PAMINA: A Certificate Based Privilege Management System

Zoltán Nochta, Peter Ebinger and Sebastian Abeck
University of Karlsruhe, Institute for Telematics,
Cooperation and Management IT-Research Group
Zirkel 2, 76128 Karlsruhe, Germany
Email:[nochta|ebinger|abeck]@tm.uka.de

Abstract

In this paper we present PAMINA (Privilege Administration and Management INfrAstructure), a privilege management system using authorization certificates. Our system supports distributed environments where autonomous authorities can manage and delegate privileges in accordance with their own policies. We introduce Improved Certification Verification Trees (I-CVTs) that guarantee very efficient and trustworthy certificate management. I-CVTs can provide undeniable proofs for the non-existence of a given certificate in contrast to CVTs as proposed in [1]. As a result, each authority can store its own I-CVT in a central, non-trusted, and replicable database. This database provides authenticated verifiers with basically only those certificates that are required to determine whether a user should be granted access to a resource or not. Since the system implements the pull model, clients need not to be involved in the access control decision process. PAMINA handles delegation trees instead of simple delegation chains because authorities can delegate privileges in one certificate that were assigned to them by several certificates. In the prototype that we describe here, PAMINA manages certificates based on X.509.

1 Introduction

The increasing use of the Internet for business critical transactions since the early 90's implied the development of different security systems and services, which are often based on public-key cryptographic techniques. Basically, there are two mechanisms that can be used by security systems, namely encryption and digital signatures. Nowadays, the most important applications for both are secure key exchange and integrity and authenticity protection of different kinds of data. In general, a certificate is a data structure that defines an association between an entity and a set of information. A certification authority states its belief in the validity of these associations by digitally signing the certificate. For example, a public-key certificate binds a public-key to a set of information that identifies the entity associated with the use of the corresponding private key. The aim of a certificate management system is to offer services that guarantee the authenticity, validity and integrity of security information (keys, user rights, etc.) stored in certificates and used by applications for authentication and authorization purposes.

A widely accepted definition of authorization is the granting of access rights to a subject. However, there is a difference between the administrative act of asserting that a subject should be granted access rights (privileges), and the operational act of allowing an authenticated subject to access a resource after determining that it holds the required set of privileges. The latter process is also referred to as access control decision.

An **authorization certificate (AC)** contains information that is generated during the authorization and used in the access control decision process. ACs have many benefits in privilege management due to the fact that they can support different access control models and privilege delegation schemes, because they are individually protected by digital signatures against fraud. Linn et al. give a good overview of application scenarios of several types of ACs that contain different kinds of authorization information, such as privileges or role definitions [2].

Surprisingly, only few access control systems exist that use authorization certificates. One reason for this fact is that both certificate management and verification generate additional computational and communicational costs that customers and providers have to bear. Widely used certificate management techniques, like Certificate Revocation Lists [3] that have originally been designed for public-key infrastructures, are not very attractive for an efficient system design, because authorization related information has other characteristics than public-keys. Access rights can change very frequently, the amount of authorization information is usually higher, and authorization certificates often must be handled confidentially.

Fortunately, there are several novel approaches with enhanced properties that can make certificate management and verification less expensive and, in many cases, more secure as well.

In this paper we introduce **PAMINA[1] (Privilege Administration and Management INfrAstructure)**, a certificate based system that provides authorization and access control in distributed environments. PAMINA is the first system that uses **Improved Certification Verification Trees (I-CVTs)** which guarantee very efficient and trustworthy certificate management. I-CVTs also provide undeniable proofs for the non-existence of a given certificate statement in contrast to CVTs as proposed in [1]. The system supports distributed environments where autonomous authorities, so called **Privilege Management Authorities (PMAs)**, can manage and delegate privileges in accordance with their own policies. Each PMA can store its own I-CVT in the central, non-trusted, and replicable **Privilege Database (P-DB)**. This database provides authenticated verifiers with basically only those certificates that are required to determine whether a user should be granted access to a resource or not.

The rest of the paper is organized as follows: In the next section we summarize the most important requirements that PAMINA as a certificate based privilege management system has to fulfill. In section 3, we describe and compare different approaches and choose a combination of best-fit solutions. We show that CVTs as proposed in [1] are qualified for efficient privilege management if the underlying data structure is chosen properly. In the following section, we describe the concept and the implementation details of Improved CVTs and certification paths, which reduce the required level of trust placed on the database storing the certificates. Finally, section 5 deals with the design aspects of PAMINA, the components and their interactions. The supported delegation model as well as some technical details are described.

2 Requirements for PAMINA

The aim of a privilege management system is to support security critical applications with valid information about the current status of the users' access rights (privileges). Digitally signed authorization certificates protecting the integrity and authenticity of privileges enable the realization of different authorization models in a more secure way than conventional systems do. For example, privilege delegation chains becomes more secure when using certificates. The components of a distributed certificate based privilege management system must provide proper mechanisms, data structures, and tools for the management and verification of certificates. In the following we summarize the most important security and performance related requirements that PAMINA has to fulfill, in order to be competitive with existing privilege management systems.

- **Efficient certificate lifecycle management:** The system must handle at least certificate generation, update and revocation. The underlying certificate management scheme should optimize communication, data maintaining and computational costs. High communication costs between a database storing certificates and the issuing authority can be critical for organizations that manage a huge number of frequently updated/revoked ACs. In order to make PAMINA attractive for access control applications (verifiers) that want to verify user rights, communication costs between the directory and the verifier should also be minimized. Optimized computational costs, for example by the reduction of the number of required signature verifications, speed up the access control process.
- **Revocation:** The system must handle situations, like the unexpected change of user privileges before the certificate expiration date or the key compromise of an issuing authority.
  In the first case, the certificate of the affected user has to be revoked and reissued. **Short-lived certificates** can reduce the probability of revocation before expiration or even make revocation unnecessary. This leads to increased costs of the issuing authority, since it must sign and distribute certificates very often. Therefore, the costs caused by certificate signing should be reduced, in order to use short-lived certificates.
  In the second case, the fast and efficient revocation and re-issuing of all certificates signed by the authority whose key was compromised is required. The system has to enable the easy exchange of an authority's key pair.
- **Freshness of certificates:** The system should provide recent authorization information. Certificate status information should be published on a regular basis, even if the situation does not change, so verifiers can be sure that their data is up to date.
- **Reduced trust on database:** From the verifiers' point of view there are two main types of components in a distributed certificate management infrastructure. Firstly, there are trusted authorities that issue and sign certificates. Secondly, there are also components (e.g., directories, online responders) that store and distribute certificates or information related to certificates (e.g., revocation status). The latter type of components cannot manipulate a certificate, but they can accidentally or intentionally provide verifiers with incorrect information, e.g. about its revocation status, or they can hold back relevant information, e.g. the certificate itself or a revocation list. Due to these facts in many existing systems a verifier must also trust in these components, because he is not able to check the correctness of this kind of information. This trust should be reduced, and therefore the database should be able to **prove the non-existence of a certificate** if he does

[1] Pamina is also a beautiful princess in W. A. Mozart's The Magic Flute

not deliver it. Naturally, the database always can just refuse to respond to a request.

- **Privacy:** Public-key certificates used for signature verification or for encryption have to be available for everyone. On the other hand, ACs storing authorization information should be kept private. This means that only a fixed set of entities (administrators and verifiers) should have controlled access to this information.
- **Cryptographic security:** In order to protect certificates against forgery, the system must use secure (long) signing keys. Signing keys should be kept in a secure environment, which makes it impossible to retrieve them.
- **Interoperability:** In order to provide interoperability and usability, PAMINA should support the management of different (standardized) certificate formats. For the encoding and representation of certificates platform-independent and standardized storage formats should be used.
- **Flexibility:** The system must be able to provide different access control models and policy schemes without technical impact. Therefore, the model has to deal with various certificate types that allow, for example, the construction of abstract groups or delegation of permission rights.
- **Availability and robustness:** Naturally, the system has to make sure that the availability of the certificates and their status information is as high as possible. The system must be based on a robust model, which does not allow any manipulation of authorization information, even if the underlying computer systems crash. For example, in case of a successful attack against the directory that stores the certificates it should not be possible for the intruder to make a revoked certificate pass as valid.
- **Auditing and non-repudiation:** The system has to support easy verification of the validity status of a certificate at any point of time. In order to provide timeliness of the authorization information, the system has to implement (or utilize) time stamp services. Another important aspect is the accountability for both authority's and user's actions. The system must not allow the creation of rogue certificates or the unauthorized revocation of certificates without being detected.

3 Evaluation of certificate management approaches for access control purposes

In this section different competing management techniques are evaluated and compared in order to choose from and combine best-fit methods for our purposes.

3.1 Extended public-key certificates vs. separate identity and access control management

One possible way to manage access control information of certified users is the use of extended public-key certificates (PKC). In this case, one authority is responsible for both key (identity) and privilege management. Since there is only one authority, the verification process becomes simpler, because a single trust path reflects both authentication and authorization of a user. As a result, fewer transactions are needed when verifying requests. This approach has benefits in systems where the validity periods of certified keys and privilege assignments are similar. On the other hand, in most cases user rights should not be open to the public. Through obtaining an extended PKC, one can learn a lot about the internal structure of an organization, since it is, for example, reflected by the roles defined in the certificates. In addition, authorization certificates (ACs) must be often issued with short validity periods (e.g. hours), contrasting with PKCs whose validity period is often measured in years.

Due to these facts, separate access control management is required in most environments. Using this concept, authorization certificate updates have no impact on longer-lived identity certificates. This is especially profitable for clients storing PKCs for authentication purposes. In addition, independent authorities issuing their own certificates make the system more flexible and this approach is also helpful to eliminate many problems caused by a central bottleneck. The most important disadvantage of such an architecture is the increasing complexity of verification and path processing. In the worst case each AC that must be verified for an access control decision can be issued from a different authority that belongs to a different verification path.

We decided to implement the first prototype of PAMINA for separated management of ACs. Therefore, the system needs the integration of a PKI, which is responsible for the distribution of PKCs that are used for signature validation. We believe that the use of appropriate data structures and algorithms can compensate many of the drawbacks mentioned above.

3.2 Push model vs. pull model

In the so-called "push" model, client, server, and the certificate management infrastructure are all involved to varying degree in the authorization process. A client must maintain (e.g., download and store) its certificates and present them to the targeted server (verifier) by inserting them in its request message. The verifier must make access control decisions on the basis of this information. Naturally, the verifier must check the correctness as well as the validity of the presented information. The infrastructure must implement mechanisms to provide all clients with the last updated ACs. This is a very expensive task in environments where user rights can change dynamically. Additionally, users having more than one AC could not know which certificate(s) they should send to a verifier to use a particular service. As mentioned before, ACs often should be kept secret, and there is no guarantee for the issuer that (probably unsecured) client machines can protect ACs properly. As a result, all clients would need to communicate over an encrypted (and authenticated) channel with the database very frequently which would lead to performance problems in the system.

In contrast to the push architectures, in systems built on the "pull" model, verifiers pull ACs from some online network service. This approach simplifies clients, because they must no longer care about the management of ACs. Additionally, users are not involved in the authorization process and they need not even to know which privilege sets they have. Since there are usually fewer verifiers than clients in a system, this model leads to reduced communication costs in comparison to the push model. Only verifiers have to pay a higher price, they must not just check the validity of certificates but also download them. The main disadvantage of this approach is that verifiers totally rely on the availability of the infrastructure storing the ACs. Moreover, the database must decide which particular ACs a given verifier may download.

PAMINA was primarily developed to support the pull model. All certificates managed by the system are stored in a central database, which can be replicated to get higher availability and better performance. This database provides only authenticated verifiers with those certificates that are needed to check a given user's rights.

3.3 Revocation mechanisms vs. freshness guaranties

Certificate revocation is the mechanism with which an issuing authority can revoke a stated association before its documented expiration. An authority may wish to revoke an authorization certificate, for example, in response to a change in the owner's access rights or because of the compromise of its own private key. There are many approaches to solve this important problem, here we analyze the most important ones of them.

Certificate Revocation List (CRL) is the most commonly used revocation mechanism in certificate management systems today (e.g., see [3] and [4]). A CRL is a continuously growing, digitally signed list of revoked certificates, which is published periodically. In order to keep the CRL size manageable, certificates are denoted by some unique identifier (id), such as a serial number or a fingerprint. When a revoked certificate's validity period ends it can be deleted from the CRL. Communication costs can be reduced by publishing periodically a delta-CRL, which is a differential list to the last CRL update. In order to verify the status of a certificate, a verifier first needs to obtain the latest CRL (delta-CRL), then verify the signature on it and search for the ID of the certificate in question. CRL-management can be very expensive for both issuing authorities as well as verifiers in access control systems since user rights change frequently and verifiers should obtain and check many certificates in order to verify delegation chains. In addition, to provide long-term non-repudiation issuers (verifiers) would have to store not only every AC but also every CRL they have ever issued (received). Another disadvantage of CRLs is given by the fact that they do not provide non-existence proofs for certificates.

The concept of **Certificate Revocation Trees (CRTs)** was proposed by Paul C. Kocher [5]. A CRT enables verifiers to get a proof that a certificate has not yet been revoked. Basically, a CRT is a binary hash tree [6], in which each leaf consists of the ID of a single revoked certificate and a range of valid IDs all greater than the revoked one. During verification the verifier obtains the hash path belonging to the ID of the certificate in question, then it checks the signature on the root and verifies that the hashes correctly bind the leaf to the root. Finally, it checks whether the ID is the lowest in the leaf.

CRTs reduce the communication cost between the verifier and the directory, but increase the authority's computational cost, which is straight proportional to the number of revoked certificates. This fact makes CRTs not very attractive for an access control management system. CRTs do as CRLs not support non-existence proofs for certificates. However, the main drawback of this system is that the insertion (deletion) of a new revoked (expired) certificate might result in the re-computation of the entire tree.

Naor and Nissim eliminated this problem by replacing the suggested binary tree with a more effective $B_{2,3}$-tree [7]. In this case, it is no longer required to change the whole tree when inserting or deleting a certificate but just one path.

All the schemes above are constructed to maintain revocation information only. Table 1 shows an overview of the average computational and communication costs that verifiers and the certificate management system have to take into account. In addition to the listed costs for managing revoked certificates, there are of course the costs for generation and management of valid certificates. Using revocation mechanisms the verification process is rather complicated, especially when more than one AC need to be checked, for example for delegation path construction.

The **Online Certificate Status Protocol (OCSP)** was specified to support the communication between verifiers and a trusted entity referred to as an OCSP responder, which supports verifiers with information about the revocation status of certificates [8]. The main aim of OCSP is to reduce the communicational and computational costs of the verification process: Instead of checking the revocation status of certificates in question, a verifier sends a simple request to the responder containing one or more certificate

n: avg. total number of certificates per authority
r: avg. number of revoked certificates per authority
u: avg. number of revoked certs since last update per authority

l_{id} : length of a certificate identifier (bits)
l_{stat} : length of a revocation status number (bits)
l_{sig} : total length of a signature (bits)

Revocation scheme	Authority computational costs	Verifier computational costs per query	Directory update communication costs per authority	Communication costs per verifier, per directory query
CRL	O(u)	O(r)	$r\cdot l_{id}+l_{sig}$	$r\cdot l_{id}+l_{sig}$
Delta CRL	O(u)	O(r+u)	$u\cdot l_{sn}+l_{sig}$	$u\cdot l_{id}+l_{sig}$
CRT	worst case O(u·r)	O(log(r))	$u\cdot l_{id}+2\cdot l_{sig}$	$\log(r)+l_{sig}$
Naor/Nissim	O(u·log(r))	O(log(r))	$u\cdot l_{id}+2\cdot l_{sig}$	$\log(r)\cdot l_{hash}+l_{sig}$

Table 1: Cost analysis of different certificate revocation schemes

Freshness scheme	Authority computational costs	Verifier computational costs per query	Directory update communication costs per authority	Communication costs per verifier, per directory query
CRS	O(n)	O(#updates)	$n\cdot(l_{id}+l_{stat})$	l_{stat}
CRS2	O(u·log(n))	O(log(n))	$u\cdot l_{id}+2\cdot l_{sig}$	$\log(n)\cdot l_{hash}+l_{sig}$
CVT	O(u·log(n))	O(log(n))	$u\cdot l_{id}+2\cdot l_{sig}$	$\log(n)\cdot l_{hash}+l_{sig}$

Table 2: Cost analysis of different schemes providing freshness information

identifiers. In its response the OCSP responder sends the revocation status of those certificates back to the verifier. Naturally, to generate such a response the OCSP responder has to gather revocation information from some backend system that has to maintain revocation status information, for example with the use of a CRL or a CRT. OCSP does not specify or enhance a particular revocation scheme but it just defines a protocol for retrieving revocation status information. The main problem with this approach is that the verifier must trust the responder, he must believe that the revocation status information he gets is correct and up-to-date. A signed and time-stamped OCSP response might be a real-time generated message, but the verifier cannot check when the included information was actually generated by the issuing authority. Furthermore, since OCSP responses have to be signed, there must be also a public-key certificate issued for the responder itself which must be known to the verifier. The verifier should be able to check the current status of this certificate, too. This can of course not be done with the help of the OCSP responder, therefore some additional mechanism is needed. Since online responders (not only OCSP responders) do not improve the underlying revocation scheme and would raise additional problems, they seem not to be qualified for our purposes.

Fortunately, there are also certificate management schemes that can provide the revocation status of a certificate and freshness information at once. This combination reduces data maintaining costs, since there is no more need to manage two separate databases.

The **Certificate Revocation System (CRS)**, which was the first system maintaining freshness information for both valid and revoked certificates at once, was invented by Silvio Micali [9]. In the CRS the issuing authority periodically sends a signed (and time-stamped) message for every certificate stating whether the certificate was revoked or not since the last update. For this purpose an off-line/on-line signature scheme is used which reduces computational costs. CRS uses a one-way hash-function h. Before storing a certificate in the directory the authority chooses two random values R_0 and R_1 and then it computes and publishes the hash-values $h(R_1)$ and $h(\ldots(h(h(R_0))\ldots)$ or more precisely $h^k(R_0)$, where k is the expected number of update periods. When the freshness of a still valid certificate should be stated the authority must compute $h^i(R_0)$ and send it to the directory, where i is the total number of possible future updates. If a certificate must be revoked the authority simply sends R_1. In order to verify that a certificate is valid, a verifier has to query the directory for a copy of the most recent update value. As one can see in Table 2 the authority-to-directory communication costs are high because a new hash value for every certificate must be sent. An improvement (**CRS2**) that solves this problem is based on binary hash trees. In this scheme the current status of a given certificate is indicated by two bits. These bits, typically 128 are stored in the leaves of the tree. Nodes of the tree are computed from the hash of their children. Only the root of the tree must be signed by the authority. This modification speeds up the system, verification becomes cheaper and also communication costs decrease, as shown in Table 2. In addition, if the Merkle tree is constructed carefully, CRS2 can also provide proofs for non-existence of certificates. Unfortunately, this system provides information about the current status and existence of many neighboring certificates, which conflicts with confidentiality requirements.

Alternatively, revocation could be accomplished by simply removing revoked authorization certificates from the directory. This approach would allow authorizations to be changed in a very responsive manner, without establishing any kind of revocation infrastructure. The problems to solve are how verifiers can be sure that the directory contains all valid certificates and that all certificates in the directory are still valid.

Certification Verification Trees (CVTs) recently proposed in [1] can solve these problems elegantly. The basic idea is that it is not necessary to sign every single certificate issued by an authority. Instead, the authors suggest to store the unsigned certificates (certificate statements) plus a hash value in the leaves of a hash-tree [6]. Only the root of the tree must be signed and time-stamped. In order to check the validity of a given certificate, the verifier must obtain the certificate and the certification path belonging to this certificate. The certification path is given by the set of hash values of all siblings of the nodes along the path from the leaf containing the certificate statement in question to the root. This scheme allows very frequent freshness updates of all certificates at once. The authority does not need to maintain any extra information about revoked certificates. It can just delete a revoked certificate from the tree and then sign the root of the new tree. There are many other advantages of CVTs; but the most important one is the enhanced security: The exchange of the root key is easier and in the case of key compromise an adversary cannot manipulate single certificates, he always changes the signed root, which can be easier detected. Finally, longer and therefore probably stronger keys can be used. An effective CVT implementation can be based on a $B_{2,3}$-tree, for example. Some experimental results on this topic can be found in [10]. Table 2 compares the average costs of the different freshness schemes. As one can see the costs of CVT and CRS2 are very similar.

The very good performance, the enhanced security and the simplicity of implementation convinced us to realize PAMINA on the basis of CVTs. However, there are some problems with this CVT design that are discussed in the next section.

4 Improved Certification Verification Tree

A problem with the proposed CVTs described above is the following: When a verifier requests some set of certificates, for example all certificates issued for a particular user, the directory storing the CVT can not prove that it actually delivers all of those certificates. Processing the certification paths a verifier can easily check the validity of received certificates, but he cannot know whether he got all valid certificates. In other words, a malicious directory could disclaim the existence of a certificate. This means in the case of authorization certificates that a user might not be able to utilize all privileges that he or she was actually assigned by the issuing authority. Buldas et al. showed in [11] that one can not generate an undeniable non-existence proof using the proposed CVTs. The issuing authority could construct an incorrectly sorted tree and provide one verifier with a certificate and a second verifier with a contradicting “proof” for the non-existence of the same certificate. Therefore the structure of a CVT should be improved in such a way that it is possible to provide an undeniable proof of non-existence if the directory does not deliver a requested certificate.

Therefore, we construct **Improved Certification Verification Trees (I-CVTs)** that are very efficient and more trustworthy. The underlying data structure of an I-CVT is a variant of a balanced search tree (B-tree), a so called B^+-tree. A B^+-tree differs in that way from a B-tree that all data is stored in the leaves (see Fig. 1). A B^+-tree of order m satisfies the following properties [12]. Every internal node of the tree has at least $\lceil m/2 \rceil$ and at most m children. The root node has at least 2 children. A non-leaf node with k children contains k-1 keys. Leaves of the tree contain at least $\lceil m/2 \rceil$-1 and at most m-1 keys and as many pointers to data records (e.g., certificate statements). The leaves of the tree are linked which optimizes sequential access to the data records. In order to store certificates in a B^+-tree, an order of the certificates must be defined, e.g. a unique serial number or a certificate hash value can be used (s. below). B^+-trees have the following advantageous properties for certificate management:

- Each path from a certificate to the root has the same length. Therefore, the certification paths have the same size and the communicational and computational costs are similar for all certificates stored in the tree.
- The operations insertion, deletion and searching are very efficient in a B^+-tree, because they can be done in $O(\log_{\lceil m/2 \rceil} n)$ where n is the number of records (certificates) in the tree [13].
- B^+-trees can be optimized for search, insertion, deletion and also certification path length by choosing the parameter m properly.
- Easy and efficient search and sequential access to the certificates.

For certificate management purposes we extend the B^+-tree to a Merkle hash-tree. The signature on the root along with a properly constructed hash-path attests that a given certificate is stored in the tree and therefore valid.

In [1] the so-called certification path is defined as “the path from the leaf containing the certificate statement to the root, along with the hash values necessary to verify that path. This includes the hash values for all siblings of nodes along that path”. If the certification path and the signature on the root hash value are both valid, it is shown that the certificate is stored in the CVT and thus valid.

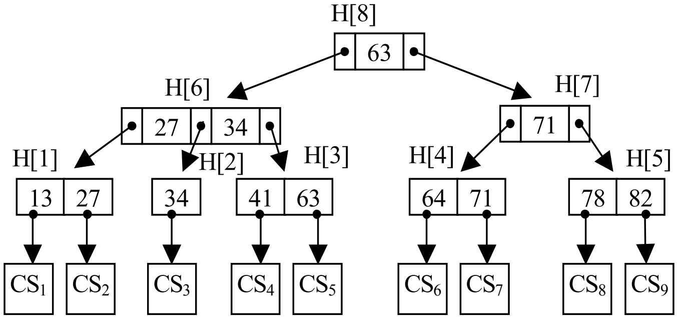


Figure 1: Improved CVT based on a B$^+$-tree (here m=3)

Since trust placed on the database storing and publishing the certificates should be minimized, it is important that the database must deliver also a proof of non-existence if it does not deliver a requested certificate. Otherwise the database could simply deny the existence of a certificate as mentioned above. If the system also provides non-existence proofs, the verifier does not have to “blindly” trust in the database. The results of Buldas et al. show that in order to provide an undeniable proof for the non-existence of a certificate, information about the order and internal structure of the tree has to be added to the hash-path. They have also pointed out that in CVTs as proposed in [1] this kind of information is missing.

Therefore we add information about the structure and the order of the a B$^+$-tree to construct I-CVT (see Fig. 1). The values H[i] are the hash values that are stored in the nodes of the tree. Each hash-value is computed from the search keys of the particular node and the hash values of its children. For example, the value H[1], which is stored in the leftmost leaf of the I-CVT in Fig. 1, is computed as follows: H[1] = h(13, 27, h(CS_1), h(CS_2)), where h must be a collision resistant hash function.

Also the search keys - that are stored in the nodes on the path from the leaf with the certificate to the root - are included in our certification path as follows:

Certification path cp = sequence of (sequence of keys, sequence of sibling hash values) + signature on root of the tree
cp = (l_0, l_1, ... , l_{n-1}) + root signature,
where l_i = ((k_{i0}, k_{i1}, … k_{it}), (h_{i0}, h_{i1}, ... h_{it}))

Example 1: *certification path* for the certificate statement with ID 27 (see Fig. 1)

cp(27) = ((13, 27; h(CS_1), h(CS_2)); (27,34; H[2], H[3]);
(63; H[7])) + signature on H[8].

The proof of non-existence of a certificate is simply the certification path for the leaf that would contain the certificate if it were in the I-CVT.

Example 2: *Non-existence* proof for the certificate statement with ID 42 (see Fig. 1)

cp(42) = ((41,63; h(CS_4), h(CS_5)); (27,34; H[1], H[2]);
(63; H[7])) + signature on H[8]

Note that the latter certification path can also be used as a certification path for the certificates with ID 41 and ID 63.

The hash value of the nodes on the path to the root and the positions within their siblings can be omitted, since the hash value of a node is determined by the hash values of its children and its position within its siblings by the search keys of its parent node. Fig. 2 shows the algorithm for creating certification paths.

Input: Identifier ID of a certificate, I-CVT
Output: Certification path or proof for the non-existence of certificate ID in the I-CVT

```
cp ← empty certification path
n ← leaf node which (should) contain(s) certificate id

finished ← false
while not finished
        k ← sequence of keys that are stored in n
        h ← sequence of hashes that are stored in n
        if n is not a leaf
                pos ← position of ID in k
                (such that $k_{i (pos-1)} < ID \le k_{i pos}$)
                delete hash value at position pos in h
        add (k,h) to cp

        if n is not the root node
                n ← parent node of n
        else
                finished ← true

return cp and the signature on hash(k, h) (=hash value of the root node)
```

Fig. 2: Algorithm for creating a certification path

One can show that this certification path is an undeniable attester as defined in [11]. This way the problems of CVTs we described above are solved. It can also be shown that we get the minimal certification path length for a B^+-tree with m=3 (i.e. a $B^+_{2,3}$-tree), but we expect to get faster search, insertion and deletion times for bigger m. This leaves some scope to optimize the data structure for a specific implementation.

In order to make the system flexible and suitable for different scenarios, we want to enable that an authority can issue more than one certificate for a particular user and store them in its I-CVT. A verifier should be able to check that he gets all certificates issued for this user, to make sure that access is not denied if the user has the required privileges. Therefore the certificates in the I-CVT are sorted by the pair (user–ID, serial number) which is unique since the serial number has to be unique in an I-CVT. The user-ID is the more significant part and the serial number is the less significant part, i.e. all certificates are sorted by user-ID and all certificates of a specific user are sorted by their serial numbers. This way all certificate statements of a particular user are adjacent and form a closed sequence. Verifiers get the certification paths for each certificate of this sequence and additionally for the certificates directly before and after it. As a result, the verifier can be sure that he got all certificates issued for a particular user.

5 Architecture of PAMINA

Due to the requirements that we defined and the results of the analysis of different competing approaches, we have decided to design PAMINA to support the pull model and to use separate certificates for privilege management. The cost analysis shows that CVTs guarantee a very good performance. In addition, this scheme has enhanced security properties. In order to reduce the trust placed on the database storing CVTs and to make the scheme more efficient, we developed I-CVTs. We believe that I-CVT is an enabling technology for the realization of open directories storing certificates issued by one authority. However, PAMINA is basically designed for environments where delegation of privileges as well as confidential management of ACs are needed. Enabling privilege delegation implies the coexistence of many I-CVTs that should be accessible for applications. Due to this fact we develop a central database storing all I-CVTs of the system. Of course, the system can support simpler scenarios, too.

At first, we describe the components of PAMINA and their relationships. Fig. 3 shows the subjects and components of the system including the most important data flows.

- **Resource**: Like any other privilege management system PAMINA maintains authorization information used for controlling access to different objects, the resources. Each resource is owned by one or more PMAs, and each PMA can own one or more resources. For example, PMA_2 owns resource R_B and R_C (see Fig. 3).
- **User:** From PAMINAs point of view, users are holders of one or more ACs. Users cannot issue ACs. We assume that each user has a unique identifier (name, public-key) which associates her/him with the privileges stored in her/his ACs.
- **Client:** In our approach a client is an application that represents a remote user. Since access control decisions can only be made if the requesting user's identity is available, the client system has to insert identification information of the authenticated user in its request messages.
- **Privilege Management Authority (PMA):** PMAs are basically issuers of ACs, they represent organizational units, such as companies or divisions of a company. Each PMA owns and/or controls one or more resources. PMAs are autonomous; they can manage privileges in accordance to their own policies. PMAs can delegate privileges to other PMAs in a controlled manner (see below in section 5.2). As shown in Fig. 3 resource R_C is owned by PMA_2, which has delegated some of its privileges to PMA_3. One can see that PMA_3 itself does not own any resources, but this way it can authorize its users to access R_C. Each PMA maintains exactly one signed I-CVT in which all certificates issued by that particular PMA are stored. We assume that the public (signing) key of the PMA is managed by an external PKI.
- **Privilege Database (P-DB):** The P-DB is a central, non-trusted database storing the I-CVTs of registered PMAs. In this context non-trusted means that a verifier obtaining ACs can always be sure that the P-DB can neither manipulate the ACs, nor it can disclaim the existence of a given AC. These properties are guaranteed by the digitally signed I-CVTs. PMAs must periodically send updated I-CVTs to the P-DB, according to their own update policy. The P-DB controls neither the validity, nor the freshness of the I-CVTs; it just stores them. ACs can be downloaded by verifiers, which are owned by registered PMAs. In order to avoid performance and availability problems the P-DB can be replicated.
- **Verifier:** In PAMINA a verifier is owned by at least one PMA. Verifiers control only access to resources that are owned by their owner PMAs. For example, V_{PMA2} is owned by PMA_2 as shown in Fig. 3. PMAs state this ownership with so called policy certificates (see below in section 5.1). Verifiers make access control decisions on the basis of the ACs downloaded from the P-DB. A verifier is a program which runs directly on the targeted system itself, or it is an external service used by server-sided applications, for example. In Fig. 3 client C, that was authorized by PMA_3, wants to use resource R_C controlled by verifier V_{PMA2}.

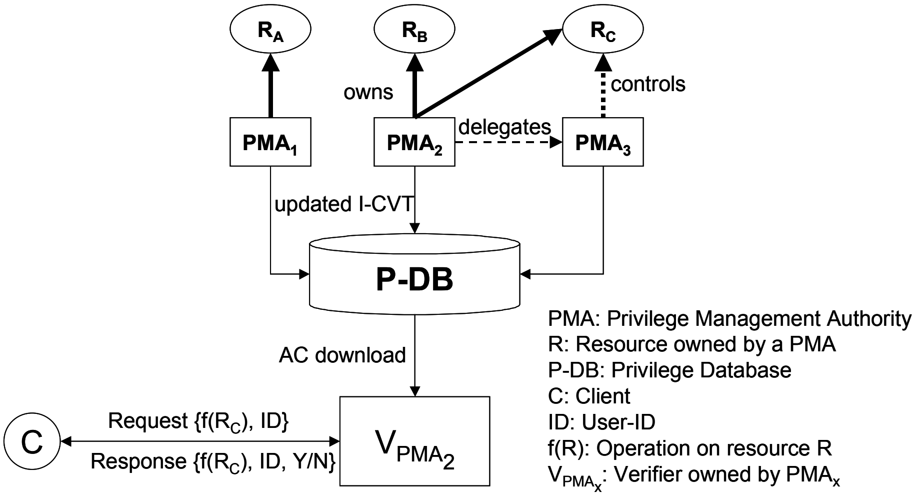

R_A
R_B
R_C
owns
controls
PMA1
PMA2
delegates
PMA3
updated I-CVT
P-DB
AC download
Request {f(R_C), ID}
Response {f(R_C), ID, Y/N}
C
V_PMA2
PMA: Privilege Management Authority
R: Resource owned by a PMA
P-DB: Privilege Database
C: Client
ID: User-ID
f(R): Operation on resource R
V_PMAx: Verifier owned by PMAx

Figure 3: Architecture scheme of PAMINA

In this case, the verifier downloads the delegation chain, which states the privileges of the user that are needed to execute the requested action, then it verifies the chain, and finally, it allows or denies the access to R_C.

- **Administrators:** Administrators, who are typically employed by a particular PMA, do daily management work. A framework consisting of management tools supports the two main types of administration related tasks, namely privilege management and certificate management. Only administrators of a PMA have access to the private (signing) key of this PMA.

5.1 Managing certificates based on X.509 with PAMINA

Theoretically, any type of (standardized) ACs can be managed using I-CVTs, because the construction of these trees is independent from the information stored in the leaves. However, due to the fact that single ACs need not to be signed, certificate fields holding signature specific information become needless when using PAMINA. Since certificate formats often reflect special authorization mechanisms, like delegation, access control model and properties of the revocation scheme, other changes could also be needed in some cases. The first practical scenario for PAMINA was to manage access control on the basis of ACs as defined in the X.509 standard [4], in order to support compatibility with existing X.509 PKI products. The "Attribute Certificate Framework" of X.509 basically specifies an AC format and recommends the (optional) use of CRLs for revocation purposes. This standard also defines a delegation scheme and deals with major aspects of role based access control management. Unfortunately, practical aspects of the realization of a privilege management infrastructure, like confidential management of ACs, are out of the scope of this framework.

In the following we describe how ACs as defined in X.509 can be modified in order to manage them with I-CVTs. An X.509 based certificate statement stored in the leaf of an I-CVT can consist of the following fields (compared to *AttributeCertificateInfo* in [4], the *signature* field is missing):

version, holder, issuer, serialNumber, attrCertValidityPeriod, attributes, issuerUniqueID, extensions

The type, format and meaning of these fields remain basically unchanged. Assuming that an I-CVT stored in the P-DB consists only of valid (not yet revoked) ACs, one could think that the field *attrCertValidityPeriod* is unnecessary. This could be true from the verifier's point of view, but this would make the administration of the system very complicated, because administrators would not know when they had to delete an expired AC from the I-CVT. The field *serialNumber* must uniquely identify an AC within the I-CVT signed by the *issuer*. The *attributes* field contains the privileges associated with the holder of the authorization certificate.

X.509 supports privilege management on the basis of role-based access control policies. There are several other known approaches competing with role-based models, in which users or privileges can be grouped according to other strategies. From the certificate managements point of view, there is no significant difference between these models (see also [14]). A privilege management system has to be able to handle ACs that hold privilege collections that are not issued for a single subject. PAMINA also supports this mechanism; the management of role certificates with I-CVTs and their verification do not present any additional difficulty.

As mentioned before, each verifier holds so-called policy certificates issued by its owner PMA. A policy certificate is a special AC that is mainly used for the configuration of verifiers. The *attributes* field of a PC can consist of information about the caching or access control strategy which the verifier has to follow. A PC also has a field that lists each PMA known to the issuer that has ever delegated any of its privileges. This way the verifier knows from which I-CVTs he needs either the certificates for the given user or a proof for the non-existence of any certificates of this user. The issuer PMA of a policy certificate states with its signature that it owns the given verifier. This statement is used by the P-DB to control access to the I-CVTs. See section 5.3 for more details.

5.2 Supporting delegation

As mentioned earlier, an important feature of PAMINA is the support of privilege delegation. The basic properties of the delegation model provided by the system are listed here:

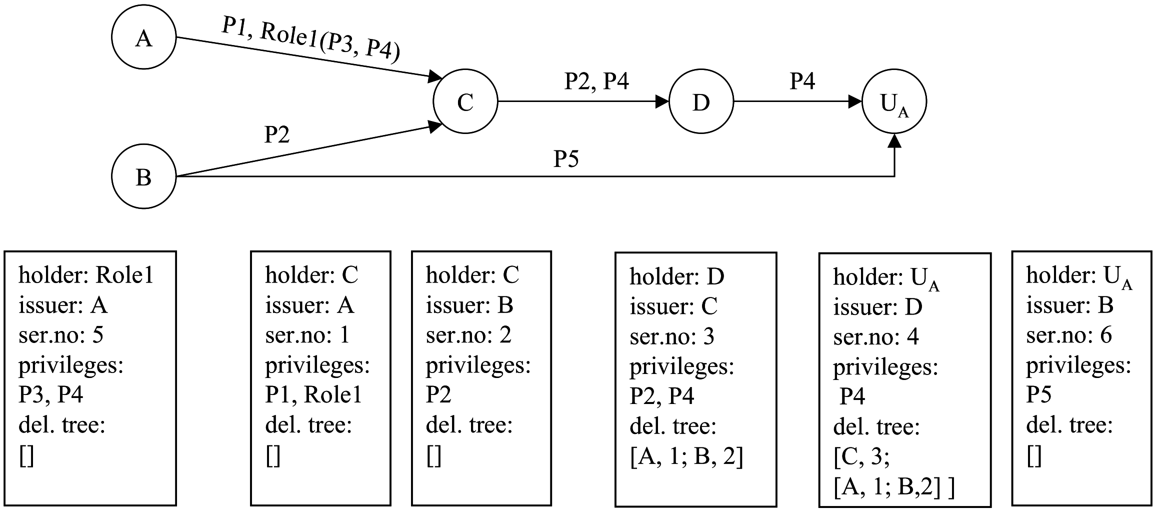
- Multiple ACs and therefore multiple delegation chains can exist for each subject (user or PMA).
- PMAs are autonomous, that means that each PMA may delegate all of its privileges to each other subject.
- PMAs can delegate privileges in one certificate that were assigned to them by several certificates. This implies that delegation chains are paths of a directed tree, a delegation tree.
- PMAs that delegated privileges must inform the resource owners about this act. Resource owners are always known since each AC consists of its delegation history (see below). This supports billing in commercial application scenarios.
- PMAs that delegated privileges must have at least the same privilege at the time of verification. PMAs can delegate privileges for a specific period of time. The beginning of this period can be in the future.
- A PMA can determine whether *all* privileges contained in a particular AC can be delegated to other subjects or not. The model does not allow the prohibition of the delegation of single privileges. If a PMA wants to allow the delegation of a subset of privileges, it should issue a separate AC containing only these privileges.
- If a PMA allows the delegation of the privileges listed in an AC, it can define the maximal length of delegation chains starting with this AC.
- PAMINA does not support ACs issued by multiple PMAs. When multiple privileges controlled by different PMAs are needed to perform a particular action on a resource, separate ACs must be issued by each of these PMAs.

There are two important technical problems with privilege delegation, namely to find and to verify ACs that build a delegation tree. Each AC that belongs to a delegation tree must include back pointers to the ACs in which the issuer was assigned the corresponding privileges. These pointers can be used during the verification process to ensure that the grantor has sufficient privileges. For this purpose [4] recommends the use of the optional field *authorityAttributeIdentifier*, which is a sequence of *IssuerSerial* fields. The *IssuerSerial* field is specified as a pair <*issuer, serialNumber*>. An AC that contains *authorityAttributeIdentifier* may include multiple privileges delegated to the certificate holder by multiple authorities. The *authorityAttributeIdentifier* field can include more than one *IssuerSerial* field if the assignment of the delegated privileges to the issuer authority was done in more than one AC.

Assume that a user has more than one ACs issued by different PMAs. In order to answer the requests of a verifier for the certificates of this user, the P-DB has to find those certificates that include privileges that origin from the verifier's owner PMA. Before responding, the P-DB had to compose all possible delegation trees ending with the user's ACs, in order to find those that contain an AC issued by a PMA that owns the verifier. This could lead to performance problems in systems where delegation is practiced frequently and delegation chains are long. Naturally, the P-DB could just send all certificates issued for the user inclusive delegation trees to the verifier. In this case the verifier would also get ACs that have been issued for the user by other PMAs and therefore should be hidden from this verifier.

In order to make the search for certificates that should be delivered to a verifier more efficient, each AC that belongs to a delegation tree stores information about *all* ACs in the same tree down to the resource owner. In contrast to X.509 the complete delegation tree is stored in the certificates, instead of inserting only the *IssuerSerials* of the direct predecessor certificates. The root of such a delegation tree is the AC itself and the leaves are ACs issued by resource owners. This way the database can easier decide which certificates should be made available to the verifier in a given situation. Note that this structure also supports the verification of parts of a delegation tree by verifiers that are not owned by a PMA which issued one of the ACs in a leaf of this tree. One drawback of this solution is that the size of ACs depends on the height of the delegation trees.

Fig. 4 shows an example situation, where user U_A gets the privilege P4 through delegation. Role1 holding privilege P4 has originally been associated with C. As you can see the certificate issued for C by A consists of an empty delegation tree. Then C delegated privilege P2 (that it got from B) together with privilege P4 to D. Therefore, the delegation tree of D's certificate consists of two <*issuer, serialNumber*> pairs. Finally, D delegates privilege P4 to user U_A by issuing the certificate with the serial number 4. As shown in Fig. 4., the included delegation tree has two leaves. Leaf A,1 points to the originator (A) of the privilege P4.



holder: Role1
issuer: A
ser.no: 5
privileges:
P3, P4
del. tree:
[]

holder: C
issuer: A
ser.no: 1
privileges:
P1, Role1
del. tree:
[]

holder: C
issuer: B
ser.no: 2
privileges:
P2
del. tree:
[]

holder: D
issuer: C
ser.no: 3
privileges:
P2, P4
del. tree:
[A, 1; B, 2]

holder: U_A
issuer: D
ser.no: 4
privileges:
P4
del. tree:
[C, 3;
[A, 1; B,2]]

holder: U_A
issuer: B
ser.no: 6
privileges:
P5
del. tree:
[]

Figure 4: Certificates and delegation trees in PAMINA

5.3 Concept of the Privilege Database

As mentioned earlier the Privilege Database (P-DB) stores all I-CVTs, which are periodically updated by the PMAs. The main benefit of this solution is that verifiers checking delegation trees do not have to connect to different databases in order to download certificates issued by different PMAs. Furthermore, PMAs need not to implement services providing verifiers; they just have to maintain their own I-CVTs.

PMAs that want to use the database must be registered before by the P-DB. Registered PMAs may send their updated I-CVTs periodically over an encrypted channel to the P-DB. The P-DB does not maintain older versions of I-CVTs. This task remains in the responsibility of the PMAs. PMAs can either send complete updated I-CVTs, or only the changes (e.g., new signed root) since the last I-CVT update.

As described above, the P-DB provides verifiers basically only with those certificate statements and corresponding certification paths that are needed for a given access control decision. This makes the P-DB more trustworthy for PMAs. An informal description of the protocol for obtaining ACs is given here:

1. Verifier V, which is not necessarily known to the P-DB, establishes a mutual authenticated connection with the P-DB. During the authentication process they agree on the use of a shared key, which will be used to encrypt the whole communication. At the moment we use SSL and the services of the external PKI for this purpose.
2. After that, V sends the name of its owner PMA (PMA_V) to the P-DB.
3. If there is an I-CVT signed by PMA_V and a valid policy certificate issued for V by PMA_V the P-DB sends the policy certificate to the verifier and continues the process. If there is no policy certificate stored in PMA_V's I-CVT the P-DB sends the certification path for the leaf that would contain the certificate and disconnects from V. If the verifier does not receive this non-existence proof it can assume that the P-DB was compromised.
4. In its later requests to the P-DB V sends the identifier (ID) of the user in question or if possible an authenticated (e.g., signed) request of the client.
5. The P-DB searches for all certificates issued for the user by those PMAs listed in the policy certificate of V and returns for each a delegation tree consisting of certificate statements incl. certification paths, where recursively,
   $issuer_i = holders_{i+1}$, for each level i in the tree
   and where $holder_0$ = ID and for some ACs issuer = PMA_V.
   The P-DB includes also referenced role certificates.
   If a certificate was not found the P-DB gives the certification path to the leaf that would contain that AC if it were in the particular I-CVT. This path proves for the verifier that the AC does not exist.
6. The P-DB sends the ACs to V over the encrypted channel. It also sends certification paths from each I-CVT of PMAs listed in the verifier's policy certificate proving that the user with ID has no ACs issued by other PMAs.

5.4 Specification of a verifier using PAMINA

Verifiers as specified in this section can typically be integrated into an application server or an application gateway. The main responsibility of a verifier is to decide if an authenticated user either has access to a particular resource or not. It is in the responsibility of the application to determine the identity of the user (ID) and to correctly formulate a request.

The main tasks that the verifier has to implement are:

- Establishing a secure authenticated connection to the Privilege Database.
- Requesting the relevant certificate(s) from the P-DB. The P-DB delivers all relevant ACs of the user with the attached roles and delegation trees.
- Verification of each certificate.
- Evaluating the delegation trees and roles to determine the set of privileges the user holds.
- Deciding if the approved privileges are sufficient.

Input: Certificate identifier ID and certification path as specified in section 4:
Certification path cp = $(l_0, l_1, \ldots, l_{n-1})$ + root signature, where $l_i = ((k_{i0}, k_{i1}, \ldots k_{it}), (h_{i0}, h_{i1}, \ldots h_{i(t-1)})$
Output: Validity of the certification path for this certificate

```
$i \leftarrow 0$
$k \leftarrow (k_{i0}, k_{i1}, \ldots k_{it})$
$h \leftarrow (h_{i0}, h_{i1}, \ldots h_{it})$
$min\_id = min(min_j(k_{ij}),id)$
$max\_id = max(max_j(k_{ij}),id)$
$d \leftarrow hash(k||h)$

$i \leftarrow i + 1$
while $i < n$
        $k \leftarrow (k_{i0}, k_{i1}, \ldots k_{it})$
        $h \leftarrow (h_{i0}, h_{i1}, \ldots h_{it})$
        pos $\leftarrow$ position of ID in k
         (such that $k_{i\,(pos-1)} < ID \le k_{i\,pos}$)

        if ($k_{ij}$ in k are not sorted)
          or (pos>0 and $min\_id \le k_{i\,(pos-1)}$)
          or (pos<t and $max\_id > k_{i\,pos}$)
                output "Invalid certification path."

        $min\_id = min(min_j(k_{ij}),min\_id)$
        $max\_id = max(max_j(k_{ij}),max\_id)$

        insert d in h at position pos
        $d \leftarrow hash(k||h)$
        $i \leftarrow i + 1$

if root signature is NOT valid signature for d
        output "Invalid certification path."
else
        if $(k_{i0}, k_{i1}, \ldots k_{it})$ contains ID
                output "Certificate ID is in I-CVT."
        else
                output “Certificate ID is NOT in I-CVT”
```

Figure 5: Algorithm for validating a cert. path

Every single AC is verified using the algorithm above (see Fig. 5). If an AC references a role definition certificate, the verifier must process the AC defining the specified role. The privileges assigned to the role are implicitly assigned to the user and are therefore included among his privileges. If the privileges are delegated to the user by an intermediary PMA, the verifier must ensure that all ACs that belong to the delegation tree are valid. Furthermore, the verifier must check whether the delegation trees follow the rules defined in section 5.2. The verifier must check for example if each PMA that issued a certificate in the delegation tree was authorized to do so and that no PMA delegation privilege is greater than the privilege held by that PMA.

The policies that the verifier follows during these processes are assigned with the policy certificate. The verifier must also check the certificate validity periods. Finally, the verifier checks if the union of all user privileges is sufficient for the context of use.

5.5 Prototypical implementation details

The first prototype of PAMINA is implemented in Java. Java enables to run the system on different platforms, but many components, such as the P-DB, would need much better performance. The PKI we use for the management of PKCs and for the verification of signatures is the Entrust/PKI v. 4.0 Developer Edition. The services of this PKI are integrated into the system with the use of proprietary developer toolkits. In PAMINA each AC is stored as an XML document. We decided to use XML due to its benefits:

- Platform independent standard,
- Many tools for converting and processing XML documents are available,
- Human readable data representation, XML files can be viewed with any text editor.

We have also implemented an administration framework which provides daily management tasks. This framework contains tools that support I-CVT related operations, like creation, signing and updating of the tree and the parsing, modification or deletion (revocation) of single ACs.

Conclusions and future work

In this paper we introduced PAMINA, a system which manages authorization certificates in distributed environments. The system utilizes the high performance and enhanced security of I-CVT, an improvement of a novel certificate management scheme, that has been proposed in [1]. PAMINA can handle multiple I-CVTs each managed by a so-called Privilege Management Authority (PMA). Since the system implements the pull model, it can be integrated in back-end architectures in which (remote) clients need not to be involved in the access control decision process.

An important feature of PAMINA is the support of privilege delegation. Verifiers become more complex in delegation networks, and a large amount of information about the user's relevant privileges has to be collected from a lot of different PMAs. This could lead to large communication costs and has motivated the development of a central database which stores all I-CVTs issued by different PMAs. The database provides only authenticated verifiers with those certificates that they need to check a user's privileges. In our flexible model a PMA can delegate privileges within one certificate that were assigned before by several certificates. Therefore, a certificate belongs to a delegation tree instead of a simple delegation chain. Storing complete delegation trees in certificates is helpful to implement confidential access to certificates in an efficient manner.

Due to the fact that I-CVTs provide proofs for the non-existence of certificates, verifiers can always be sure that they get all existing certificates with relevant privileges of a given user. However, the database storing multiple I-CVTs would be able to hide complete I-CVTs storing relevant certificates from the verifier. In order to solve this problem, all PMAs that delegate privileges inform the PMA(s) that own the resource in question about this act. This way PMAs can configure their verifiers with the use of so-called policy certificates to demand all certificates of a specific user from a limited set of I-CVTs.

Our results show that due to the properties of I-CVTs, this scheme is qualified for privilege management. The first prototype of PAMINA states that the integration of I-CVTs in an operational architecture is possible despite the special requirements that have not originally been considered when this scheme was constructed.

At the moment PAMINA uses the services of an external PKI since digital signatures are needed for the protection of integrity and authenticity of I-CVTs. In the next future approach, we will extend the system and add key management functionality. Furthermore, we plan to implement a version of PAMINA which can manage authorization information based on SDSI/SPKI.

Acknowledgments

This research work was supported by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG). The authors would like to thank to Günther Augustin and David Garces for numerous discussions and useful comments that helped to improve PAMINA.

References

[1] I. Gassko, P. S. Gemmell and P. MacKenzie: Efficient and Fresh Certification, Proceedings of the Conference Public Key Cryptography 2000, v. 1751 of LNCS, pp. 342–353, Springer, 2000

[2] J. Linn and M. Nyström: Attribute Certification: An Enabling Technology for Delegation and Role-Based Controls in Distributed Environments, Proc. of the 4th ACM Workshop on RBAC, pp. 121-130, Fairfax, USA, 1999

[3] R. Housley, W. Ford, W. Polk, D. Solo: Internet X.509 Public Key Infrastructure Certificate and CRL Profile, IETF Request for Comments 2459, January 1999

[4] ITU-T Recommendation X.509: Information Technology – Open Systems Interconnection – The directory: Public-Key and Attribute Certificate Frameworks, February 2001

[5] P. C. Kocher: On Certificate Revocation and Validation, Proceedings of the 2nd International Conference Financial Cryptography, 1465 of LNCS, pp. 172-177, Springer, 1998

[6] R. C. Merkle, A Certified Digital Signature, Advances in Cryptology: CRYPTO '89, 0435 of LNCS, pp. 218-238, Springer, 1989

[7] M. Naor, K. Nissim: Certificate Revocation and Certificate Update, Proceedings of the 7th USENIX Security Symposium, pp. 217-228, San Antonio, USA, 1998

[8] M. Myers, R. Ankney, A. Malpani, S. Galperin and C. Adams: X.509 Internet Public Key Infrastructure On-line Certificate Status Protocol – OCSP, IETF Request for Comments 2560, June 1999

[9] S. Micali: Efficient Certificate Revocation, Technical Report, Massachusetts Institute of Technology, 1996

[10] I. Nestlerode: Implementing EFECT, Master Thesis, Massachusetts Institute of Technology, 2000

[11] A. Buldas, P. Laud, H. Lipmaa: Accountable Certificate Management using Undeniable Attestations, Proceedings of the 7th ACM Conference on Computer and Communication Security, pp. 9-17, Athens, Greece, November 2000

[12] D. E. Knuth: The Art Of Computer Programming, Volume 3, Sorting and Searching, Second Edition. Addison-Wesley, 1998

[13] D. Comer: The Ubiquitous B-Tree, Computing Surveys, Vol. 11., No 2., pp. 121-137, ACM, June 1979

[14] S. Osborn, R. and Q. Munawer: Configuring Role-Based Access Control to Enforce Mandatory and Discretionary Access Control Policies, ACM Transactions on Information and System Security, Vol. 3, No. 2, pp. 85–106, May 2000