

Life-Cycle Management of X.509 Certificates Based on LDAP Directories

M. Lippert, V. Karatsiolis, A. Wiesmaier, J. Buchmann
Technische Universität Darmstadt, Fachbereich Informatik
Hochschulstr. 10, 64289 Darmstadt, Germany
{mal, karatsio, wiesmaie, buchmann}@cdc.informatik.tu-darmstadt.de
Tel. +49 (6151) 16-6168

Abstract. Companies and organizations employ PKI technology to secure the communication in their intranets and over the internet. The services of authentication, non-repudiation, confidentiality and the transport of authorization information are often supported by X.509 certificates. The synchronization of the certificates' life-cycle with the management of the PKI users is a common problem. We propose a mechanism to achieve this synchronization based on directory services. This enables to transparently update the information provided by the PKI and offers a high potential for automation. The mechanism spares personnel and is less error-prone, since it relies on processes and data that are already established. It reduces the costs to bootstrap and operate the infrastructure. We show a case study on 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, Trust Center, PKI, Registration, X.509, Life-cycle Management, Automation

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 supports the services of authentication, non-repudiation as well as confidentiality. The most common means for transporting the respective information are X.509 certificates [1], [2]. Many applications rely on these services like secure e-mail based on S/MIME [3], secure access to web sites using SSL/TLS [4] and software updates in modern operating systems. Additionally, PKI-aware applications calculate authorization information on basis of certificates' contents. X.509 certificates therefore allow containing arbitrary extensions to express respective information. However, each piece of data that is contained in a certificate may have its own life-cycle. The processes that determine these life-cycles are often already set up independently of the PKI. New employees in an organization need a certificate in order to use certain services and to represent their affiliation. If they move to another department their certificates need to be updated. When

they leave the organization, further access to services must be denied. Servers may be installed, removed or renamed, authorizations may be granted or revoked. In order to preserve the expressiveness and validity of the PKI, the certificate life-cycle management needs to synchronize with these processes. Reducing the need for manual intervention to a minimum allows for better control of the complexity and costs of the PKI and avoids mistakes.

Two factors determine the synchronicity of the certified data. Firstly, certificates are issued for a fixed validity period which is (or should be) determined by the time span for which the data is expected to remain valid. This is not always possible. Secondly, certificates can be revoked when their content becomes invalid before they expire. This is done by providing additional revocation status information in an authentic way within the PKI. Basically two approaches exist. In the more usual setting, certificates are issued for a rather long validity period (usually 6 to 36 months). Consequently, revocations may occur due to unpredictable changes of the certified data and must be supported. The revocation rate in such scenarios is usually expected to be 10%, as described by Årnes et al. in [5]. The second approach tries to minimize the need for revocations or to avoid it at all. This is done with short-lived certificates whose validity period typically is only several hours. As a consequence, the renewal of certificates becomes a major issue putting additional emphasis on the role of the entity registration, as shown by Hsu et al. in [6]. This approach cannot be applied to all applications. In case of long-lived certificates, the detection of changes in the certified data as well as the (early) certificate renewal should be automated. The application of short-lived certificates requires the automation of certificate renewals.

Directory servers are widely employed for a variety of non-PKI purposes. These are for example electronic address books, support of network-wide access control and maintenance of information relevant for network-administration. They usually contain data about employees, computers and digital services within the organization's environment. Moreover, the procedures about how the data gets into the directory as well as its 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 directories usually provide an interface that conforms to the Lightweight Directory Access Protocol (LDAP) [7].

PKIs employ directory services as well. In this context they are used for publishing certificates and certificate revocation information. We show in [8] how LDAP directories can 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 in the directory, and only the legitimate user (the one who possesses the private key) can decrypt and publish it in the directory. The other is the delivery of software personal security environments. These are usually encoded according to PKCS#12 [9]. This is achieved with proper use of the access control mechanisms that are available in the LDAP directories. The article at hand goes further. The whole certificate life-cycle management processes are based on a directory. The schemes for proof-of-possession and token delivery serve as building blocks for the proposed mechanism.

In [10] a case study is provided where the registration is supported by an Active Directory Server. In contrast to our approach, this server is set up especially 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.

Gutmann describes in [11] 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 is restricted to networked devices and services. It employs the Certificate Management Protocol (CMP) [12] to request and obtain certificates. Because of the complexity of CMP and several problems that Gutmann identifies in [11] only a limited and reinterpreted subset of the protocol is used. In contrast, our approach is based on LDAP. This is straightforward as common PKI enabled applications already implement the LDAP protocol. They have to be adapted only slightly to use the proposed mechanism.

In this article we describe the mechanism to utilize existing LDAP directories in order to bootstrap and operate a PKI efficiently and automatically. This saves time as well as resources. The basic idea is to reuse information that is already available in a directory for registration purposes. Furthermore, it employs the directory as a means to formulate requests for certificate life-cycle functions. 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 administration 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 it is visible to trust center operators as well as to entities. The mechanism can further be combined with the proof-of-possession described in [8]. This work has first been presented at the 4th International Workshop for applied PKI (IWAP'05) [13].

This article 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 certificate life-cycle functions that are covered by our mechanism. Section 4 defines the new mechanism. We show how to bootstrap and maintain the PKI in the described scenario. We therefore show how the certificate life-cycle functions are supported by the proposed mechanism. The security of our approach is discussed in Section 5. Section 6 shows a case study of the approach conducted at the Technische Universität Darmstadt. In Section 7 we conclude our work and give an outlook on further developments.

2. LDAP Directories

LDAP directories are a common means to organize all kinds of objects in a network. This includes hardware components (e.g. computers, network switches), services (e.g. web servers, virtual private networks) as well as users. All these

objects are modeled 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 directories¹ the type of an entry is described by the values of the special attribute `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 by 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 enabled schema checking. This enforces that the content of each entry always complies with the defined structure.

LDAP 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 (SASL) [14]. 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. For authenticated users it is further differentiated between actions that concern their own entry² 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, this 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 is feasible but requires additional checking during regular operations.

Table 1 lists object classes that are commonly used for modeling 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

¹More specifically, we refer to LDAP version 3 [7]. This is the recommended version to use with PKIs.

²The one they did bind to.

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 utilizing LDAP directories for computer login.

3. Certificate Life-cycle Management

We now analyze the certificate life-cycle management functions. These are registration, certificate and/or key renewal and revocation. The goal of this chapter is to analyze how they combine with respect to the life-cycle of entities in an organization. By this we determine which functions can be carried out automatically to react on life-cycle changes of an entity. This reduces maintenance costs. These functions and respective combinations will then be modeled by our approach. Furthermore we show their properties and security implications.

Registration is the first step for an entity to become member of a PKI. Its goal is to establish the data on whose basis a certificate can be issued. We will subsequently call it *registration data*. Registration should meet the following requirements:

1. Identification of the requesting entity.
2. Establishment of data that describes the entity.
3. Establishment of additional information to be included in the certificate.
4. Enforcement of the certificate policy by
 - verifying that the entity is allowed to receive a certificate, and
 - having the entity to accept the PKI's policy.
5. Generation of a unique distinguished name to be used in the certificate.
6. Key management. There are two alternatives: Firstly, the establishment of the public key, in case it was generated by the entity. Secondly, the establishment of a protected channel through which the entity will later on receive the private key.
7. Linking the *registration data* to the respective certificate in order to
 - detect naming collisions,
 - be able to resolve a certificate to the identity of its owner,³ and
 - monitor which entities belong to which certificates.

Registration is the most expensive step, because no trust relationship exists between entities and the trust center beforehand. Thus, the authenticity of the *registration data* must be achieved by out-of-band (often manual) means. Respective processes are usually already set up within an organization. This is especially the case for attributes like authorization information. With this, the establishment and the freshness of *registration data* does highly depend on external processes and automation can be achieved by synchronizing these processes with the life-cycle management.

³In 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.

The quality of the registration has several implications to the PKI's security. Improper identification may enable an adversary to impersonate other entities. The same applies to the establishment of unique digital names. As a certificate does only bind the public key to a digital name, the PKI's security services also depend on the ability of the trust center to resolve this name to the respective entity. The freshness property is important to guarantee that the certificate represents proper information. Improper information may either lead to denial or false permission of services.

The trust center also needs to keep track of changes to the *registration data* after the certificate was issued. If the data becomes invalid before the certificate expires, it should be revoked and, if appropriate, replaced by a new one.⁴

A revocation may be requested by an entity, by the trust center or emerge from the organization's workflow. Proper authorization is necessary to prevent denial of service attacks on the PKI.

To be able to plan subsequent processing the trust center must know the reason that led to the revocation. It determines whether the revocation is terminal or the certification can be extended, and whether the key must be renewed. In case the reason is not clear at the time of the revocation request, it must be examined afterwards. This is usually a manual process and should be avoided as it potentially extends the time span in which no certificate is available for an entity. While revocations that are requested by the entity itself fall into this category, external processes that lead to a revocation can usually provide sufficient information.

The trust center can extend the certification, if the old certificate expires or is revoked for a reason that permits it. Again, it is a prerequisite that it has kept track of the validity of the *registration data*. Two processes have to be distinguished. If the key-pair is not concerned, it is sufficient to issue a new certificate for the existing public key. This function is called *certificate renewal*. Otherwise, the entity and the trust center have to establish a new key as well. This function is called *key renewal*. The key exchange makes it significantly more complex and time consuming. Some possibilities for the key management will be discussed in Paragraph 3.1.

Another critical point, besides the optional key exchange, is the update of the certificate at the entities' side and in the directory. The goal is to have no gap, in which the entity does not have a valid certificate. Expiration is a scheduled event. Thus, the new certificate can be issued with a validity period that overlaps the old one. This intersection allows the entity to fetch and acknowledge the new certificate and initiate its publication. In case of a key renewal this is even more important, as an additional step of interaction may be required, in which the trust center negotiates key information with the entity. In case of a revocation, this gap can only be minimized. This is best achieved by automating the certificate and key renewal.

Summarizing the above analysis, the following life-cycle management functions and combinations have to be modeled:

⁴Depending on the security needs and the certificate policy this may be handled with a certain tolerance. E.g. a revocation may not be done, if the certificate expires soon.

1. Registration and initial certification.
2. Scheduled certificate or key renewal (after expiration).
3. Revocation that allows for manual intervention (unknown reason).
4. Explicit termination (revocation without subsequent renewal).
5. Early renewal (after revocation).

Furthermore, the trust center needs to continuously keep track of the validity of the *registration data*.

3.1. Management of Keys

During the initial certification and each subsequent key renewal, key material must be exchanged between the entity and the trust center. The private key can be protected by a variety of means ranging from software based formats like PKCS#12 to cryptographic hardware like smart cards or built in hardware security modules. Each means of protection puts different requirements on how the public keys are exchanged and how they are associated with the respective entity. We discuss the following scenarios:

1. Pre-personalized smart cards are used. They already contain a key pair that was generated by the manufacturer. The manufacturer provides the public keys as an authentic list or they are gathered when the cards are handed out. The trust center can then associate the keys with the respective entities. The certificates are loaded onto the card at the client side.
2. The entity generates the key-pair. Thus a proof-of-possession scheme needs to be applied and a certificate needs to be issued. The trust center must therefore be able to accept the respective data format, e.g. PKCS#10 [15], or perform an indirect proof-of-possession as described in [8].
3. The trust center generates the key-pair and ships it as a softtoken. The entity chooses a passphrase in order to protect the transport of the token and communicates it to the trust center over a secure channel. Then the trust center can generate the key-pair, protect it and send it electronically to the entity.

4. Mechanism

Before going into the very details of the proposed mechanism we want to provide a short motivation for the strategy we chose. A main goal of PKIs is to support the security of pre-existing communication. The idea for the proposed mechanism is based on several observations we made in “real-world” projects. Hence, it is the generalized essence of our case studies.

Client applications, like e-mail clients and web browsers, are usually already aware of PKI technology in the sense that they can deal with signed and encrypted data. We have experienced in several projects that such software does not easily integrate with certificate life-cycle management. In order to keep changes to the existing systems at a minimum, the basic idea was to adapt the PKI to the already established processes rather than the other way round. The goal of the proposed

mechanism is to provide an interface between existing processes and certificate life-cycle management that is easy to integrate, maintain and automate. Instead of employing message based communication for this interface, we model the life-cycle of each certificate as a finite state machine. Thus, the existing processes do not need to cope with the processing model of the PKI's trust center — it may be completely asynchronous. Moreover, this allows for expressing the state with minimal data. For representing these states, we use LDAP directories. They are widely employed independently of PKIs, and their data model can be extended without side effects.

We now describe our approach to implement the life-cycle management functions and their combinations that we have identified in Section 3.

We want to put as few requirements on the behavior of the trust center as possible. We assume that it consists of a Registration Authority (RA) that accepts the requests and a Certificate Management Authority (CMA) that updates the directory if new products like certificates have been generated. Both encapsulate the Certification Authority (CA) that actually signs the certificates and may generate key-pairs. It can be kept offline or provided by a third-party as a service.

We model the life-cycle states of a certificate as a finite state machine. It is depicted in Figure 1. The transitions between the states mark the changes to the life-cycle status. They are either triggered by the trust center (TC), the entity (E) or by an administrator (A). The dark-gray boxes mark final states with respect to the life-cycle management. The light gray boxes denote transient states that demand interaction with the entity. Those states on which the trust center reacts are white. The certificate life-cycle management functions and their combinations that have been identified in Section 3 correspond to the paths between the special states 'Start' and 'Stop' and the dark-gray states.

In the following we assume that an LDAP directory is already set up. It contains one entry per entity that needs to be certified. Each entry holds sufficient data to describe the entity's identity. The entities are able to securely bind to their entry. Furthermore, the processes that establish and maintain the entries are already available and sufficiently secure. This assumption has been motivated in Section 1. Moreover, we assume in the first place that each entity has at most one certificate at a time. In this case certificate and state information is kept in the entity's directory entry. We will later extend this to scenarios where the entities have more than one certificate for different purposes in parallel. The state information is represented by the newly added integer attribute `pkiStatus`. It is part of the auxiliary object class `pkiManagement`.⁵ The attribute is of type integer to make comparison less costly and allow faster search operations. To speed up searches further it should also be indexed by the directory. The trust center scans the directory in regular intervals and reacts on the state information. The entity also needs to react on state changes. While the trust center has to monitor the whole directory tree, the entity only cares for and has access to its own entry. The processing carried out by the entity should be integrated in its client software.

In order to implement the different key management modes described in Paragraph 3.1 we need to model additional information about the public keys. The

⁵The definitions of the object class and its attributes are given in Appendix A.

idea is to combine this key information with the *registration data* that is already in the directory on a per entity basis. When pre-personalized smart cards are used, only the public key needs to be associated. This is done by providing the attribute `userPublicKey` and respective parameter information as described in Appendix A. The public keys are stored in the entries when the cards are handed out, either during the initial rollout or if a broken or lost card needs to be replaced. The integrity of the key is protected by the directory's access control facilities. Further authentication information is not necessary. In case the user generates the key pair, also the public key needs to be uploaded. The format depends on the way the proof-of-possession is implemented. For a direct proof based on a PKCS#10 structure the respective entry `userPKCS10` is used. If the proof is done indirectly by encrypting the certificate, the field `userPublicKey` suffices. If the key-pair is generated by the trust center, the entity is required to set a passphrase. This is supported by the attribute `userPassphrase`. Depending on the security requirements the passphrase may be provided in clear or encrypted for the trust center, for example with a special public key.

We also need to provide attributes to transport the trust center's products back to the entity. Soft tokens can be put into the attribute `userPKCS12`. Certificates can be put into the attributes `hiddenUserCertificate` or, in case of indirect proof-of-possession, `userEncryptedCertificate`. All these attributes are optional and can be added in advance or on the fly.

The trust center must be able to determine which key management mode applies to each entity. To simplify the further description we assume that this is realized as a policy decision based on the entry's distinguished name and / or its object class. However, if more fine grained adjustments shall be possible, an attribute can be provided that regulates this matter for each entry.

During renewal operations the trust center must be able to determine if a key must be renewed also. This is again a policy decision. We facilitate this by including the private key usage period in the X.509 certificate in the respective extension as described in [2, Par. 4.2.1.4]. This is especially useful when the certificate's validity period is significantly shorter than that of the private keys. A new key is then either certified if the remaining key usage period is too short for issuing a new certificate based on the old key, or if the old key was removed by clearing the attribute `userCertificate`. An administrator can do this to enforce a renewal.

4.1. Registration and Initial Certification

The first state that is visible to our mechanism is `ENTRY_PLAIN`. Initially it describes the state when the entity does not yet have a certificate but all required information is in place in its directory entry. Thus, the requirements 1 to 3 of Section 3 are achieved by the processes that lead to this initial state. The same applies to requirement 4. However, it is possible for the trust center to additionally check whether the required information is really in place in order to ensure that the certificate policy is met.

This state can be denoted in two ways. Firstly, by directly setting the attribute `pkiStatus`. It can then easily be detected by the trust center as the respective

query only includes one attribute which allows for easy comparisons and can further be indexed. The drawback is that the processes which establish the entry's content need to be modified accordingly. This can be avoided at the expense of more complex queries on the directory. The trust center must then check if the entries already provide sufficient information for issuing certificates, for example if the common name, the e-mail address, and the public key are already present. The attribute `pkiStatus` can then be created by the trust center in the respective entries.

The trust center now determines the key management mode. As a user certificate is not yet set, the certification of a new key is requested and the existence of proper key information is checked. The trust center derives the content of the certificate from the entry's attributes and from values that are predefined by the certificate policy. It then changes the attribute `pkiStatus` to `CERTIFICATION_INITIATED` to show that a certification is ongoing. Two things are important here. Firstly, the trust center must enforce that the derived subject distinguished name is unique. This is already the case if the entry holds attributes that can be considered to be already unique like an e-mail address. Also the entry's DN provides a unique name. It can at least be used to detect naming collisions. Secondly, according to requirement 7, the certificate must be linked to the *registration data*. We do this directly by including the entry's DN as alternative name in the certificate.⁶ This is also necessary for the security of revocations. This is explained in Section 4.3.

The certificate request is forwarded to the CA where a respective certificate is issued. Afterwards the certificate is transferred back to the directory into the attribute `hiddenUserCertificate`. It is not published, i.e. written to `userCertificate`, because the certification process is not yet finished and thus no one should use the certificate. If the request included the generation of a PKCS#12 token, this is put into the attribute `userPKCS12`. To avoid multiple processing of the same piece of data, the attributes `userPassword`, `userPKCS10` and `userPublicKey` are cleared if they exist. The `pkiStatus` is then set to `CERTIFICATION_DONE`, to signal that the trust center has completed the request and is waiting for the entity to confirm receipt.

The entity can now download the certificate and optionally the private key from the directory. It is now able to complete its local data in order to use the PKI. Simultaneously, it publishes the certificate in the attribute `userCertificate` to make it available for others. This assures that a certificate cannot be used before the entity has updated its local data. The process can be enhanced to an indirect proof-of-possession as described in [8], 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 be encrypted with the contained public key and written to `userEncryptedCertificate`. The entity must then be able to decrypt the certificate, in order to have it published.

⁶As we assume that the structure of the DIT was established independently of the PKI, we cannot expect that the entry's DN and the subject DN are always the same.

4.2. Scheduled Renewal

The trust center is able to detect that a certificate has expired or will expire soon. The respective entry must show the status `CERTIFICATE_CONFIRMED` and the certificate's `notAfter` must be (nearly) reached. In our approach a renewal operation is then carried out automatically.

In the first step the trust center determines whether only the certificate or also the key needs to be renewed. This is done based on the information that is stored within the certificate.

When the key needs to be renewed, the trust center checks whether an additional step of interaction with the entity is required, for example to set a passphrase. In this case the status is temporarily set to `AWAIT_KEY_INFO` in order to signal this to the entity. Otherwise, the status is directly changed to `ENTRY_PLAIN`, from which a new certification is initiated. `AWAIT_KEY_INFO` can be avoided, by providing the respective information before the trust center starts the renewal. This is possible with scheduled events.

This process is also a means to refresh the certified information in a PKI. The validity period of the issued certificates should therefore be determined such that regular updates in the *registration data* fall together with the scheduled renewal of the certificates. The students at a university, for example, need to report back before the beginning of each semester. This is usually also the point, when changes to their *registration data* become visible to the university administration, especially when they do not report back. It is reasonable to have the certificates expire shortly afterwards. By this the PKI can adopt to changes without producing unnecessary revocations.

4.3. Revocation by the Entity

An entity can request the revocation of its certificate, for example, if the private key is compromised or broken or the e-mail address has changed, by setting the `pkiStatus` to `REVOCATION_REQUESTED`. The trust center now fetches the certificate from the respective entry. It then checks two factors. Firstly, the certificate's signature to ensure that it contains authentic information. Secondly, it compares the entry's DN to the one that is contained in the certificate in order to assure that the certificate and the entry correspond. This is important for the security of the PKI. The certification process requires that an entity is able to write into the attribute `userCertificate`. An attacker could launch a denial of service attack by putting another's certificate into its attribute and initiate the revocation. If correspondence was not checked, the trust center would not have any possibility to reject the revocation.

The trust center sets the `pkiStatus` to `REVOCATION_INITIATED` if it accepts the request, and initiates the production of revocation information by the CA. When the revocation has been carried out and published, the `pkiStatus` is set to `REVOCATION_CONFIRMED`. The process now remains in this state until it is manually changed. This allows the administrator to manually examine the reason that lead to the revocation. In addition the administrator can now determine whether renewal or termination of the certification is the appropriate reaction.

4.4. Termination of Certification

The life-cycle management functions have been designed to automatically extend the certification whenever possible. As a consequence its termination needs manual intervention, usually carried out by an administrator. The operation is initiated by setting the entry's status to `TERMINATION_REQUESTED`. The trust center now takes the certificate from the entry and performs the same check for correspondence as described in Paragraph 4.3. If the certificate does not expire within a short time, a revocation is initiated. As no further interaction is required, the trust center can now immediately remove the certificate and other PKI relevant data from the entry and set the status to `TERMINATED`.

4.5. Early Renewal

Early renewal is necessary, if *registration data* changes significantly, i.e. if the content of a certificate is affected while the respective certificate is still valid, or if the private key cannot be used any more. An early renewal is indicated by changing the `pkiStatus` to `EARLY_RENEWAL`. Depending on the reason this may be done manually by an administrator or automatically by the process that led to the data change.

The process that follows is a combination of the revocation and the scheduled renewal. However, the consecutive execution of both processes would lead to two consecutive requests to the CA with the additional drawback that the second one cannot be launched until the first has succeeded. As we do not want to put any requirements on the frequency with which data is exchanged with the CA, this is not applicable. Instead, we model this function as a process of its own that allows putting the two requests to the CA in parallel.

If the trust center detects the status `EARLY_RENEWAL`, it prepares a revocation request for the old certificate. This is done in the same way as described for the 'normal' revocation (see Paragraph 4.3). In parallel, it determines whether a key renewal is indicated. The decision is made on basis of the private key usage period and the general certificate policy. An administrator may enforce the key renewal by removing the certificate from `userCertificate` prior to changing the status to `EARLY_RENEWAL`.

If the key shall be renewed and no proper key material is already contained in the entry, the status is changed to `AWAIT_KEY_INFO`, otherwise directly to `ENTRY_PLAIN`.

Similar to the initial certification, triggering this function is done by explicitly changing the status. Consequently, already existing processes have to be modified, if they shall cause it. Again, this can be avoided by directly querying the *registration data* represented by an entry and comparing it to the contained certificate⁷ at the cost of more complex search operations. If a change is detected, the early renewal function is triggered automatically.

⁷To avoid retrieving the certificates each time, a hash value over the content of relevant attributes can be utilized. It is stored per entry and regularly compared to the current content to detect changes.

Not all changes to the *registration data* can be handled automatically. If the changes affect the entry's DN, i.e. the entry is actually moved in the DIT, the correspondence check that is performed prior to the revocation will fail. In this case it is not possible for the trust center to decide whether the entry was moved or the certificate was deliberately changed. It is advisable that the status is then changed to some error state that allows an administrator to manually resolve the problem. However, all other kinds of changes can be accomplished without manual intervention.

4.6. Support for Multiple Certificates per Entity

The attribute `userCertificate`, as a multi-valued attribute, already provides the possibility to store more than one certificate per entity. However, the attributes that control our mechanism, especially the `pkiStatus`, directly relate to a certificate. This cannot be achieved by means of multi-valued attributes, as they constitute an unordered set of values. An enhancement of the LDAP protocol allows selecting subsets of values by means of special filters [16]. While it solves the problems that occur when certificates need to be selected from a multi-valued attribute based on their content, it is still not sufficient for our purposes. The `pkiStatus` changes often and cannot be included in the certificate. Filters do not suffice to establish relationships between specific values of two multi-valued attributes.

The solution at hand is to store every certificate in a separate entry. These entries should be subordinate to the entity's entry in order to represent the binding. Such a proposal had appeared as draft at the IETF. This approach perfectly combines with our mechanism. Each certificate entry then holds one certificate together with the `pkiStatus` attribute and optional key information. Moreover, this allows handling the key management differently for each certificate.

5. 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 personnel of organizations is already implemented in a way that satisfies their security needs. Of course these processes have to be reviewed regarding their security properties before applying the proposed mechanism.

We now describe the requirements for the access control policy that is enforced by the directory. We thereby distinguish between anonymous access, access by an entity to its own entry and access to others' entries either by an administrator or by the trust center.

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 address book should be publicly available, and the certificates in order to allow sending encrypted messages. They must not be allowed to modify any entry. The attributes `hiddenUserCertificate`, `encryptedUserCertificate`, `userPublicKey`, `userPKCS10`, `userPKCS12`, and `userPassword` must not be visible to them.

An entity that binds to its entry may see all of its contents. In any case it must be able to read the entries `pkiStatus`, `hiddenUserCertificate`, and `userEncryptedCertificate` as required by the certification processes.

The entity must not be able to (unconditionally) manipulate the attributes of its entry that represent its *registration data*. Otherwise it could forge the contents of the certificate. However, it is sometimes applicable that the entity may choose such values in a restricted way, for example initially select an e-mail address from a restricted set of possible addresses. Directory servers do not support access control on basis of the values that are to be stored in attributes. Consequently, this behavior has to be emulated by a proxy application that controls the input and is granted respective write permissions on the target attribute. This can be easily implemented as a web application.

The same applies to the `pkiStatus` attribute. The entity must be able to change its values in a restricted manner. For example, changing the status `TERMINATED` must be prohibited. Again, access must be mediated by a proxy application. This situation can be avoided by describing the status with two attributes per entry, for example `pkiStatusTC` and `pkiStatusEntity`. The entity is then only granted the right to manipulate `pkiStatusEntity` and thus to change the status only in a restricted way. A suggestion for this is described in Table 2. The entity further needs write access to the attributes `userPublicKey`, `userPKCS10` and `userPassword`, if they are used.

The trust center needs access rights for the entries of all entities. It needs to write the attributes `pkiStatus`, `hiddenUserCertificate` and, respectively, `userEncryptedCertificate` to carry out its tasks. Furthermore, it needs to be able to read all attributes that make up the *registration data*.

Our approach allows for manual administration of all or certain certificates. This is done by granting administrative personnel write access to the `pkiStatus` attribute of other 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 who is responsible for managing the certificate requests of 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 maintain their life-cycle.

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 its entry. We suggest establishing this binding by adding the entry's DN to the certificate's `subjectAlternativeName` and having the trust center check this binding. Otherwise an entity could revoke arbitrary certificates of the PKI by writing the targeted certificate to the entry and request-

ing a revocation. The trust center would then revoke this certificate without any possibility to reject illegal requests.

The approach puts relatively high security requirements on the LDAP directory. It is a legitimate question, why not to use the directory directly for disseminating public keys. The following reasons support the usage of X.509 certificates:

Firstly, a lot of applications already implement X.509 certificates. From this point of view it would be a proprietary solution to directly provide the public keys in a (trusted) directory. Relying applications would have to be adapted.

Secondly, certificates can also be used without direct connection to a directory. In the proposed approach a direct connection is only required in order to manipulate a certificate's life-cycle status and not to prove the authenticity of public keys.

Thirdly, only the owners of certificates of the PKI need to trust the directory. All others only need to rely on the authenticity of the certificates, but no trust in the authenticity of the directory is required.

Fourthly, PKIs provide concepts that exceed the boundaries of one PKI like cross-certification or hierarchical trust. These concepts are not compatible with the idea of directly providing public keys in directories.

6. Case Study

We have implemented the proposed mechanism in a project at the Technische Universität Darmstadt. This project started in 2004 and the goal 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 to protected web sites, to sign up for exams and get knowledge of the results. Thus the targeted security services were mainly authenticity and non-repudiation. The students' private keys should be kept on secure smart cards. The main goal was to keep management effort at a minimum during the initial enrollment and the operation for a user group of 20,000 students. The PKI started operation in October 2005.

We now provide more details about this PKI. Initially, the university was already operating an X.500 directory. It is maintained by two administrators. It already contains all personal and administrative data that was necessary 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 postal service 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 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 modeling the PKI was that the students are unknown to the university until they appear for matriculation. Thus we are not able to do any

preparations that need the knowledge of a student's identity. On the other hand, it is no option to have each student appearing a second time for any management issues. Each additional minute counts times 20,000.

We installed the trust center software at the IT department. 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 only with the X.500 directory over an SSL/TLS secured channel.

We use smart cards for protecting the students' private keys. Each card is equipped with two chips. One is a crypto-processor for the PKI functionality and contains the private key. The second one is a contactless chip for anonymous electronic payment services within the university. The cards are pre-personalized by the manufacturer, who is also responsible for sending the cards to the students per postal service. 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. A batch process puts the public keys into the corresponding entries in the directory. The keys on the card are protected by a special Transport-PIN. I.e. 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 return to this state again. By checking this state upon receipt of the card the student can ensure 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 `ENTRY_PLAIN` when a student has ascertained the e-mail address. The modifications of these services were only minor but have significantly improved the speed of the directory queries conducted by the RA. 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 download 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 by using the `userEncryptedCertificate` attribute as proposed in [8]. Since we use RSA keys, we employ this scheme for signature keys as well. The application is capable of checking the Transport-PIN status of the smart cards. Implementing a dedicated application was less time consuming than integrating the functionality in all PKI-enabled applications that the students will probably use (e.g. e-mail client, web browser, VPN client).

The benefit from this approach was as follows:

- The directory was already set up and contained all necessary information. No additional registration step and no manual processing were needed. Thus time as well as money was saved.
- The processes which deal with the information stored in the LDAP could remain nearly unchanged. No additional personnel was necessary and no additional processes had to be established. The whole PKI is maintained by the two operators, which have also been responsible for the directory server.

- The maintenance of the directory was already known to the operators. This knowledge is sufficient for also maintaining the PKI. Therefore, no new skills were required.

7. Conclusion and Future Work

We have shown how the administration of entities in an organization can be synchronized with the life-cycle management of their certificates. The approach is based on the reasonable assumption that directory services have already been set up in many organizations. It allows to bootstrap a PKI in a cost efficient and less error-prone way. Both 'traditional' certifications and short-lived certificates are supported. Distributed administration and administration of groups is possible. The mechanism can be used with a variety of key management modes like smart card or soft token based PKIs. It is modeled as a finite state machine and meets the requirements for the respective management operations. The applicability of the mechanism is shown in a case study.

Our approach may be combined with the Client Update Protocol for LDAP that was proposed by the IETF [17]. Employing this protocol the RA could avoid polling data from the directory. Instead the RA would be notified upon changes.

A. Object Class and Attribute Definitions

We now present the definitions of the object class and its attributes that are introduced in this article.

```
# 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))

# hidden user 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 )

attributetype ( 1.3.6.1.4.1.8301.3.2.2.1.10.8
  NAME 'userPassphrase'
  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 [8]
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] Recommendation X.509 ITU-T. Information Technology - Open Systems Interconnection - The Directory: Authentication Framework. 1997.
- [2] 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, 2002.
- [3] B. Ramsdell. Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification. *IETF Request For Comments*, 3851, 2004.
- [4] T. Dierks and C. Allen. The TLS Protocol Version 1.0. *IETF Request For Comments*, 2246, 1999.
- [5] A. Årnes, M. Just, S.J. Knapskog, S. Lloyd, and H. Meijer. Selecting Revocation Solutions for PKI. In *Proceedings of The Fifth Nordic Workshop on Secure IT Systems (NORDSEC 2000)*, 2000.
- [6] Y. Hsu and S.P. Seymour. An Intranet Security Framework Based on Short-lived Certificates. *IEEE Internet Computing*, 02(02):73–79, 1998.
- [7] J. Hodges and R. Morgan. Lightweight Directory Access Protocol (v3): Technical Specification. *IETF Request For Comments*, 3377, 2002.
- [8] 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, 2004.
- [9] RSA Labs. PKCS#12 v1.0: Personal Information Exchange Syntax Standard. 1999.
- [10] 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.
- [11] P. Gutmann. Plug-and-Play PKI: A PKI your Mother Can Use. In *Proceedings of the 12th USENIX Security Symposium*, pages 45–58, 2003.
- [12] C. Adams and S. Farrell. Internet X.509 Public Key Infrastructure Certificate Management Protocols. *IETF Request For Comments*, 2510, 1999.
- [13] M. Lippert, E. Karatsiolis, A. Wiesmaier, and J. Buchmann. Directory Based Registration in Public Key Infrastructures. In *The 4th International Workshop for Applied PKI, IWAP 2005*, pages 17–32, September 2005.
- [14] J. Myers. Simple Authentication and Security Layer (SASL). *IETF Request For Comments*, 2222, 1997.
- [15] RSA Labs. PKCS#10 v1.7: Certification Request Syntax Standard. 2000.
- [16] D. Chadwick and S. Mullan. Returning Matched Values with the Lightweight Directory Access Protocol version 3 (LDAPv3). *IETF Request For Comments*, 3876, September 2004.
- [17] R. Megginson, M. Smith, O. Natkovich, and J. Parham. Lightweight Directory Access Protocol (LDAP) Client Update Protocol (LCUP). *IETF Request For Comments*, 3928, 2004.

Table 1. Object classes commonly used for modeling 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 <code>person</code> , adds information as postal address and room number.
inetOrgPerson (S)	Derived from <code>organizationalPerson</code> , 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.

Table 2. Allocation of the life-cycle status information with two attributes. An entity has only write access to the attribute `pkiStatusEntity`.

<code>pkiStatus</code>	<code>pkiStatusTC</code>	<code>pkiStatusEntity</code>
ENTRY_PLAIN	0	0
CERTIFICATION_INITIATED	1	0
CERTIFICATION_DONE	2	0
CERTIFICATE_CONFIRMED	2	1
REVOCACTION_REQUESTED	2	2
REVOCACTION_CONFIRMED	3	0
EARLY_RENEWAL	4	0
AWAIT_KEY_INFO	5	0
TERMINATION_REQUESTED	10	0
TERMINATED	11	*