Directory Based Registration in Public Key Infrastructures

M. Lippert, E. Karatsiolis, A. Wiesmaier, J. Buchmann
Technische Universität Darmstadt, Fachbereich Informatik
Hochschulstr. 10, 64289 Darmstadt, Germany
{mal, wiesmaie, karatsio, buchmann}@cdc.informatik.tu-darmstadt.de

Abstract. Certificate life-cycle management operations, especially the initial registration of users, are important but expensive tasks in Public Key Infrastructures. In this paper we propose a mechanism that integrates these tasks with established processes, technology and data. Our approach is based on directory services that are usually already employed by companies, public administration and similar organizations for non-PKI purposes. It spares additional personell and is less error-prone, since the utilized processes are already set up and known to administrative personell and users. This reduces the costs to bootstrap and operate a Public Key Infrastructure. We show a case study about the proposed mechanism that was conducted at the Technische Universität Darmstadt in Germany in order to supply 20,000 students with certificates and keys.

Keywords. LDAP, Trustcenter, PKI, Registration, X.509

1. Introduction

Public Key Infrastructure (PKI) is an effective technology for securing digital communication in large networks like an organization's intranet or the internet. It provides the services of authentication and non-repudiation as well as confidentiality. Many applications already rely on these services like secure e-mail based on S/MIME, secure access to web sites using SSL/TLS, software updates in modern operating systems, and home banking.

Many companies and similar organizations like universities already employ directory services for a variety of non-PKI purposes. These are for example electronic address books, support of network-wide access control and maintainance of information relevant for network-administration. They usually contain data about employees, computers and digital services within the organization's environment. Moreover, the procedures how the data gets into the directory as well as their life-cycle management are already established and well known to both administrators and users. These procedures are already set up in a way that satisfies the organization's needs for security and usability. The directory can be assumed to be complete and up-to-date. Users are able to authenticate to the directory. The di-

rectories usually provide an interface that conforms to the Lightweight Directory Access Protocol (LDAP).

PKIs employ directory services as well. In this context they are used for publishing certificates and certificate revocation information. In [1] we have utilized LDAP directories to support two important PKI functions. One is to provide a proof-of-possession for keys that can be used only for encryption. This is done by placing the certificate encrypted on the directory and only the legitimate user (the one who possesses the private key) can decrypt it and place it in the directory. This task is associated to the registration of an entity in the PKI. The other is the delivery of software personal security environments. These are usually encoded according to PKCS#12 [2]. This is achieved with proper use of the access control mechanisms that are available in the LDAP directories. The paper at hand goes one step further. The whole certificate life-cycle management processes are based on a directory. The proof-of-possession scheme is one of the functions needed to implement the registration. We did not employ the delivery scheme since in our PKI we use smart cards. We therefore show how to support a hardware token based PKI.

In [3] a case study is provided where the registration is supported by an Active Directory Server. In contrast to our approach, this server is specially set up for this purpose. It is supplied with user information from other directory servers that were present in advance. Other certificate life-cycle management functions are not discussed.

Gutman describes in [4] the PKIBoot protocol which is in many respects similar to our approach. The protocol is intended to bootstrap network components and services in a PKI similar to the concept of Domain Name Services (DNS). In contrast to our approach it relays on the Certificate Management Protocol (CMP) [5] which is rarely used and comes with the flaws described in [4]. Instead our approach is based on LDAP. This is straight forward as common PKI enabled applications already implement the LDAP protocol. They have to be adapted only slightly to use the proposed mechanism.

In this paper we describe the mechanism to utilize existing LDAP directories in order to bootstrap and operate a PKI efficiently and automatically. This saves time and money. The basic idea is to reuse information that is already available in the directory for the registration. Furthermore, it employs the directory as a means to formulate requests for certificate life-cycle operations. These are initial certificate requests including registration, certificate revocation requests, and certificate renewal requests. Utilizing the LDAP's access control mechanisms allows for sophisticated PKI-management strategies as e.g. distributed administrations and group responsibilities. As directories are accessible from all over the network, this also applies for the PKI management. The progress of each operation is reflected by the LDAP and thus visible to trust center operators as well as to entities. The mechanism can further be combined with the proof-of-possesion described in [1].

The paper is organized as follows:

Section 2 gives a short introduction to LDAP directories. It provides details about their common usage and structure. It shows how their structure can be altered dynamically to adapt to new requirements. This is a basic feature for

our approach. Section 3 identifies the functions of the Registration Authority (RA) that will be covered by our mechanism. In Section 4 we describe the initial scenario and assumptions. Section 5 defines the new mechanisms. We show how to bootstrap and maintain the PKI in the described scenario. We therefore show how the certificate life-cycle operations are supported by the proposed mechanism. It is divided into three subsections, one per each request (certification, certificate revocation and certificate renewal). The security of our approach is discussed in Section 6. Section 7 shows a case study of the approach conducted at the Technische Universität Darmstadt. In Section 8 we conclude our work and give an outlook on further developments.

2. LDAP Directories

LDAP directories are a common means to organize all objects in a network. This includes hardware components (e.g. computers, network switches, etc.), services (e.g. web servers, virtual private networks) as well as the users. All these objects are modelled as entries of a hierarchically organized Directory Information Tree (DIT). Each entry can be uniquely described by the path leading from it to the tree's root. This path is called the entry's Distinguished Name (DN).

Each entry consists of a set of attributes. In LDAP directories[1] the type of an entry is described by special attributes called `objectClass`. Object classes define which attributes the entry is able to hold, their type and whether they are mandatory or optional. Object classes may be structural, abstract or auxiliary. Structural object classes define the principal nature of an entry. Abstract object classes are meant for defining common aspects of different structural classes. This is done using the concept of inheritance. Abstract classes are not allowed to be used in an entry without an inheriting structural class. Auxiliary classes specialize an entry by adding additional attributes. An entry can take zero or more auxiliary classes. The set of supported object classes define the directory's schema. Directories should always be operated with schema checking turned on. This enforces that the content of each entry always complies to the defined structure.

Directories already provide various security features. These are server authentication, confidentiality of the transmitted data, user authentication, and access control. The first two features are achieved by having the communication wrapped into an SSL/TLS protocol. Thus, they are based on PKI technology. Users authenticate themselves by means of passwords that are related to their directory entries. This is referred to as *binding* to an entry. The passphrase can hereby either travel in clear (Simple Authentication) or as a salted and iteratively computed hash value using Simple Authentication and Security Layer [7]. In the usual setup, directories use cleartext passwords over an SSL/TLS secured communication path. Access control is provided by means of access control lists (ACLs). They assign access permissions to read, modify, create or delete entries and/or attributes. The ACLs distinguish between anonymous and authenticated users.

[1] More specificly, we refer to LDAP version 3 [6]. This is the recommended version to use with PKIs.

Table 1. Object classes commonly used for modelling persons in LDAP directories. (S) denotes that the class is structural, (A) that it is auxiliary.

Object Class	Description
`country` (S)	Describes a country.
`organization` (S)	Describes an organization.
`organizationalUnit` (S)	Describes an organizational unit like departments or working groups.
`dcObject` (A)	Adds the concept of domain components for creating entries based on domain names.
`person` (S)	Describes a person (e.g. name, phone number, userPassword).
`organizationalPerson` (S)	Derived from `person`, adds information as postal address and room number.
`inetOrgPerson` (S)	Derived from `organizationalPerson`, adds information usable in the internet like emailAddress, optionally allows certificates.
`pkiUser` (A)	Optionally adds certificates to an entry.
`strongAuthenticationUser` (A)	Mandates a certificate, necessary for PKI based binding to the directory.
`posixAccount` (A)	Describes a UNIX account (UserID, password hash). Used for LDAP based login.

For authenticated users it is further differentiated between actions that concern their own entry[2] or those that concern other entries.

The following fact is important for our approach. The type of an entry can be changed dynamically. This is done by adding new, commonly auxiliary, object classes. If the attributes of an added object class are either optional or reasonable default values exist, the schema change can be done at once before their first usage. Otherwise the type of an entry has to be changed on the fly, when the respective information becomes available. This would require additional checking during regular operations, but still is feasible.

Table 1 lists object classes that are commonly used for modelling persons in an LDAP directory. The first three classes may be used to make up the path in the DIT to the user entries. They usually mirror the structure of the organization and/or the driven applications. The fourth class adds information that can be used to build internet domain names. This is often used in LDAP for the root entry of the tree. The class `person` models a person. It can be specialized to `organizationalPerson` and further to `inetOrgPerson`. The last one optionally takes a certificate for the user. This can also be achieved by adding the auxiliary class `pkiUser` (optional certificate) or `strongAuthenticationUser` (mandatory certificate), respectively. The class `strongAuthenticationUser` provides strong authentication when binding to the entry. The class `posixAccount` allows to utilize LDAP directories for computer login.

[2]The one they did bind to.

3. Certificate Life-cycle Management

Registration is the first step for an entity to become member of a PKI. It is usually the most expensive one. This is because no trust relationship exists between the entities and the trust center prior to their registration. Thus, the authenticity of the registered data must be achieved by out-of-band (mostly manual) means. Registration should meet the following requirements:

1. Identification of the requesting entity.
2. Establishment of data that describes the entity.
3. Enforcement of the certification policy by
   (a) verifying that the entity is allowed to receive a certificate, and
   (b) having the entity to accept the PKI's policy.
4. Generation of a unique distinguished name to be used in the certificate.
5. Linking the established registration data to the certificates produced for the entity. This is necessary to
   (a) detect naming collisions,
   (b) be able to resolve a certificate to the identity of its owner, and
   (c) monitor which certificates belong to which entity.[3]

To be able to undoubtly link certificates to their owners, each operation that modifies the certificate status must be visible to the RA. This can be achieved by having it act as an interface where entities request all certificate life-cycle operations. The RA then passes appropriate requests to a Certification Authority (CA).

For the remainder we assume the following structure of the trust center: It consists of a RA, a Certificate Management Authority (CMA) and a CA. The RA acts as in interface for the entities. All certificate life-cycle operations can be requested there. The CMA is responsible for the postprocessing of the requests. This is mainly the dissemination and publication of certificates and certificate revocation lists. The CA is encapsulated by the two others. It is therefore transparent to the entities. This makes it easier to protect it, since it only communicates with known entities and may be kept offline.

4. Initial Assumption

We now state the assumptions our approach relys on. They have been motivated in the introduction.

For the following we assume that an LDAP directory is already set up in an organization's intranet. It serves as an address book and for computer login. It contains one entry per employee. Each entry holds sufficient data to describe the identity of the respective employee. The entries have the object classes `inetOrgPerson` and `posixAccount`.

[3] In cases where the distinguished name in the certificate is not derived from the entity's identity, this is not trivial. Such certificates are called pseudonymous certificates.

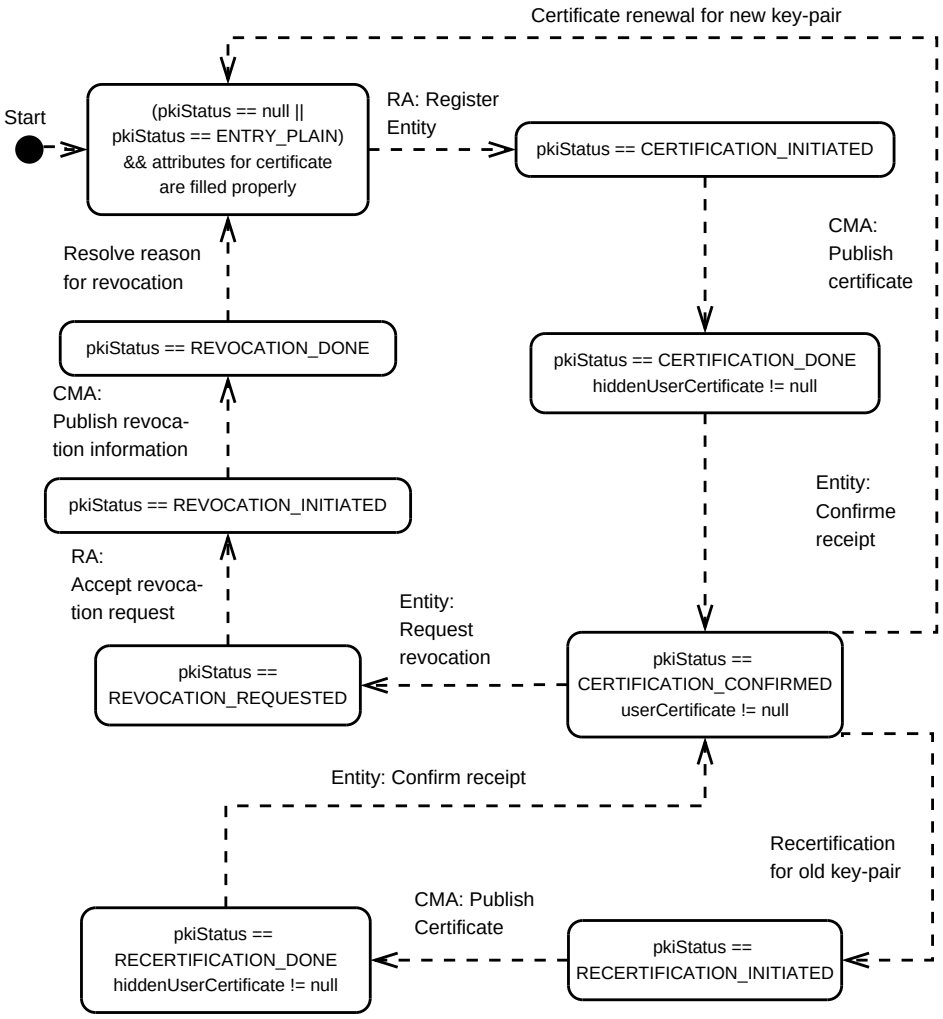

Figure 1. Statemachine of the Registration/Certification Process.

The processes that establish and maintain the directory contents are already available and sufficient with respect to the organization's security needs.

Furthermore, we assume that smart cards are employed for storing the employee's private keys and the respective certificates. The cards are pre-personalized. I.e. each card already contains a key-pair, but no user information has been put on it, yet.

5. Mechanism

We now describe the mechanism to reuse the available information and to automate the functions of the RA.

The mechanism can be described as a finite state machine. The state machine is depicted in Figure 1. The state is kept per entity in the directory. For this reason we introduce the new optional integer attribute `pkiStatus`. It is part of the auxiliary object class `pkiManagement`[4] which can easily be added to a directory. The attribute is of type integer to make comparison less costly and thus allow fast search operations. For this reason, it should be indexed by the directory as well. The RA scans the directory in regular intervals and reacts on the status of the respective entries. Also the entities' client software reacts on status changes of this attribute. In contrast to the RA, the client software only has to check the entity's own entry, not the whole tree.

5.1. Requests for Certification

The first operation for each entity is the registration. It spans the process from the identification of the entity to the initial certificate being issued. It is triggered for an entity, if the respective entry is in the following state:

1. The attribute `pkiStatus` does not exist or contains the value `ENTRY_PLAIN`.
2. All attributes that are used for certificate creation are set. This may include
   - the common names ('CN'),
   - the e-mail address ('mail'),
   - the organizational unit ('OU'), and
   - the user-ID

Thus, the requirements 1, 2, 3a, and 3b of Section 3 are achieved by the processes that lead to this initial state. We have assumed them to be sufficiently secure and already established.

The first condition offers a choice. The established processes may be altered to set the newly introduced `pkiStatus` to `ENTRY_PLAIN`. This allows faster scanning of the directory by the RA because the queries become simpler. On the other hand, processes need to be adjusted what may sometimes be unwanted or even impossible.

The RA now generates a certification request for the CA. This request consists of the to-be-signed part of an X.509 certificate (see [8]) and optional operational parameters. Each attribute of the certificate is either derived from the entry's attributes, from constant values defined by the certification policy or from a mixture of both. We take the entry's distinguished name as a meta attribute to be able to use it in requests. After posting the request to the CA, the RA sets `pkiStatus` to `CERTIFICATION_INITIATED`. This indicates that the certification request is on the way. The described mapping achieves the requirements 4 and 5 of Section 3.

We give an example for a mapping from an entry to a to-be-signed certificate. John Doe is a student. He is registered for the department of Computer Science. His directory entry is depicted in Figure 2. It contains his full name 'John Doe' (`CN`), his network account id 'jdoe' (`uid`), the name of the department 'Computer

[4]The definitions of the object class and its attributes are given in the appendix.

```
dn: CN=John Doe, OU=CS, ...
givenName: John
sn: Doe
cn: John Doe
ou: Computer Science
uid: jdoe
mail: jdoe@cs.tu-darmstadt.de
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
```

Figure 2. Example LDAP entry for John Doe

Science' (`OU`), and his e-mail address 'jdoe@cs.university.de' (`mail`). We assume the distinguished name (DN) of the object to be `CN=jdoe, OU=CS, OU=student, OU=addresses`. The RA now generates a certification request based on these values. It is depicted in Figure 3. The `CN` and `OU` attributes are copied to the respective components of the certificate's subject name. The information that John Doe is a student is taken from the entry's DN (`OU=student`). This affects the certificate's subject DN as well as the certificate policy set in the respective extension. For better readability the attribute `OU=student` is inserted right after the `CN` in the subject name. The suffix of the DN `O=university, C=DE` is provided as a constant as demanded by the policy. The attribute `notBefore` is set to the current time, `notAfter` is set to the end of the semester term which is also supplied by the policy as a constant. The e-mail address is added as `rfc822Name` to the `subjectAlternativeName`. Also the entry's DN is added as `directoryName` to the `subjectAlternativeName` to provide a strict binding between the certificate and the entry. This is important for the security of the revocation and further discussed in Section 6.

Especially the mappings that make up the certificate's distinguished name have to be chosen with care. It must be assured that the result is a unique name within the PKI. Of course, the RA is able to detect name collissions since it can distinguish entities by their entry's distinguished names. But resolving the collissions would mean manual processing and this is what we want to avoid.

In the following, the request is augmented with the entry's DN as an operational parameter and routed to the CA that generates the certificate. The certificate is then forwarded to the CMA. This authority puts the certificate on the LDAP into the attribute `hiddenUserCertificate`. It is not published, i.e. written to `userCertificate` at this stage, because the certification process has not yet finished and thus no one should already use the certificate. However, the CMA finishes this step by setting `pkiStatus` to `CERTIFICATION_DONE`.

The entity can now download the certificate from the directory using the client software, store it on the smart card and write it back to the directory into the attribute `userCertificate`. The user is now able to use the PKI's services. In parallel all other participants of the PKI can see and use the certificate. This approach assures that the certificate cannot be used before the enrollment of the entity has finished. It can be enhanced to a proof-of-possession as described in

```
...
validity:
notBefore: 05-05-2005
notAfter: 06-06-2006
subject:
CN=jdoe, OU=student, OU=Computer
Science, O=university, C=DE
...
extensions:
subjectAltName:
rfc822Name: jdoe@cs.university.de
directoryName: CN=jdoe, OU=CS,
OU=student, OU=addresses
CertificatePolicies:
policyIdentifier: idStudentPolicy
```

Figure 3. Example of the to-be-signed certificate for John Doe

[1], if the entity's key-pair can be used for encryption. In this case the certificate would not be put in the attribute `hiddenUserCertificate`, but encrypted with the contained public key and written to `userEncryptedCertificate`. The entity must then be able to decrypt the certificate, i.e. use the correct smart card, in order to have it published.

One thing is still left open for explanation. This is how the public key gets into the certificate request. To be consistent with our scheme, the public key should be put into the directory as well. We therefore introduce the attributes `userPublicKey`, `userPublicKeyAlgorithm` and `userPublicKeyParameters` that may contain the binary encoding of a public key as well as its algorithm ID and parameters. It should either be used, if the entity does not participate in the process in which the key is uploaded or if proof-of-possession is done. Otherwise the authenticity of the public key cannot be guaranteed. To be able to deal with signature-only keys, we have introduced a `userPKCS#10` attribute which replaces `userPublicKey` in this case.

Adding the public key requires to extend the workflow. There are in general two possibilities. Either the public keys are uploaded in a batch process by the trust center or by the entities themselves. In the first case the trust center needs a list that maps the identities of the entities to the public keys on the cards that they have received. This requires that the identity is recorded when the card is sent or handed out to the entity. In any case the authenticity of this mapping is important, since issuing a certificate means to guarantee that the entity owns the private key that corresponds to the respective public key. Having the entity upload the public key requires an additional communication step between the entity and the directory, one for uploading the key and one for fetching the certificate. In this case a proof-of-possession must be conducted. Moreover, the trust center must assure that the public key belongs to one of the cards that were provided to the entities. Otherwise it would not be able to give any guarantees about the quality of the keys and tokens which the keys are stored in. For this reason the RA should

obtain a list of valid public keys from the card vendor and match the public keys in certificate requests against this list.

Now all required information is in place and the client application changes `pkiStatus` to `CERTIFICATE_CONFIRMED`. This is done in order to signal to the trust center that the certificate has reached the intended recipient and the process has finished so far. In this process two factors are important for an authentic certificate request. The entity needs to know the passphrase for binding to the directory in order to access either `hiddenUserCertificate` or `userEncryptedCertificate` and to set `pkiStatus` and `userCertificate`. Furthermore, the entity must have the correct smart card, if proof-of-possession is done.

5.2. Requests for Certificate Revocation

The proposed mechanism supports certificate revocation in a very similar way. From the RA's point of view this is done by setting `pkiStatus` to `REVOCATION_REQUESTED`. This can be done by the entities for their own entry. They can bind to their LDAP entry by PKI means or by passphrase and change the value. This step should be supported by the client software. The status can also be changed by an operator of the trust center who was granted access to the entry by means of the directory's access control list.

When the RA recognizes the `pkiStatus` being set to `REVOCATION_REQUESTED`, it posts a revocation request to the CA. This request basically consists of an appropriate `revocationEntry` structure as used in a Certificate Revocation List [9]. The serial number is taken from the certificate of the entry. The revocation date is set to the current time. After posting the request, the RA sets the `pkiStatus` to `REVOCATION_INITIATED` to signal that a revocation is in progress. After the revocation information has been published, the CMA sets `pkiStatus` to `REVOCATION_CONFIRMED`. This signals that the certificate is revoked.

To get from this state to a new certificate will most likely involve manual processing, since the reason for the revocation needs to be considered and resolved.

5.3. Requests for Certificate Renewal

When certificates reach the end of their validity period, they new ones must be issued. The RA can detect this case automatically. The `pkiStatus` must be set to `CERTIFICATE_CONFIRMED`, a user certificate must be present and the current date must be close to `notAfter`. It is important to assure that no revocation request is ongoing for this entry. Otherwise newly issued certificate would overwrite the revocation.

If certificates expire, basically two options can be taken. Either the new certificate is issued for the old public key (recertification), or a new key-pair is generated for which a new certificate is issued (renewal). Which approach to choose, depends on the situation and must be defined by the trust center's policy. They are treated differently by our approach.

In case of a recertification the process differs from the initial certificate creation in that the old certificate can be used as template for issuing the new one and that no proof-of-posession is required. When the RA notices that a

recertification is necessary, it sets the `pkiStatus` of the respective entry to `RECERTIFICATION_INITIATED`. It then takes the old certificate from the directory, verifies the signature and that the `subjectAlternativeName` corresponds to the entry. It then takes the public key from the certificate and the other attributes from the directory as described in the initial certification to form a request for the CA. The certificate is produced and put into the attribute `hiddenUserCertificate`. To signal that the certificate has been produced, the RA sets `pkiStatus` to `RECERTIFICATION_DONE`. The entity can now update the smart card, publish the certificate and confirm the process by setting `pkiStatus` to `CERTIFICATE_CONFIRMED`. It is notable here that the old certificate will not be replaced, until the entity has updated his smart card. Up to this point the old certificate remains in place and operable.

In case of renewal the process is almost identical to the initial certification. A new smart card is handed out or sent to the entity. The respective public key is written to the directory either by the entity or the trust center and `pkiStatus` is set to `ENTRY_PLAIN`.

6. Security Considerations

The entire security of a PKI following our approach is based on the security features of the directory. This concerns both the effectiveness of the access control mechanisms and the processes which lead to the data that is stored in the directory. Controlling the directory means controlling the PKI. We now show, how the attributes of each entry have to be protected.

It is recommended that the processes that establish the data lead to authentic information in the directory. We have assumed that the directory is already in place and that these processes were set up properly. This mirrors the fact that identification and registration of the personell of companies, universities and similar organization is already implemented in a way that satisfies the security needs. Of course these processes have to be reviewed regarding their security properties before applying the proposed mechanism.

First of all, clients who anonymously bind to the directory should only be able to read the "regular" attributes of each entry as needed for the services offered to everyone. This may include the names and addresses, if the electronic addressbook should be publicly available, and the certificates in order to allow sending encrypted messanges. They must not be allowed to modify any entry. The attributes `hiddenUserCertificate`, `encryptedUserCertifcate`, `pkiStatus`, `userPublicKey` and `userPKCS#10` should not be visible to them to prevent that certificates are used before their enrollment has finished.

We now describe the access policy for those entities that authenticate themselves to the directory. We thereby distinguish between access to own entries, i.e. those they did bind to, and to other entries.

An entity must not be able to (unconditionally) manipulate those attributes of the entry that are used to make up the to-be-signed certificate. Otherwise the entity could forge the contents of the certificate. As will be seen in the case study, it is sometimes applicable that entities have influence on the value to which a

certain attribute is set. They may e.g. want to choose the e-mail address from a set of possible addresses. In any case, this access must be limited. This can be done by a server side application that temporarily grants access to the respective attributes while monitoring the values which are stored. It is not possible to achieve this with the access control mechanisms provided by LDAP.

The `pkiStatus` attribute must be writable for the entity to be able to launch certification and revocation requests. The same applies to the `userCertificate` as this is demanded by the certification mechanism. In case the public key is uploaded by the entity, write access for `userPublicKey` and `userPKCS#10` must be granted, respectively.

The RA also needs access to the entries of the entities. It needs write access to the attributes `pkiStatus`, `hiddenUserCertificate` and, respectively, `userEncryptedCertificate` to carry out its tasks. Furthermore, it needs to be able to read all attributes that are necessary for creating the certification request for the CA.

Our approach allows for manual administration of all or certain certificates. This is done by granting administrational personell write access to the `pkiStatus` attribute of other's entries. By this means administration can also be distributed or partitioned. The directory is accessible from all over the network. This allows for geographically distributed administration as well. An organization may for example have one employee per department responsible for managing the certificates requests of the coworkers. Also, a network administrator may be responsible for a group of servers. By having proper access to the respective directory entries, the administrator can issue certificate or revocation requests for them.

Special care has to be taken for the revocation mechanism. Since the entity has write access to the `userCertificate` attribute, it is important that the certificate is authentically linked to this entry. We suggest to establish this binding by adding the entry's distinguished name in the directory to the certificate's `subjectAlternativeName` and having the RA check this binding. Otherwise an entity could revoke arbitrary certificates of the PKI by writing the targeted certificate to the entry and setting `pkiStatus` to `REVOCATION_REQUESTED`. The RA would then take this certificate without any checks and produce a revocation request for the CA based on its serial number.

7. Case Study

We have implemented the proposed mechanism in a project at the Technische Universität Darmstadt. This project started in 2004 and was to establish a PKI that allows the students among other things to log on to the university's computers, to have secure access to the Virtual Private Network and protected web sites, to register for exams and get knowledge of the results. Thus the targeted security services were mainly authenticity and non-repudiation. The student's private key should be kept on secure smart cards. The main goal was to keep the management effort at a minimum both during the initial enrolment and the operation for a user group of 20,000 students. The PKI is for now implemented as a prototype. It is planned to be put into operation in October 2005.

We now provide more details about the PKI. In the initial situation the university already operated an X.500 directory. The directory is maintained by two administrators. It already contains all personal and administrative data that we need for issuing certificates. The data is recorded per student during immatriculation. It is regularly updated by the university's administration and can thus be considered to be correct and up-to-date. The students are provided a passphrase that allows them to authentically bind to the directory. They receive this passphrase per mail together with their student ID card. Upon receipt the students must bind to the directory and change the passphrase within a limited period of time. The student can then choose an e-mail address from a set of still available address that are based on combinations of first and last name. These services are provided as SSL/TLS secured web services. Changing the e-mail address is only possible through these web services. The students do not need or have direct write access to the respective LDAP attribute. It is not possible for them to change the e-mail address a second time.

An important fact for modelling the PKI was that the students are unknown to the university until they appear for immatriculation. Thus we are not able to do any preparations that need the knowledge of the student's identity. On the other hand, it is no option to have each student appear a second time for any management issues. Each additional minute counts times 20,000.

We installed a trust center software at the department for network administration. The software consists of three major modules, RA, CA and CMA. The CA is kept offline and exchanges requests and products with the two other components via mobile media. RA and CMA are installed in a protected area and communicate with the X.500 directory over an SSL/TLS secured channel.

We use smart cards for protecting the student's private keys. Each card is equipped with two chips. One is a crypto-processor for the PKI functionality. It contains the student's private key and the root certificate of the PKI as a trust anchor. The second one is a contactless chip for anonymous electronic payment services within the university. The cards are pre-personalized by the manufacturer. It is also responsible for sending the cards to the students per mail. It therefore receives a list that maps the addresses of the students to a pseudonym. The manufacturer in turn provides a mapping from the pseudonyms to the public keys of the cards that have been sent. This mapping is processed in a batch process to put the public keys into the corresponding entries in the directory. The keys on the card are protected by a null-PIN. I.e. the PIN is initially set to zeros and the card is put in a special state, that allows no operation but changing the PIN. The card leaves this state when the PIN is altered and it is impossible to get into this state again. By checking this state upon receipt of the card the student can be sure that no one has used it up to now. This is sufficient for non-repudiation and authentication purposes and spares the additional effort to securely exchange a PIN.

We chose to modify the existing web services to set the `pkiStatus` to `CERTIFICATION_INITIATED` when a student has ascertained the e-mail address. The modification of these services were only minor but significantly improve the queries conducted by the RA. Revocation is supported in two ways. Students are

able to revoke their certificate through a web service. Authorization is done by verifying that the students are able to bind to their entry.

We implemented a client application for the students. It allows them to fetch the certificates from the LDAP, store them on the smart card and write them back to the `userCertificate` attribute. The application supports proof-of-possession using the `userEncryptedCertificate` attribute. Since we use RSA keys, we employ this scheme for signature keys as well. The application is capable of checking the null-PIN status of the smart cards. Implementing a dedicated application was less time consuming then integrating the functionality in all PKI-enabled applications that the students are going to use (e.g. e-mail client, web browser, VPN client).

We did benefit from this approach in the following ways:

- The directory was already set up and contained all necessary information. No additional registration step and no manual processing was needed. Thus time and money was saved.
- The processes which deal with the information stored in the LDAP could remain nearly unchanged. No additional personell was necessary and no additional processes had to be established. The whole PKI is maintained by the two operators, that have also been responsible for the directory server.
- The maintainance of the directory was already known to the operators. This knowledge is sufficient for also maintaining the PKI. Therefore, no new skills were required.

Besides the smart card based infrastructure for the students, a second one for the SSL/TLS servers was established. This PKI is already working. It keeps the private keys in software and utilizes PKCS#10 [10] messages to request certificates. It also uses the proposed mechanism with some adaptations.

8. Conclusion and Future Work

We have shown a mechanism to reuse information that is already available in a directory. The mechanism allows to bootstrap a PKI and carry out certificate lifecycle management operations on basis of a directory service. Distributed management and administration of groups is possible. The mechanism can be used with smart card based PKIs. It is modelled as a finite state machine and it is shown that it suffices the requirements for the respective management operations. The applicability of the mechanism is shown in a case study.

The next steps will be to consider variations of this mechanism. It has already proven to be combinable with keys held in software (e.g. conformant to PKCS#12). A future improvement will be to allow multiple certificates per entity. Therefore, each certificate must have its own `pkiStatus` attribute. This can be done by storing each certificate in an entry that is subordinate to the entity's entry in the DIT. A major improvement will be to have the RA also react on changes to the values of an entry's attributes or on whole entries being moved to a different position in the DIT. The vision is here that a directory administrator, if e.g. an employee changes the organizational unit, simply moves the entry to a

new branch in the directory. The RA notices this and triggers the required processes to revoke the old and issue a new certificate. This is work in progress. Our approach may further be combined with the Client Update Protocol for LDAP that was proposed by the IETF. Employing this protocol the RA could avoid polling data from the directory. Instead the RA would be noticed upon changes. The protocol is described in [11].

A. Object Class and Attribute Definitions

Here are the definitions of the object class and its attributes that have been introduced in this paper.

```
# object class for pki management
objectclass   ( 1.3.6.1.4.1.8301.3.2.2.1.10.1
    NAME 'pkiManagement'
    SUP top
    AUXILIARY
    MAY (
    hiddenUserCertificate $ pkiStatus $ userPKCS10 $ userPublicKey $
    userPublicKeyAlgorithm $ userPublicKeyParameters))

# blinded certificate
attributetype ( 1.3.6.1.4.1.8301.3.2.2.1.10.2
    NAME 'hiddenUserCertificate'
    EQUALITY certificateExactMatch
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.8
    SINGLE-VALUE )

# status of management operations
attributetype ( 1.3.6.1.4.1.8301.3.2.2.1.10.3
    NAME 'pkiStatus'
    EQUALITY
    integerMatch
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
    SINGLE-VALUE )

# for uploading a public key with a proof-of-possession
attributetype ( 1.3.6.1.4.1.8301.3.2.2.1.10.4
    NAME 'userPKCS10'
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.5
    SINGLE-VALUE )

# the public key to certify together with required
# parameters and algorithm ID
attributetype ( 1.3.6.1.4.1.8301.3.2.2.1.10.5
    NAME 'userPublicKey'
    EQUALITY octetStringMatch
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.40
    SINGLE-VALUE )
```

```
attributetype   ( 1.3.6.1.4.1.8301.3.2.2.1.10.6
    NAME 'userPublicKeyAlgorithm'
    EQUALITY objectIdentifierMatch
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.38
    SINGLE-VALUE )

attributetype ( 1.3.6.1.4.1.8301.3.2.2.1.10.7
    NAME 'userPublicKeyParameters'
    EQUALITY octetStringMatch
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.40
    SINGLE-VALUE )

# encrypted certificate for proof-of-possession scheme taken from [1]
attributetype ( 1.3.6.1.4.1.8301.3.2.2.1.8
    NAME 'userEncryptedCertificate'
    EQUALITY octetStringMatch
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.40
    SINGLE-VALUE )
```

References

[1] V. Karatsiolis, M. Lippert, and A. Wiesmaier. Using LDAP Directories for Management of PKI Processes. In *Proceedings of Public Key Infrastructure: First European PKI Workshop: Research and Applications, EuroPKI 2004*, volume 3093 of *Lecture Notes in Computer Science*, pages 126–134, June 2004.

[2] RSA. PKCS#12 v1.0: Personal Information Exchange Syntax Standard, June 1999. http://www.rsasecurity.com/rsalabs (24 Jun. 1999).

[3] R. Guida, R. Stahl, T. Bunt, G. Secrest, and J. Moorcones. Deploying and Using Public Key Technology: Lessons Learned in Real Life. *IEEE Security & Privacy*, 2(4):67–71, 2004.

[4] P. Gutmann. Plug-and-Play PKI: A PKI your Mother can Use. In *Proceedings of the 12th USENIX Security Symposium*, pages 45–58, 2003.

[5] C. Adams and S. Farell. Internet X.509 Public Key Infrastructure Certificate Management Protocols. *IETF Request For Comments*, 2510, March 1999.

[6] J. Hodges and R. Morgan. Lightweight Directory Access Protocol (v3): Technical Specification. *IETF Request For Comments*, 3377, September 2002.

[7] J. Myers. Simple Authentication and Security Layer (SASL). *IETF Request For Comments*, 2222, October 1997.

[8] Recommendation X.509 ITU-T. Information Technology - Open Systems Interconnection - The Directory: Authentication Framework. August 1997.

[9] R. Housley, W. Polk, W. Ford, and D. Solo. Internet X.509 Public Key Infrastructure certificate and Certificate Revocation List (CRL) Profile. *IETF Request For Comments*, 3280, April 2002.

[10] RSA. PKCS#10 v1.7: Certification Request Syntax Standard, May 2000. http://www.rsasecurity.com/rsalabs (20 May. 2000).

[11] R. Megginson, M. Smith, O. Natkovich, and J. Parham. Lightweight Directory Access Protocol (LDAP) Client Update Protocol (LCUP). *IETF Request For Comments*, 3928, October 2004.