4 An XKMS demonstrator server/client

As a part of an XKMS feasibility and interoperability study a demonstrator of a PKIX-based XKMS server was constructed. The elements of this demonstrator was:

- The Certificate Authority was implemented with OpenSSL[17], an open source library for sig-

[17]see `http://www.openssl.org`

nature, encryption and certificate management. OpenSSL was used to create a set of CA certificates in a hierarchical structure, and set of user certificates was issued from the subCAs. OpenSSL was also used to create small CRLs.

- The PKIX repository was implemented with OpenLDAP[18], an open source implementation of an LDAP/X.500 directory system. The directory system was manually loaded with certificates and CRLs, and was acting as a back-end server for the XKMS service.
- The XKMS server was programmed as a Java servlet[19] which used the SAAJ library for SOAP protocol support. The SAAJ is a part of the standard Java Developer's Kit (JDK) distribution and does not have to be downloaded separately. The Java servlet receives client requests as SOAP messages over HTTP protocol, sends LDAP requests to the PKIX repository and uses the response from the PKIX (containing certificate information) to construct an XKMS response.
- The XKMS client was written as Java application over the SAAJ library and did not offer a user interface. The client was only used to construct a variety of messages and study the response. The Java classes which was written during the construction of the client may be re-used by other clients.

4.5 An XKMS interoperability experiment

The home-brew XKMS system was used to test interoperability issues on existing XKMS systems. Three more implementations were included in the test:

1. WSo2 (Web Services Oxygen Tank[20]) is an application server for Java which also includes an XKMS implementation in the form of a *service provider interface*. It handles the "protocol part" of XKMS, but not the service part. WSo2 simply passes the DOM tree[21] to the service provider, and offers little added value over the SAAJ-library. WSo2 also offers a client API with a simple interface for the KISS protocol (Locate and Validate services).
2. SQLData[22] offers an XKMS client in the form of a COM+ component, which can be employed by most programming languages on the Microsoft Windows platform. It also offers a server component for sale, and an on-line test service (which requires the use of a two-phase invocation protocol).
3. *The Markup Security project*[23] offers an on-line test service which has been a part of the XKMS interoperability experiment (termed "TL" and "TL Server" in Section 4.6).

[18]see `http://www.openldap.org`
[19]A servlet is a web server component able to service requests from web browsers and web service clients.
[20]see `http://www.wso2.org`
[21]Document Object Model, a general internal form of an XML document.
[22]see `http://www.sqldata.com/XKMS.htm`
[23]see `http://markupsecurity.com/info/xkms/index.html`

The interoperability experiment conducted as a part of this study was simply to start a simple KISS Locate request from the three different clients (FFI, SQLData, WSo2), on the three different servers (FFI, SQLData, MarkupSec).

Most of the “cross-product” invocations failed due to small problems in the XML syntax, or because the products used optional elements of the protocol differently.

The network analyzer that was used to observe the actual traffic which was sent over the network also observed problems which may potentially cause problems. These problems were related to violations of protocol rules (in the XML syntax or the HTTP headers).

The experiment contributed to debugging of the demontrator XKMS server and client, but was otherwise unsuccessful in the study of interoperability properties. XKMS interoperability issues will be further studied as part of our project.

4.6 The W3C interoperability experiment

During the development process of the XKMS 2.0 specification an interoperability experiment took place. Implementation/interoperability experiments appear to be a requirement in order to move a specification from “Candidate Recommendation” to “Proposed Recommendation” in the W3C standardization process.

The interoperability experiment was built around a *test collection*[24] consisting of a series of messages which are sent to a server with an expected response. The tests are related to specified assertions in the XMKS specifications. There were 36 test scenarios, mostly for testing different KISS and KRSS details.

Seven different client implementations (termed BL, RS, YZ, VM, TL, GA and RL) participated in the test, as well as four servers (termed TL Server, Entrust, ASF-XKMS and SQL Data). In order to meet the criteria for accepting the XKMS as a “Proposed Recommendation” at least two client implementations must make contact with at least two server implementations in a number of areas.

Of the four servers, two (TL Server, SQL Data) implemented all the tests and were contacted by at least two different servers each. Of the seven clients, three implemented all the tests (TL, GA, VM).

An interoperability matrix was made on the basis of the experiment results, which indicates that all tests were successful in the sense that at least two servers were contacted (assuming a successful contact) by at least two clients.

A detailed report on the observed interoperability problems is not given, nor detailed information on the conduct of the experiment. The experiments appear to be done in the developer’s office and the results are submitted through questionnaires. It is possible that debugging and “retrofitting” of

[24]see `http://www.w3.org/2001/XKMS/Drafts/test-suite/CR-XKMS-test-suite.html`

the programs have taken place during the experiments, and that potential interoperability problems have been masked.

The detailed results from the interoperability experiment is shown in the report *XKMS Candidate Recommendation Implementation Report*.[25]

4.7 Concluding remarks on XKMS experiments

The principle of *delegation* is well accepted in distributed theory. Also in the case of key and certificate management delegation gives a clear benefit for the client configuration; the middleware becomes less bloated and the configuration management of clients becomes easier. XKMS is therefore of great interest.

The interoperability testing conducted at FFI was not that encouraging, however, which is found to be due to errors in the implementation of the XKMS (and even XML) standards. The XKMS recommendation is very comprehensive, and implementations are likely to implement a subset of it. Two servers were only allowing the two-phase protocol (although this is an optional element), excluding those clients without this feature. There appears to be a need for profiles that refine the standard protocol requirements.

The interoperability experiment conducted by W3C indicates that the precision of the specification is good enough to avoid elementary interoperability problems, which also suggests that the observed problems are due to implementation issues.

Given these observations, it is recommended that at least one of the parties (client or server) is based on an in-house written implementation of XKMS when conducting demonstration and experimental activitites. In such cases interoperability problems may be solved by editing the program code. Open Source implementations exist (WSo2), but these are embedded in a large application server which will consume resources unnecessarily. The test implementation used in the described experiment is based on the SAAJ library and is a good candidate for an in-house product.

5 Key management for mobile networks

5.1 Introduction

Mobile networks are quite different from stationary networks in a number of aspects: The radio-based network links are frequently broken due to radio propagation problems (distance and obstacles) and the available bandwidth is several orders of magnitude lower than what is available in a stationary network with cables.

[25]see `http://www.w3.org/2001/XKMS/Drafts/test-suite/CR-XKMS-Summary.html`

For these reasons, key management (certificate validation etc.) in a mobile network should not rely on:

- Availability of on-line network servers
- Protocols with high bandwidth demand

5.2 Security vs. Latency

There exists a contradiction between the latency of a PKI and the level of security it offers. The requirements of a PKI should specify the maximum time lapse allowed from a certificate is declared invalid by the CA until it is invalidated in every client, i.e. the time it takes to distribute a revocation of this certificate.

If the latency is assumed to be zero, every certificate validation operation must consult a *centralized* repository in order to check the validity of the key. This hypothetical solution scales poorly and creates a single point of failure. It contradicts the property of offline validation of public keys, and can even be implemented using symmetric cryptography only.

> *Overestimating latency requirements pushes the design towards an on-line implementation and contradicts the original ideas of digital certificates.*

For any practical purposes the acceptable latency will have to be non-zero. A public key cannot be used after:

- The revoked certificate has been announced in a CRL which has been received by the validating agent, or
- the validity period of the certificate has expired.

(Online status checking (using e.g. Online Certification Status Protocol[12]) is disregarded since it requires an available network connection at all times.)

Two schemes exist for the control of latency, certificate *revocation* and *-expiration*. A brief discussion of the two alternatives follows:

5.2.1 Certificate revocation

The use of certificate revocation lists (CRLs) requires that every validating agent (a PKI client or an XKMS server) possess a recent copy of the CRLs in distribution. This copy can be obtained by regular distribution (push-based), by client request (pull-based) or a combination of these. The CRLs expires after a specified period, after which they should be disregarded.

In order to choose a reasonable scheme for CRL distribution, one should estimate a range of parameters:

1. The bandwidth between the client and the CRL distribution point
2. The availability of a network connection between the client and the CRL distribution point
3. The availability of a multicast service in the underlying network.
4. The update frequency of the CRLs
5. The size of the CRLs
6. The client's frequency of certificate validation
7. The number of different certificates which the client will validate

Given these parameters, it is possible to provide an analysis of the traffic requirements when different CRL distribution schemes are in effect. The problem of optimization of revocation schemes is thoroughly investigated, and a presentation of this research is provided by Årnes et al.[2].

In the case of a mobile tactical network, where the typical communication pattern includes a small number of nodes, the distribution of "full" CRLs appear to be a waste of precious bandwidth. It would be preferable to distribute a CRL where only the relevant nodes were included [26]. For this to happen, there must be an agent that

1. Knows which nodes are on a mobile network and who they communicate with
2. Is trusted by the mobile nodes to sign CRLs (i.e. can present a certificate path to their trust anchor)

The feasibility of a solution that distributes CRL partitions (Section 1.2.6) in a mobile network is not investigated in this report, but left for further study.

5.2.2 Certificate expiration

A different approach would be to abandon the CRL mechanism all together and rely on the validity period of the certificate. Using this approach, there are no way to stop a compromised certificate from operating until it expires. The validity period of the certificate becomes the worst case latency for the revocation mechanism.

[26]Distribution of partitioned CRLs could seem to reduce the network load, but partitioned CRLs do not automatically contain more relevant entries

Using this approach, it is necessary to issue certificates with shorter validity periods, since this value now directly relates to the security of the system. Consequently, the traffic volume resulting from the distribution of issued certificates will also increase.

A typical PKI configuration (e.g. as specified by NSM in [1]) will issue certificates with a validity period of a few months, and update its CRLs several times a day. In order to provide a reasonable latency, the number of issued certificates will therefore be multiplied with a factor of 100-1000.

Informal analysis (conducted by the author) indicates that this approach will generate more network traffic than the approach outlined in Section 5.2.1 under most circumstances, and put a greater workload on the CA, which need to re-issue the certificates through expensive signature operations.

When a certificate expires, it is no longer usable and will never be validated. This is a likely situation in a mobile network if short-lived certificates are used, since mobile networks will experience frequent disconnections. In the approach outlined in Section 5.2.1, a disconnected network will inhibit the distribution of updated CRLs and a cause gradually increasing risk for incorrect validation of certificates.

5.2.3 A hybrid configuration for mobile networks

It appears to be a good idea to combine the approaches mentioned in sections 5.2.1 and 5.2.2 in the case of a heterogeneous network with both mobile and immobile nodes and a mix of high- and low-bandwidth links.

It probably does not make sense to base an enterprise network on certificates with only a few hours lifetime. It neither makes sense to distribute enterprise-wide CRLs to a group of soldiers on the battlefield with a very restricted communication pattern.

It seems reasonable, however, to develop a key management system which is aware of the network topology and behaves accordingly:

- Only distributes CRLs in the part of the network which is well connected and fast
- Load mobile units with necessary certificates while they are docked (connected to a wired network)
- Monitor certificate exchanged in the mobile nodes in order to tailor-make CRLs relevant for the mobile network.

The clients in both environment (mobile and immobile) will use the same code library for certificate validation, so the processes outlined above must be transparent to the clients. This means that changes to the “traditional” key management must be done within the PKIX framework and strictly adhere to the protocols and and data structures described in these standards.

5.3 The role of XKMS in a mobile network

An XKMS server is a lightweight component, compared to a PKIX server. This is due to the fact that only a rather simple SOAP protocol is required to serve client requests. Behind the service interface the server is at large liberty to use any technology, e.g. local services (e.g. flat files or memory cache), remote services (e.g. RDMS or other Web Services) or any distributed service.

These properties make XKMS interesting for mobile networks. All mobile nodes are able to contain a small XKMS server and all local software on this node will be configured to contact this server in order to locate and validate certificates. Obviously, this server will be available regardless the networking conditions. The implementation of the local server can employ different strategies for its service:

- Forward the request to an authoritative XKMS server
- Consult the local cache for previous results
- Contact other mobile nodes for cached results
- Speculate[27]

Trust becomes a central issue when the local XKMS server have to rely on other sources than an authoritative key managements server. When peer nodes contribute with their information, they should be able to present a relatively recent validation response from an authoritative source. The policies on how to manage "peer trust" and the implementation of that policy is left for further study.

Figure 5.1 shows how a mobile network may equip all nodes with a local XKMS server, and how these XKMS servers may collaborate with each other as well as communicate with a central resource. The underlying grid of radio connections between the mobile nodes are not shown, only the chain of XKMS requests from soldier A to the centrally located XKMS server.

The client software in soldier A's node communicates only with the local XKMS server and is unaware of any further communication (which may not be IP, or may not be available at all). The request may be forwarded without any intermediate processing, or soldier B may process the request and return the answer based on information in its local cache. The same holds for vehicle C or the central XKMS server. In the same manner as HTTP proxies, the local XKMS servers behave like a set of proxies that contribute to faster response and less network traffic.

Management of trust becomes important under such an arrangement. The client software in soldier A only trusts the local XKMS server (it is not aware of any other server). If B chooses to process the request, then the communication between A and B must include mechanisms to ensure the identity of B and the integrity of the message. B can also choose to use a (signed) response from the central server from a previous invocation, if B consider it reasonably recent.

[27] In the absence of information, both accepting and rejecting a certificate involves a risk.

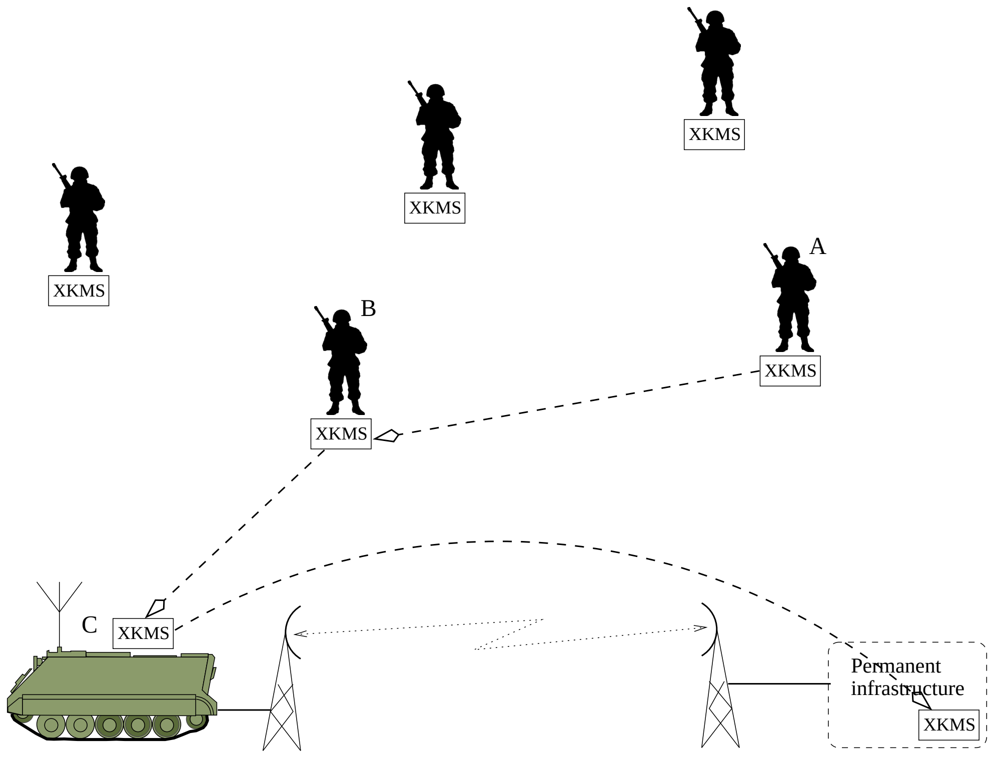

Figure 5.1: Local XKMS servers in every mobile node communicate and coordinate their activities during processing of a client request

The analysis of an XKMS proxy network must consider the balance between an unsafe validation and a safe refusal. Both options has its risk and cost, which must be chosen between in those cases where safe and recent validation info is impossible to obtain.

6 Conclusion

The principles of asymmetric cryptography and digital certificates have been presented in this report. The need for a Public Key Infrastructure (PKI) has been discussed and the PKIX architecture throughly presented.

The PKI architecture leaves a lot of unsolved problems, however. These problems are related to configuration issues, technical problems, managerial bottlenecks, open legal matters, revocation issues etc. A global PKI, where we can validate certificates from anyone in the whole world, is probably an unrealistic idea due to these unsolved problems.

The report has further presented the XKMS service specification, which places an application server between the PKI service and the validating clients and relieves the clients for processing demands and offers them a smaller software installation footprint. All the clients need in order to issue XKMS requests are a basic Web Services library for the programming language in use.

A demonstrator XKMS server has been developed as a part of the XKMS study in this report, and experiences with this server are presented.

PKI in mobile system raises concern since the PKI model depends on a reliable network connection with relatively high bandwidth to the PKI repository. In general, a node with no connection to a PKI will have to validate a certificate on locally stored information, information which is potentially out of date. The report outlines an arrangement of local XKMS servers which may employ any distribution or forwarding technique in order to disseminate certificates and revocation information. The arrangement of XKMS servers in a mobile network is a topic recommended for further study.

References

[1] *Digital Certificates and PKI version 2.0*. Norwegian National Security Authority (NSM), 2007. http://www.nsm.stat.no/.

[2] Andre Arnes, Mike Just, Svein J. Knapskog, Steve Lloyd, and Henk Meijer. Selecting revocation solutions for PKI. In *Proceedings of the 5th Nordic Workshop on Secure IT Systems*, Reykjavik, Iceland, 2000.

[3] Mark Bartel, John Boyer, Barb Fox, Brian LaMacchia, and Ed Simon. *XML-Signature Syntax and Processing*. W3C, 2002. http://www.w3.org/TR/xmldsig-core/.

[4] Bruce Christianson and William S. Harbison. Why isn't trust transitive? In *Proceedings of the International Workshop on Security Protocols*, pages 171–176, London, UK, 1996. Springer-Verlag.

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

[6] Peter Gutmann. Everything you never wanted to know about PKI but were forced to find out, 2002. http://www.cs.auckland.ac.nz/ pgut001/pubs/pkitutorial.pdf.

[7] Peter Gutmann. PKI: It's not dead, just resting. *Computer*, 35(8):41–49, 2002.

[8] Phillip Hallam-Baker and Shivaram H. Mysore. *XML Key Management Specification (XKMS 2.0)*. W3C, 2005. http://www.w3.org/TR/xkms2/.

[9] R. Housley, W. Polk, W. Ford, and D. Solo. RFC 3280 - Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, 2002.

[10] Audun Josang and Simon Pope. Semantic constraints for trust transitivity. In *APCCM '05: Proceedings of the 2nd Asia-Pacific conference on Conceptual modelling*, pages 59–68, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc.

[11] S. Kille, M. Wahl, A. Grimstad, R. Huber, and S. Sataluri. RFC 2247 - Using Domains in LDAP/X.500 Distinguished Names, 1998.

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

[13] D. Pinkas and R. Housley. RFC 3379 - Delegated Path Validation and Delegated Path Discovery Protocol Requirements, 2002.

[14] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[15] Ronald L. Rivest. Can we eliminate certificate revocations lists? In *Financial Cryptography*, pages 178–183, 1998.

[16] Ken Stillson. *PKI Interoperability - Tools and Concepts*. Mitretek Systems, 2007. http://www.noblis.org/Publications/Stillson07.pdf.

[17] M. Wahl, T. Howes, and S. Kille. RFC 2251 - Lightweight Directory Access Protocol (v3), 1997.