12

A new approach for delegation using hierarchical delegation tokens

Yun Ding[1] · *Patrick Horster*[2] · *Holger Petersen*[2]
[1] *Institute of Parallel and Distributed High-Performance Systems*
University of Stuttgart, Breitwiesenstraße 20–22, D-70565 Stuttgart
E-Mail: ding@informatik.uni-stuttgart.de
[2] *Theoretical Computer Science and Information Security*
University of Technology Chemnitz-Zwickau
Straße der Nationen 62, D-09111 Chemnitz, Germany
E-mail: {pho,hpe}@informatik.tu-chemnitz.de

Abstract

In this paper we give a classification of delegation schemes into four main classes. To solve the problem with simply chained tokens in cascaded delegations we introduce the concept of hierarchical delegation tokens. To realize this concept we use the Schnorr signature scheme and self–certified public keys introduced by Girault. We describe the first approach for hierarchical key generation based on an unregarded idea of Günther and the generation of designated verifier signatures. Using these tools, we present efficient delegation schemes for the four main classes, which are efficient in generating and using delegation keys compared with other existing approaches. This is one of the few works, that combines cryptographic algorithms and protocols to benefit for the complexity and the efficiency of the resulting delegation mechanisms.

Keywords

Delegation, access control, distributed systems, hierarchical certificates

1 INTRODUCTION

The advent of distributed, interconnected systems has brought significant change in the way people communicate. Distributed systems enable users to coordinate their activities and to share the resources of the system – hardware, software and data. Users, who are physically located at distant sites can share these resources as if they were located in the same room. This advantage is unfortunately shadowed by security threats, as the number of dishonest users increases with the expansion of the system. One of the concepts to avoid misuse of services and data provided by the system is *delegation of rights*, whereby some users are authorized to act on behalf of other users.

In this paper we point out two important requirements, that delegation schemes must

P. Horster (ed.), *Communications and Multimedia Security II*

satisfy and classify delegation schemes into four classes. To solve the problem with simply chained tokens in a cascaded delegation we introduce the new idea using hierarchical delegation tokens. This leads to our new approach for delegation, which uses new techniques in generating and using delegation keys and is hence very efficient. To realize this approach we use the Schnorr (1991) signature scheme, which is one of the most efficient ElGamal like signature schemes, self-certified public keys introduced by Girault (1991) and the concept of hierarchical key generation first presented by Günther (1989). For every class we present a delegation scheme, which provides cryptographic assurance that a delegation was originated from the named delegator and that the delegated principals are authorized. Hence reliable access control and audit of the delegation are supported with these schemes. It is the first time that protocols for all four classes have been realized utilizing one basic mechanism. Half of our schemes allow a delegator to specify the final principal with cryptographic methods. The security of our schemes relies mainly on the security of the Schnorr signature scheme discussed by Schnorr (1991).

In section 2 we review some basic definitions and point out the requirements of a delegation scheme. In section 3 we present our classification of delegation schemes. In section 4 we classify related work. Section 5 describes the motivations of our new approach. Section 6 reviews the cryptographic algorithms used for the realization of the delegation protocols. Section 7 introduces the protocols for all classes including their subclasses. In the last sections we discuss the efficiency and security of our new schemes.

2 DEFINITIONS

A *principal* is an entity that can be granted access to objects or can make statements affecting access control decisions (Gasser et. al., 1989). *Delegation* is the process whereby a principal in a distributed environment authorizes another principal to act on his behalf. We focus here on the delegation of rights. In a delegation we separate the following participating principals:

- The *delegating principal*, also called the *originator of the delegation* (Hardjono and Ohta, 1994, Varadharjan, Allen, Black, 1991) or *grantor* (Neuman, 1993), who authorizes another principal,
- the *intermediary principals* who are between the delegating and final principal, each of them is also called *delegated principal*,
- the *executor of the delegation* (Hardjono and Ohta, 1994) or *grantee* (Neuman, 1993), who is the last intermediary principal and contacts the final principal to act on behalf of the delegating principal,
- the *final principal*, also called "end point" (Varadharjan et al., 1991) or "end-server" (Neuman, 1993). This is the principal in a delegation who enforces the authorization.

We denote a delegation from principal A to B with $A \Rightarrow B$. A *delegation path* describes a sequence of principals participating the delegation from the delegating principal to the final principal time ordered. A delegation is called *cascaded* if there exists more than one intermediary principal in the delegation path.

Usually the delegating principal gives the delegated principal a *delegation token*, also called *delegation certificate*, which allows him to operate with some (restricted) rights

of the delegating principal. A delegation token consists of the identity of the delegating principal, the privilege attributes $Pr_A(B)$, which include the rights that the delegating principal A grants B, and his restrictions, for example the duration of the delegation or the identity of the delegated principal and a timestamp TS.

In computer-based systems a delegation must be verifiable, that means the used delegation scheme should fulfill the following two requirements:

1. It must be verifiable that a delegation was originated by the delegating principal. This is called the *authentication* problem (Neuman, 1993).
2. The final principal must be able to verify that *all* intermediary principals are authorized to act as delegated principals. We call this the *authorization* problem.

3 CLASSIFICATION OF DELEGATION SCHEMES

The authorization of the delegated principals is based on a secret *key*, e.g. a delegation key shared between the delegating and the authorized delegated principal or his *identity*. This leads to the following classification of delegation schemes:

1. *Key-based:* The authorization of the delegated principal is based on a secret *delegation key* shared between the delegating principal and him. Using a symmetric cryptosystem the delegation key is enclosed in the delegation token. In a public-key cryptosystem there is a keypair (public and private delegation key). The public delegation key is enclosed in the delegation token and the private delegation key is passed through a private channel from the delegating to the delegated principal. In our schemes we use authentic hierarchical delegation keys described in sections 6.2 and 6.5.
2. *Identity-based:* The authorization of the delegated principal is based on his identity. The delegating principal determines the principals who can act on his behalf by embedding their identities into his delegation token. We distinguish *untraceable* delegations and *traceable* delegations.

   (a) In an *untraceable* delegation the delegating principal names a group of his possible delegated principals, whose identities are included in his delegation token. The final principal has to check, whether the executor of the delegation is member of this group. He doesn't know, how the delegation token has reached the executor (if there exists more than one intermediary principal). For example the delegating principal A authorizes a group $\{B, C\}$ to act on his behalf. The final principal, server S, cannot distinguish the two delegation paths $A \Rightarrow B \Rightarrow C \Rightarrow S$ and $A \Rightarrow C \Rightarrow S$. To grant the executor of the delegation C the rights of A, the final server S has only to verify if C is authorized by A.

   (b) In a *traceable* delegation the delegating principal chooses one delegated principal to act on his behalf. This principal can either contact the final principal directly as executor of the delegation or choose another delegated principal to act on behalf of A. (It depends on the rights which B delegated from A, whether B is allowed to delegate further.) The final principal has to learn the identities of all intermediary principals on the delegation path to verify the authorization of the executor of the delegation. For example, the delegating principal A names the principal B to act on

his behalf. B delegates C with the rights he received from A. As C has not been authorized directly by A, the final principal S will not grant C the rights of A unless he is sure about how C was authorized.

If we have more than one intermediary principals, the authorization of the delegated principals can also be key-based from one principal to the next and identity-based from this principal to the next one or vice versa. We call this case *combined delegation.*

If the delegating principal knows the final principal, he can enforce, that only this principal can verify his delegation. If the delegating principal doesn't know or doesn't want to predetermine the final principal, he can allow that any principals can verify his delegation. This aspect divides the delegation schemes into two classes:

1. *Unknown final principal:* The final principal is not fixed by the delegating principal. The executor of the delegation is allowed to select the final principal.
2. *Known final principal:* The final principal is determined by the delegating principal.

Combining these two aspects results in the classification of delegation schemes into four main classes for which we describe detailed protocols in section 7.

4 RELATED WORK

In the last few years there have been many efforts to realize delegation. We classify this work using our classification:

- *Key-based:* The delegation mechanism developed in the Digital Distributed System Security Architecture (DSSA) (Gasser et al., 1989, Gasser and McDermott, 1990) uses keypairs of asymmetric cryptosystems which are generated by the delegated principals. The ECMA standard defines Privilege Attributed Certificates (PACs) whose use is protected with one or several integrated keypairs (control values, protection values) (Kaijser, Parker and Pinkas, 1994). Only the bearers of these control values are allowed to use the PAC. Neuman (1993) designed the restricted proxy mechanism for Kerberos Version 5. He differentiates bearer proxies from delegate proxies. Proxies are delegation tokens. Bearer proxies use delegation keys, also called proxy keys.
- *Identity-based:* Karger (1986) proposes a proxy login mechanism using Girling's (1982) one-time keywords instead of signatures. Each one-time keyword, which is related to the identity of a principal, is provided by a central authentication server. Sollin's (1988) cascaded authentication mechanism uses digital signatures and nested tokens (called passports). Varadharajan et al. (1991) generalize the mechanisms of Karger and Sollin. They present delegation schemes with chained and nested tokens and investigate the problem with simply chained tokens. Delegate proxies from Neuman (1993) are identity-based. All of these identity-based delegation schemes are traceable.
  In SESAME a delegation is realized by forwarding PACs. Only those principals can be delegated principal whose identity is embedded in PAC (Parker, 1991). This delegation is untraceable identity-based.

Neuman (1993) has also proposed a combined delegation mechanism.

5 MOTIVATION OF OUR APPROACH

Figure 1 shows the outline of a delegation. The delegating principal A chooses B as his delegated principal. He authenticates B and gives him a delegation token $DT_{A,B}$ which he has signed. Only if this delegation is key-based, that means the authorization of B is based on a secret delegation key (see section 3), A transmits B confidentially the secret delegation key $DSK_{A,B}$. Then B acts on the behalf of A and wants to access the final principal S. Therefore B shows S the signed token $DT_{A,B}$ and proves that he is authorized.

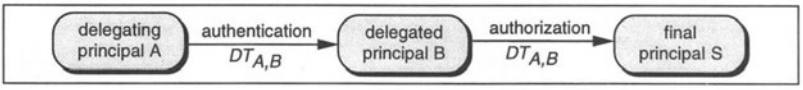

Figure 1 Outline of a delegation

Figure 2 shows the outline of a cascaded delegation. We have omitted the process of authentication and authorization for the sake of clearness.

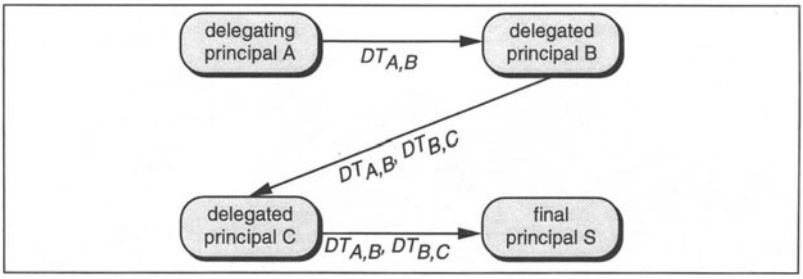

Figure 2 Cascaded delegation

Here we have two different delegation tokens $DT_{A,B}$ and $DT_{B,C}$. A relation between them is necessary, as one of them can otherwise be replaced by an attacker to construct another delegation path (Varadharajan, Allen and Black, 1991): Assume that A trusts B but not C. A delegates B and gives him a signed delegation token $DT_{A,B}$. Acting on behalf of A principal B will not delegate C as delegated principal of A, so the delegation path (1) : $A \Rightarrow B \Rightarrow C$ is not valid. Another principal $\tilde{A}$ trusts both B and C, so the delegation path (2): $\tilde{A} \Rightarrow B \Rightarrow C$ is valid. An attacker who gets the token $DT_{A,B}$ from the delegation $A \Rightarrow B$ and the token $DT_{B,C}$ from delegation path (2) can construct the invalid delegation path (1). There are several solutions to cope with this problem:

1. Varadharajan et al. suggest the use of nested tokens. That means that the token $DT_{A,B}$ is included in the succeeding token $DT_{B,C}$. The token $DT_{B,C}$ will be signed by B, so that an attacker can not replace the token $DT_{A,B}$ undetected. This solution has the disadvantage that the size of the delegation tokens increases with the length of the delegation path.
   They also provide a variation of their solution. Instead of the whole predecessor token $DT_{A,B}$ its unique identifier is included within the next signed token $DT_{B,C}$. However, the identifiers in the whole system must be unique globally. If the token $DT_{A,B}$ and

the token $DT_{\tilde{A},B}$ have the same identifier, the above invalid delegation path can still be constructed.
2. Low and Christianson (1994) introduce the use of signatures to bind two tokens. Instead of the whole token they embed only its signature in the succeeding token, that means the signature of $DT_{A,B}$ by A is a part of the token $DT_{B,C}$.
3. Neuman (1993) proposes the use of delegation keys to bind two tokens. The token $DT_{B,C}$ will be signed with a secret delegation key, which itself (in a symmetric cryptosystem) or the corresponding public delegation key (in an asymmetric cryptosystem) is embedded in the preceding token $DT_{A,B}$.

We have observed that A generates the delegation token $DT_{A,B}$ for B and due to this delegation B generates the token $DT_{B,C}$ for C. Thus the tokens $DT_{A,B}$ and $DT_{B,C}$ are created *hierarchically.* On the other hand we have noticed Günthers (1989) idea of creating identity–based certificates hierarchically using the ElGamal signature scheme. So we investigated the new idea that we solve the problem with simply chained tokens by generating *hierarchical delegation tokens.* To realize this idea, we use delegation keypairs (DSK, DPK) which have the following interesting characteristics:

1. The public delegation keys are self-certified public keys (section 6.2).
2. The secret delegation keys are signature parameters of delegation tokens.
3. They are used to sign the succeeding delegation tokens.
4. The delegation keypairs in a cascaded delegation are hierarchical (section 6.5).

Our solution can be viewed as a combination of the ideas of Low and Christianson and Neuman, as the secret delegation keys are obtained from the signature parameters. Our solution has advantages in generating delegation keys and in the simplicity of our delegation tokens.

6 CRYPTOGRAPHIC ALGORITHMS

In this section we briefly review the cryptographic algorithms which we use as tools for our delegation schemes. They are well known cryptographic primitives which we adopt to the Schnorr (1991) signature scheme to give an example of an efficient implementation. Other ElGamal like signature schemes, e.g. the DSA, can also be used (NIST, 1994). For the efficiency analysis we count only discrete exponentiations as the most expensive operations.

6.1 Schnorr signature scheme

A certification authority Z chooses large primes p, q with $q|(p-1)$, a generator α of a multiplicative subgroup of $\mathbf{Z}_p^*$ with order q, a collision resistant hash function h and publishes p, q, α and h. The signer *Alice* chooses a random number $x_A \in \mathbf{Z}_q^*$ as her secret key and computes $y_A := \alpha^{x_A} \pmod p$ as her public key. To sign the message m Alice chooses a random number $k \in \mathbf{Z}_q^*$. She computes $r := \alpha^k \pmod p$, $s := x_A \cdot h(m, r) + k \pmod q$ and $e := h(m, r)$. The triple $(m, e; s)$ is the signed message. Anyone can verify it by checking the equation (1): $e = h(m, \alpha^s \cdot y_A^{-e} \pmod p)$. We can also use the triple $(m, r; s)$

as the signed message. In this case the verification can be done using the equation (2): $\alpha^s \equiv y^{h(m,r)} \cdot r \pmod{p}$. This scheme can easily be shown to be *equivalent* to the original Schnorr scheme, as e can be easily computed using m and r and on the other side r can be computed using e and s as $r := \alpha^s \cdot y_A^{-e} \pmod{p}$). In our further work we will refer to the later notation as *Schnorr signature.*

The signature generation needs one exponentiation and the verification two exponentiation, which can be computed in parallel (Yen and Laih, 1992). Thus it needs two inversions less than the Digital Signature Algorithm (DSA) (NIST, 1994). The security of this scheme has been discussed by Schnorr (1991).

6.2 Self-certified public keys

In an asymmetric cryptosystem the public keys must be certified by a trustworthy certification authority. This certification authority issues certificates of public keys which are signed using his secret key. Anyone who knows the public key of the certification authority can verify these certificates. We use identity based *self-certified public keys* (Girault, 1991), for identified users with distinguished names by applying the ideas described by Guenther (1989). These keys can be obtained using the Schnorr signature. They are computed as a function of the users identity, the public key of the certification authority and the signature parameter r.

The trusted authority Z signs the identity ID_A of user Alice. Z chooses a random number $k_A \in \mathbf{Z}_q^*$, computes the signature parameters $r_A := \alpha^{k_A} \pmod{p}$ and $s_A := x_Z \cdot h(ID_A, r_A) + k_A \pmod{q}$. The tuple (r_A, s_A) is a signature on the identity ID_A. Alice publishes the parameter r_A and keeps the parameter $x_A := s_A$ as her secret key. Her corresponding public key is computed as $y_A := y_Z^{h(ID_A, r_A)} \cdot r_A \pmod{p}$. This computation needs only one exponentiation modulo p.

Note, that the public key y_A is not necessarily authentic, as the parameter r_A could have been modified by an attacker. The authenticity is verified implicitly by the proper use of the corresponding secret key which is the discrete logarithm of y_A and is only known by Alice. We demonstrate this characteristic of self-certified public keys in the following two subsections.

6.3 Authentication scheme

User *Alice* authenticates herself to verifier *Bob* by proving the knowledge of the discrete logarithm of her self-certified public key y_A (her secret key x_A). She signs a random message $m = H(m_1, m_2)$, where m_1 is chosen randomly by Bob and m_2 randomly by Alice, using the Schnorr signature scheme (where H is a one-way hash function). Bob calculates Alice's self-certified public key $y_A := y_Z^{h(ID_A, r_A)} \cdot r_A \pmod{p}$ and checks the verification equation (2) using the value m_2 submitted by Alice to compute m. If this equation holds, Bob is aware of the authenticity of r_A and of Alice's self-certified public key y_A. Thereby Alice's authenticity is guaranteed. The computational effort is the same as in the Schnorr signature scheme.

6.4 Authentic key-exchange

An authentic session key between users *Alice* and *Bob* who possess self-certified public keys can be generated by applying the ideas described by Guenther (1989).

Setup:
In the setup phase of the protocol Alice and Bob mutually exchange the public data of their self-certified public keys and compute each others self-certified public key. They also agree on a (public) one-way hash function H, which can also be chosen in advances.

Computation of session key:
Alice chooses a random number $a_A \in \mathbf{Z}_q^*$ and computes $b_A := \alpha^{a_A} \pmod p$. b_A is transmitted to *Bob* who chooses a random number $a_B \in \mathbf{Z}_q^*$ and computes $b_B := \alpha^{a_B} \pmod p$, which he transmits to *Alice*. She computes her session key $K_A := H(b_B^{x_A} y_B^{a_A} \pmod p)$ and *Bob* his session key $K_B := H(b_A^{x_B} y_A^{a_B} \pmod p)$. If the public data in the setup phase and the values r_A, b_A and r_B, b_B are transmitted correctly, the two keys K_A and K_B are equal as the following equation holds:

$$K_A := H(b_B^{x_A} y_B^{a_A}) \equiv H(\alpha^{a_B x_A + x_B a_A}) \equiv H(b_A^{x_B} y_A^{a_B}) := K_B \pmod p.$$

Authentication:
To check the equality of K_A and K_B Alice and Bob can exchange random messages encrypted with the keys K_A and K_B using any secure block cipher. This also verifies the authenticity of both self-certified public keys y_A and y_B and the mutual authenticity of Alice and Bob, as the secret keys of both users are necessary for the calculation of the session key.

The computation of the session key needs one exponentiation in the setup phase and three exponentiations during the computation of the session key for each party.

6.5 Hierarchical key generation

If we have many users in a system it is possible that not all of them get their self-certified public key from the same certification authority, but instead they can get their keys from hierarchical ordered authorities like in the X.509 authentication directory (CCITT, 1988, ISO, 1990). The computation of the public keys must be done recursively along several nodes in the certification tree. The idea of hierarchical key generation using identity based public keys has been mentioned by Günther (1989), using the ElGamal signature scheme, but has been unregarded since then. It is the first time to our knowledge, that his ideas are fixed and described more precisely using the Schnorr signature scheme as a basis for the key issuing protocol.

The root Z_0 of the certification tree is a principal that all users in the system trust. In the following we describe how the certification authority Z_i issues the authority Z_{i+1} his key.

Initialization:
The certification authority Z_0 chooses prime numbers p, q with $q|(p-1)$, generator α and his secret key $x_0 \in \mathbf{Z}_q$. Then he computes his public key $y_0 := \alpha^{x_0} \pmod p$.

Issue of keys:
The certification authority Z_i, $i \geq 1$, possesses the triple $(ID_i, r_i; s_i)$ issued from certifica-

tion authority Z_{i-1}, where r_i and ID_i are public and $x_i := s_i$ is his secret key. To issue the triple $(ID_{i+1}, r_{i+1}; s_{i+1})$ for Z_{i+1} he calculates a Schnorr signature on the identity ID_{i+1}.

Hierarchical computation of public keys:
The public key of certification authority Z_i can be computed by any user who knows all public parameters $(ID_j, r_j), j \in [1 : i-1]$, and y_0. He computes

$$\begin{aligned} y_i &\equiv \alpha^{x_i} \equiv y_{i-1}^{h(r_i, ID_i)} \cdot r_i \equiv (y_{i-2}^{h(r_{i-1}, ID_{i-1})} \cdot r_{i-1})^{h(r_i, ID_i)} \cdot r_i \\ &\equiv ((\cdots((y_0^{h(r_1, ID_1)} \cdot r_1)^{h(r_2, ID_2)} \cdot r_2)\cdots)^{h(r_{i-1}, ID_{i-1})} \cdot r_{i-1})^{h(r_i, ID_i)} \cdot r_i \pmod p. \end{aligned}$$

If the user already knows any authentic public key $y_j, j \geq 1$, the computation simplifies for him because he can stop the recurrence when he arrives at this key.

Instead of signing ID, delegation certificates in a cascaded delegation can also be signed in the same way. Thus hierarchical and authentical delegation keys can be obtained.

6.6 Designated verifier signatures

Designated verifier signatures allow only a designated verifier, specified by the signer of the message, to verify the authenticity of the message. Here, we present the first protocol to embed a designated verifier signature into the Schnorr signature scheme in a natural way.

Alice generates a designated verifier signature for the verifier *Bob* by computing the signature parameters $r := y_B^k \pmod p$, where y_B is Bob's public key and $s := x_A \cdot h(m, r) + k \pmod q$. Instead of transmitting the signature (r, s) on the hash value of the message m, Alice transmits $(\tilde{r}, s)$ where $\tilde{r} := \alpha^k \pmod p$. Bob computes $r := \tilde{r}^{x_B} \pmod p$ and verifies the signature by checking the congruence $y_B{}^s \equiv y_A^{h(m,r) \cdot x_B} \cdot r \pmod p$. As he needs his secret key x_B twice no one else can verify this signature. Alice doesn't transmit the value r, as otherwise everyone could compute the Diffie-Hellman key $y_B{}^{x_A} \equiv y_A{}^{x_B} \pmod p$ as $y_B{}^{x_A} := (y_B{}^s \cdot r^{-1})^{h(m,r)^{-1}} \pmod p$, if $h(m, r) \not\equiv 0 \pmod q$. The same designated verifier property of the signature could be achieved by encrypting the signature parameters of the Schnorr signature scheme separately for the verifier Bob. The drawback would be, that this doesn't result in such a natural, efficient design.

7 THE NEW DELEGATION PROTOCOLS

Each principal i in the system has a certificate $(ID_i, r_i; s_i)$ from the trusted certification authority Z, from which he obtains his secret and public keys $(x_i, y_i) := (s_i, \alpha^{s_i} \pmod p)$ as described in section 6.2. They are used for authentication and authentic key exchange in the following.

7.1 Unknown final principal

In this section, the delegating principal A doesn't know the final principal. The authorization of the delegated principal is based either on a secret delegation key, his identity or a combination of both. A protocol for each case is given below.

Suppose, that the delegation path is $A \Rightarrow B \Rightarrow C \Rightarrow S$. It can be divided into three

different phases: 1. The delegating principal A initiates the delegation with B ($A \Rightarrow B$), 2. Intermediary delegation ($B \Rightarrow C$), 3. The enforcement of authorization by the final principal S ($C \Rightarrow S$). These steps will be described separately. They are signed by the delegating principals using the Schnorr signature scheme described in section 6.1, which is not existential forgery unless the hash function h is one-way in the argument m.

Key-based

In this subsection the authorization of the delegated principal is based on a secret delegation key. The exact proceeding is as follows:

1. $A \Rightarrow B$: The delegating principal A generates a delegation token $DT_{A,B} := \{A, Pr_A(B), TS_{A,B}\}$ and computes the Schnorr signature $(DT_{A,B}, r_{A,B}; s_{A,B})$: He chooses a random value $k_{A,B} \in \mathbf{Z}_q^*$, computes $r_{A,B} := \alpha^{k_{A,B}} \pmod p$ and $s_{A,B} := x_A \cdot h(DT_{A,B}, r_{A,B}) + k_{A,B} \pmod q$.
   He obtains a *delegation keypair* $(DSK_{A,B}, DPK_{A,B}) := (s_{A,B}, \alpha^{s_{A,B}} \pmod p)$, whose authenticity can be proved by demonstrating the possession of the secret key $DSK_{A,B}$ with the authentication scheme in section 6.3.
   A and B authenticate themselves using the authentic key exchange protocol in section 6.4. As result they obtain an authentic session key $K_{A,B} := K_A \equiv K_B$, which is needed to transmit the secret delegation key $DSK_{A,B}$ confidentially to B using any secure symmetric cryptosystem. Additionally A transmits the tuple $(DT_{A,B}, r_{A,B})$ to B.
   Note, that the public delegation key $DPK_{A,B} := \alpha^{h(DT_{A,B}, r_{A,B})} \cdot r_{A,B} \pmod p$ is self-certified and the secret delegation key $DSK_{A,B} := s_{A,B}$ is a signature parameter of the token $DT_{A,B}$ (see the first two properties of our delegation keypairs in section 5).
2. $B \Rightarrow C$: B generates a token $DT_{B,C} := \{B, Pr_B(C), TS_{B,C}\}$ and signs it with the *secret delegation key* $DSK_{A,B}$ (see the third property of our delegation keypairs in section 5): He computes the signature parameter $s_{B,C} := DSK_{A,B} \cdot h(DT_{B,C}, r_{B,C}) + k_{B,C} \pmod q$, where $k_{B,C} \in \mathbf{Z}_q^*$ and $r_{B,C} := \alpha^{k_{B,C}} \pmod p$.
   B gets a new delegation keypair $(DSK_{B,C}, DPK_{B,C}) := (s_{B,C}, \alpha^{s_{B,C}} \pmod p)$. B and C authenticate themselves, using the authentic key exchange protocol in section 6.4: During the setup they exchange the values $\{(ID_A, r_A), (DT_{A,B}, r_{A,B})\}$ and (ID_C, r_C). B computes $y_C := y_Z^{h(ID_C, r_C)} \cdot r_C \pmod p$ and C successively the values $y_A := y_Z^{h(ID_A, r_A)} \cdot r_A \pmod p$ and $DPK_{A,B} := y_A^{h(DT_{A,B}, r_{A,B})} \cdot r_{A,B} \pmod p$. They use $DPK_{A,B}$ and y_C as basis for the computation of the session key $K_{B,C}$.
   To transmit the secret delegation key $DSK_{B,C}$ they use the session key $K_{B,C}$ obtained from the authentic key exchange protocol. Additionally B sends C the tuple $(DT_{B,C}, r_{B,C})$.
3. $C \Rightarrow S$: C sends the message $m = \{(ID_A, r_A), (DT_{A,B}, r_{A,B}), (DT_{B,C}, r_{B,C})\}$ to the final principal S from which S computes the values y_A and $DPK_{A,B}$ in the same manner as C did above. Furthermore S computes the public delegation key

$$DPK_{B,C} := DPK_{A,B}^{h(DT_{B,C}, r_{B,C})} \cdot r_{B,C} \pmod p.$$

   Then C proves his authorization by showing the knowledge of the secret delegation key $DSK_{B,C}$, where the authentication scheme in section 6.3 can be used.

Note that these computations of the delegation keypairs $(DSK_{A,B}, DPK_{A,B})$ and $(DSK_{B,C}, DPK_{B,C})$ are based on the ideas of hierarchical key generation described in

section 6.5. Figure 3 summarizes these computations. The keypairs below the names of principals indicate that they are used to sign the identity of A and delegation tokens respectively.

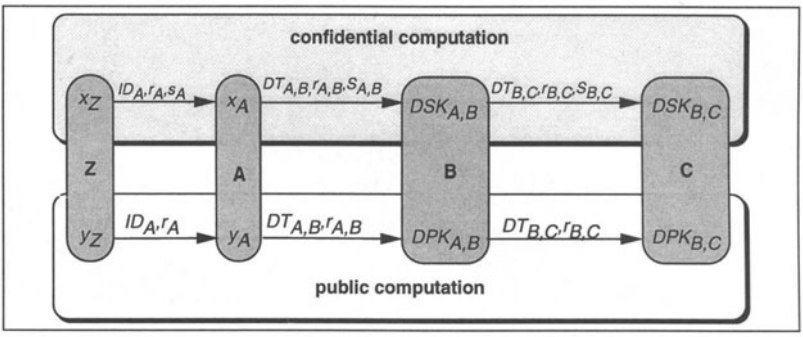

Figure 3 Hierarchical key generation

The tokens $DT_{A,B}$ and $DT_{B,C}$ are transmitted simply chained instead of nested (Varadharajan, Allen, Black, 1991) to S. The key $DSK_{A,B}$ binds the two tokens together,as it is one of the signature parameters of the token $DT_{A,B}$ and it is used to sign the token $DT_{B,C}$. Hence an attacker can replace neither $DT_{A,B}$ nor $DT_{B,C}$ to construct another delegation path. The security of the scheme relies on the security of the Schnorr signature scheme and the authentic key exchange protocol.

Identity-based

In this subsection the authorization of the delegated principal is based on his identity.

Untraceable delegation

In this case the final server S doesn't necessarily learn about the execution of the second step of the protocol.

1. $A \Rightarrow B$: Principal A selects B and C as his delegated principal. He generates a token $DT := \{A, Pr_A(B,C), TS\}$ with the identities ID_B and ID_C included in $Pr_A(B,C)$ and computes a Schnorr signature $(DT, r; s)$ of this token, where s is a *public* signature parameter. He sends this triple to B.
2. $B \Rightarrow C$: If B is not willing to act as executor of the delegation, e.g. he is busy, he forwards simply the triple $(DT, r; s)$ to C. Note, that this can be done using a public channel.
3. $C \Rightarrow S$: C sends S the tuple (ID_C, r_C) to authenticate himself using the authentication scheme in section 6.3. For the delegation he sends the signature $(DT, r; s)$, so that S can verify the signature of DT and whether the identity ID_C is included in $Pr_A(B,C)$ or not.

This delegation is untraceable, as S doesn't learn about from whom C received the delegation token DT.

Traceable delegation
We first describe a scheme which uses the ideas of the key-based and untraceable delegation. Then we investigate one main problem and improve this scheme.

1. $A \Rightarrow B$: Principal A delegates B as in an untraceable delegation. He sends B the signed message $\{DT_{A,B}, r_{A,B}; s_{A,B}\}$, where the identity ID_B is included in $Pr_A(B)$ of $DT_{A,B}$. Also in $Pr_A(B)$ A allows B to delegate another principal, e.g. C, to act on his behalf.
2. $B \Rightarrow C$: B generates a token $DT_{B,C}$ including the identity ID_C in $Pr_B(C)$ and computes a signature using *his secret key* x_B. Thus he obtains $(DT_{B,C}, r_{B,C}; s_{B,C})$. B and C mutually authenticate themselves using the values (ID_B, r_B) and (ID_C, r_C). Afterwards, principal B sends the message $m = \{(ID_A, r_A), (DT_{A,B}, r_{A,B}; s_{A,B}), (ID_B, r_B), (DT_{B,C}, r_{B,C}; s_{B,C})\}$ to C.
3. $C \Rightarrow S$: Principal C authenticates himself with the values (ID_C, r_C) and sends the message m to S. For the verification of the signature of $DT_{A,B}$ the principal S uses the public key y_A computed from the values (ID_A, r_A). He checks the embedded identity ID_B in $DT_{A,B}$ implicitly by verifying the signature of the token $DT_{B,C}$ with the corresponding public key y_B. Then he compares the embedded identity in $DT_{B,C}$ with ID_C. Thus the final principal learns about the whole delegation path.

The simply chained tokens $DT_{A,B}$ and $DT_{B,C}$ above are not related, since $DT_{B,C}$ is signed with the secret key x_B. Thus an attacker can replace $DT_{B,C}$ to construct a new delegation path. We solve this problem using hierarchical delegation tokens as follows:

1. $A \Rightarrow B$: The principal A delegates B as above. He sends B the message $\{(ID_A, r_A), (DT_{A,B}, r_{A,B}; s_{A,B})\}$.
2. $B \Rightarrow C$: B generates the token $DT_{B,C}$ including the identity ID_C in $Pr_B(C)$ and computes the signature using the secret key x_B *and* the signature parameter $s_{A,B}$ of the preceding token $DT_{A,B}$:

$$s_{B,C} \equiv x_B \cdot h(DT_{B,C}, r_{B,C}) + k_{B,C} \cdot s_{A,B} \pmod{q}.$$

   Note this is a Schnorr signature on *two* message blocks $DT_{B,C}$ and $S_{A,B}$. Its security has been analyzed by Horster, Michels and Petersen (1994).
   B authenticates himself as above and sends principal C the message $\tilde{m} := \{(ID_A, r_A), (DT_{A,B}, r_{A,B}; s_{A,B}), (ID_B, r_B), (DT_{B,C}, r_{B,C}; s_{B,C})\}$.
3. $C \Rightarrow S$: C sends S the message $\tilde{m}$. S checks the embedded identity ID_B in $DT_{A,B}$ implicitly by verifying the signature of the token $DT_{B,C}$ with the related public key y_B and the signature parameter $s_{A,B}$ as follows:

$$\alpha^{s_{B,C}} \equiv y_B^{h(DT_{B,C}, r_{B,C})} \cdot {r_{B,C}}^{s_{A,B}} \pmod{p}.$$

We need one extra multiplication for computing the Schnorr signature on $DT_{B,C}$ and one more exponentiation for the verification.

The combined approach

This approach is a combination of a key-based and a traceable delegation. Suppose, that the authorization of B is based on his identity ID_B and the authorization of C is based on a secret delegation key $DSK_{B,C}$. A brief description is as follows:

1. The principal A delegates B as in an identity-based delegation.
2. The principal B delegates C as in the improved version of the identity-based traceable delegation, except that he transmits the value $DSK_{B,C} := s_{B,C}$ confidentially to C.
3. If principal C requires a service from the final principal S, he sends him the message $\{(ID_A, r_A), (DT_{A,B}, r_{A,B}; s_{A,B}), (ID_B, r_B), (DT_{B,C}, r_{B,C})\}$. S checks the identity of B as in the improved version of the identity-based traceable delegation and checks C's knowledge of $DSK_{B,C}$.

Due to the delegation keypairs the final principal doesn't learn the whole delegation path, except the identities of the delegating principal and the delegated principals whose authorization is based on their identities and who are traceable delegated.

Summary

We compare the above approaches and summarize them. Note that, if the delegation is identity-based, we only consider the case of traceable delegation which is more interesting.

1. $A \Rightarrow B$: In all cases the message (ID_A, r_A) has to be transmitted, as the final principal S shall be able to verify the origin of the delegation. If the authorization of B is key-based, A transmits him the signature parameter $s_{A,B}$ confidentially which is the secret delegation key $DSK_{A,B}$. If the authorization of B is identity-based, A transmits $s_{A,B}$ public which is used to bind the token $DT_{A,B}$ with $DT_{B,C}$.
2. $B \Rightarrow C$: The message (ID_B, r_B) has to be transmitted only if the delegation from A to B is identity-based. If the authorization of C is key-based, B has to transmit the delegation key $DSK_{B,C} := s_{B,C}$ confidentially using the authentic key exchange protocol in section 6.4.
3. $C \Rightarrow S$: If the delegation of C by B is key-based C proves his authorization to the final principal S by showing the possession of the secret delegation key $DSK_{B,C}$. If the delegation of C by B is identity-based, S verifies that C knows the secret key x_C corresponding to the self-certified public key which he computes from the tuple (ID_C, r_C).

The following table shows the results of this comparison. "1" means that the protocol in the head of the row is used or the value is transmitted, "0" means that this is not necessary. "C" means that a value is transmitted confidentially by encrypting it with an authentic session key and "P" stands for public transmission. The identity based case in the second row considers only a traceable delegation.

delegation			$A \Rightarrow B$			$B \Rightarrow C$			$C \Rightarrow S$	
			key-	message		key-	message		authori-	message
sect.	$A \Rightarrow B$	$B \Rightarrow C$	exch.	ID_A, r_A	$s_{A,B}$	exch.	ID_B, r_B	$s_{B,C}$	zation	ID_C, r_C
7.1	key	key	1	1	C	1	0	C	$DSK_{B,C}$	0
7.1	id	id	0	1	P	0	1	P	x_C	1
7.1	id	key	0	1	P	1	1	C	$DSK_{B,C}$	0
7.1	key	id	1	1	C	0	0	P	x_C	1

7.2 Known final principal

In this section the delegating principal A predetermines the final principal S. The delegation path is simplified to $A \Rightarrow B \Rightarrow S$, as we only want to explain the main ideas. We use designated verifier signatures described in section 6.6.

The delegating principal A computes the designated verifier signature $(DT_{A,B}, \tilde{r}_{A,B}; s_{A,B})$ of $DT_{A,B}$ by choosing $k_{A,B} \in \mathbf{Z}_q^*$, computing $\tilde{r}_{A,B} := \alpha^{k_{A,B}} \pmod p$, $r_{A,B} := {y_S}^{k_{A,B}} \equiv \tilde{r}_{A,B}^{x_S} \pmod p$ and $s_{A,B} \equiv x_A \cdot h(DT_{A,B}, r_{A,B}) + k_{A,B} \pmod q$. Due to the use of the secret key x_S, only S can compute the value $r_{A,B}$.

- In a *key-based* delegation A sends his delegated principal B the tuple $(DT_{A,B}, \tilde{r}_{A,B})$ public and the value $s_{A,B}$ confidentially from which B obtains a delegation keypair $(DSK_{A,B}, DPK_{A,B}) := (s_{A,B}, {y_S}^{s_{A,B}} \pmod p)$. The final principal S computes the public delegation key $DPK_{A,B} \equiv {y_A}^{h(DT_{A,B}, r_{A,B}) \cdot x_S} \cdot r_{A,B} \pmod p$ and verifies B's knowledge of $DSK_{A,B}$. Note that *only* S can compute this public delegation key $DPK_{A,B}$. Other steps remain the same as in section 7.1.
- In an *identity-based* delegation A sends B the triple $(DT_{A,B}, \tilde{r}_{A,B}; s_{A,B})$. The final principal S verifies the signature on $DT_{A,B}$ by checking the congruence ${y_S}^{s_{A,B}} \equiv {y_A}^{h(DT_{A,B}, r_{A,B}) \cdot x_S} \cdot r_{A,B} \pmod p$. The remaining steps are the same as in section 7.1.

7.3 Revocation

So far we have described how a principal A delegates his rights to another principal B. In real life it happens that A wants to revoke the delegated rights, that means, B should not be able to act on behalf of A anymore. Blacklists of revoked delegation tokens can be used for this situation. The final principal checks against the blacklists before granting B access. If the final principal is not known by the delegating principal A, A contacts a trustworthy central server, who keeps these blacklists, to revoke his delegated rights. Such a central server causes usually a bottleneck in the system, as a high number of final principals have to contact it before granting access. If the final principal S is known by the delegating principal A, A contacts S directly. In this case the final principal S stores a blacklist of revoked tokens.

8 COMPARISON WITH OTHER APPROACHES

The efficiency of our schemes compared with other work relies on the method, how the delegation keys are generated, used and are involved to bind delegation tokens in a cascaded delegation. Especially, we consider the following aspects:

1. *Generation of delegation keys:* In other work delegation keys are randomly generated (Gasser et al., 1989, Gasser and McDermott, 1990, Kaijser et al., 1994, Neuman, 1993, Parker, 1991). The generation of asymmetric keys can be cumbersome. In our scheme we *don't* generate *extra* delegation keys, since the signature parameter s of a token is used as the secret delegation key. The delegating and his delegated principal don't need the corresponding public delegation key.

2. *Use of delegation keys:* In other work the public delegation key of an asymmetric cryptosystem or the secret delegation key of a symmetric cryptosystem is embedded in the delegation token which is signed by the delegating principal. In our schemes a token *doesn't* contain any delegation keys since the public delegation keys are self-certified. Hence the size of our delegation tokens is smaller. Thus the transmission and signing of these tokens in our schemes are less expensive.
3. *Binding of delegation tokens:* To solve the problem with simply chained tokens in identity–based delegations, other works employ nested tokens or unique identifiers of the tokens (Varadharajan et al., 1991) or signatures (Low and Christianson, 1994). These solutions suffer from the larger size of the tokens, because either the whole predecessor token or its unique identifier or its signature is an extra component of the succeeding delegation token. For key–based delegation schemes, called bearer proxy, Neuman (1993) uses delegation keys to bind the succeeding delegation tokens in a cascaded delegation, whereby the succeeding token is signed with the delegation key embedded in the predecessor token. (For identity–based delegation schemes, called delegate proxy, Neuman doesn't investigate this problem.) We combined the ideas of Neuman and Low and Christianson using the advantage of self–certified delegation keys, such that the tokens in both key–based and identity–based schemes are related without additional components in the delegation tokens.

9 SECURITY

The security of the new delegation schemes is based on the security of the underlying signature scheme, which has been analyzed by Schnorr (1991). We can amplify the security of this signature scheme by using the approach described by Cramer and Pedersen (1995). The security of the simply chained tokens relies on the security of the hierarchical self-certified public keys.

10 CONCLUSION

We gave a classification of delegation schemes into four classes. To solve the problem with simply chained tokens in cascaded delegations we introduced the concept of hierarchical delegation tokens. Based on this concept we presented efficient delegation schemes for all classes. Finally, we compared our delegation schemes with existing approaches and discussed their efficiency in generating and using delegation keys. Our approch is a practical contribution to solve the delegation problem in distributed systems.

11 REFERENCES

CCITT, (1988), Recommendation X.509 : *The Directory–Authentication Framework*, Blue Book - Melbourne, Fascicle VIII.8: Data communication networks: directory, International Telecommunication Union, Geneve, 1989, pp. 48 – 81.

R. Cramer, T. Pedersen, (1995), Efficient and provable security amplifications, *CS-R9529*, Computer Science, Dept. of Algorithms and Architecture, CWI, Amsterdam, 9 pages.

M. Gasser, E. McDermott, (1990), An Architecture for Practical Delegation in a Distributed System, *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pp. 20 – 30.

M. Gasser, A. Goldstein, C. Kaufman, B. Lampson, (1989), The Digital Distributed System Security Architecture, *Proceedings of the 1989 National Computer Security Conference*, pp. 305 – 319.

M. Girault, (1991), Self-Certified Public Keys, Lecture Notes in Computer Science 547, *Advances in Cryptology: Proc. Eurocrypt '91*, Berlin: Springer Verlag, pp. 490 – 497.

C. G. Girling, (1982), Object Representation on a Heterogeneous Network, *Operating Systems Review*, Vol. 16, pp. 49 – 59.

C. G. Günther, (1990), An identity based key exchange protocol, Lecture Notes in Computer Science 434, *Advances in Cryptology: Proc. Eurocrypt '89*, Berlin: Springer Verlag, pp. 29 – 37.

T. Hardjono, T. Ohta, (1994), Secure end-to-end delegations in distributed systems, *Computer Communications*, Vol. 17, No. 3, pp. 230 – 238.

P. Horster, M. Michels, H. Petersen, (1994), Meta-ElGamal signature schemes, *Proc. 2. ACM conference on Computer and Communications security*, pp. 96 – 107.

International Organization for Standardization, (1990), ISO/IEC 9594-8, *Information technology – Open systems interconnection — The Directory — Part 8: Authentication framework.*

P. Kaijser, T. Parker, D. Pinkas, (1994), SESAME: The solution to security for open distributed systems, *Computer Communications*, Vol. 17, No. 7, pp. 501 – 518.

P. A. Karger, (1986), Authentication and Discretionary Access Control in Computer Networks, *Computers and Security*, Vol. 5, pp. 314 – 324.

M. R. Low, B. Christianson, (1994), Self Authenticating Proxies, *The Computer Journal*, Vol. 37, No. 5, pp. 422 – 428.

B. C. Neuman, (1993), Proxy – Based Authorization and Accounting for Distributed Systems, *International Conference on Distributed Computing Systems*, pp. 283 – 291.

NIST, (1994), *Federal Information Processing Standards Publication*, National Institute of Standards and Technology, FIPS Pub 186: Digital Signature Standard (DSS), May 19, 20 pages.

T. A. Parker, (1991), A Secure European System for Applications in a Multi-vendor Environment, *Proceedings of the 14th American National Security Conference*, Washington, pp. 505 – 513.

C. P. Schnorr, (1991), Efficient Signature generation by smart cards, *Journal of Cryptology*, Vol. 4, pp. 161 – 174.

K. R. Sollins, (1988), Cascaded Authentication, *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, pp. 156 – 163.

V. Varadharajan, P. Allen, S. Black, (1991), An Analysis of the Proxy Problem in Distributed Systems, *Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy*, pp. 255 – 275.

S. M. Yen, C. S. Laih, (1993), A fast cascade exponentiation algorithm and its application on cryptography, Lecture Notes in Computer Science 718, *Advances in Cryptology: Proc. Auscrypt '92*, Berlin: Springer Verlag, pp. 447 – 458.