to appear in Proc. *3. Conf. on Communications and Multimedia Security, Athens, September 22–23, 1997, Chapman & Hall*

Self-certified keys – Concepts and Applications

Holger Petersen[1*] · *Patrick Horster*[2]
[1] *DMI - GRECC, Ecole Normale Supérieure,*
45, rue d'Ulm, F-75005 Paris, France, E-mail: petersen@dmi.ens.fr
[2] *Computer Science – System Security, University of Klagenfurt,*
A-9020 Klagenfurt, Austria, E-Mail: pho@ifi.uni-klu.ac.at

Abstract

The authenticity of public keys in an asymmetric cryptosystem can be gained in two different ways: either it is verified explicitly after knowing the public key and its certificate, e.g. X.509 certificates, or it is verified implicitly during the use of the keys. The latter concept has been introduced by Girault 1991 as self-certified keys.

In this paper we extend this concept: We show how to issue self-certified keypairs under different trust levels and show how to use them in authentication trees. Then we demonstrate, how a user can switch his keys to enhance the security of his actual secret key against compromising. We illustrate the relevance of all concepts by discussing several useful applications. Among them are delegation of rights, delegated signatures, delegated encryption and electronic voting schemes. Furthermore, we propose a new non-interactive key exchange protocol, that provides backward and forward secrecy of session keys.

Keywords

Cryptography, public key infrastructure, self-certified keys, delegation, electronic voting, key exchange

1 INTRODUCTION

In an asymmetric cryptosystem each user possesses secret and public keys. They might be organized in public directories, such that every user in the system has direct access to the public keys of other users. To ensure the authenticity of published keys, and thus to avoid masquerade attacks, the keys have to be certified. For this purpose, we can distinguish two concepts: *explicit* and *implicit verifiable* certificates. In the first case, the authenticity of the public key y can be verified directly after knowing a guarantee G, also called *certificate*, where G is a digital signature on the user identity ID and his public key y issued by the authority. In the second case, the authenticity is verified at

*The author's work was granted by a postdoctoral fellowship of the NATO Scientific Committee disseminated by the DAAD.

the time, when the key is used for encryption, signature verification, key exchange or any other cryptographic use. This concept has been introduced by Girault 1991 as *self-certified public keys* [Gira91]. The first schemes already using this idea (but not the name) appeared in 1989/90 [Günt89, GiPa90]. Here the guarantee is equal to the public key $G := y$. The user's attributes (ID, x, y) satisfy a computational unforgeable relationship, which is tested implicitly by the proper use of x in any cryptographic protocol.

However, a problem with this general notation is, that x and y are not necessarily related by a desirable mathematical relationship, which is useful in many applications, e.g. as $y := \alpha^x \pmod p$. Therefore, Girault uses the function value $f(G, ID)$ as "real" public key Y, which satisfies such a mathematical relationship. To avoid any notational problems, we use the slightly modified public key $Y := f(G, ID)$ with corresponding secret key x. All users, that have knowledge of the public values (G, ID) can compute Y by applying the function f. If G and ID are correct, so is Y, otherwise its incorrectness will be mentioned during its use. The generation of x, Y and G for the user with identity ID is called *key issuing protocol.* It is executable at four *trust levels*:

- At level 1, the authority knows the secret key x and therefore can impersonate any user at any time without being detected.
- At level 2, the authority doesn't know the user's secret key, but can still impersonate him, by generating a false guarantee G.
- At level 3 even this fraud is no longer possible. Thus, at least the reach of level 3 is desirable for all key issuing protocols.
- At level 4, the authority issues a self-certified public key to a user with pseudonym PS, such that the real identity of the user is hidden to the authority. Nevertheless, all operations using the same pseudonym are linkable for any person.

Overview of the results

We extend the basic concept of self-certified keys (SCKs) by the following properties:

1. We show, how to issue self-certified keys under trust levels one, three and four.
2. We show, how to realize authentication trees using hierarchical self-certified keys.
3. We show, how a user can switch his keypairs to guarantee his privacy even if an old secret key is compromised.

Furthermore, we show the relevance of the above concepts in the following applications:

1. The self-certified keys issued at trust level 1 are used in *proxy cryptosystems* for delegating decryption operations.
2. The self-certified keys issued at trust level 3 are used in *proxy signature* schemes for delegating signature operations.
3. The self-certified keys issued at trust level 4 are used in an efficient *electronic voting* protocol to guarantee the anonymity of the vote. They can also be used in anonymous *electronic cash* systems with revocable anonymity of the coins.
4. The hierarchical self-certified keys are used for the *delegation of rights.*
5. The user controlled key progression is used in a non-interactive *key exchange* protocol providing forward and backward security.

We use the Schnorr signature scheme [Schn89] as underlying signature scheme, which has been proven to be secure under the random oracle model [PoSt96]. Generally, every variant of the Meta-signature scheme can be used instead [HoMP94].

2 THE 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 and a collision resistant hash function h. He publishes p, q, α and h. The signer *Alice* chooses a random number $x_A \in_R \mathbf{Z}_q^*$ as her secret key and computes her public key $y_A := \alpha^{x_A} \pmod p$. To sign the message m Alice chooses a random number $k \in_R \mathbf{Z}_q^*$. 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 $e = h(m, \alpha^s \cdot y_A^{-e} \pmod p)$.

Equivalently, it is possible to use the triple (m,r,s) as signed message instead of (m,e,s). Then the verification equation is check by $\alpha^s \equiv y^{h(m,r)} \cdot r \pmod p$.

3 SELF-CERTIFIED KEYS

We describe the issuing of *self-certified keypairs* for users with distinguished identities at different trust levels. They are computed as a function of the users identity, the public key of the certification authority and the signature parameter r. Therefore, they are somewhat related to *identity-based* public keys, first introduced by Shamir in 1984 [Sham84], where the identity is used directly as public key. As we avoid the direct use of the identity, the computation of the corresponding secret key is more efficient in our setting.

3.1 Basic key issuing protocol

The certification authority Z signs Alice's identity ID_A using the Schnorr signature scheme. Z chooses a random number $k_A \in_R \mathbf{Z}_q^*$, 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 $(r_A, s_A) := \mathcal{S}(x_Z, ID_A)$ is a signature on Alice's identity ID_A. Alice publishes the parameter r_A together with her identity and keeps the parameter $x_A := s_A$ as her *secret key*. Her corresponding *public key* is computable by everyone who knows the public parameters y_Z, ID_A and r_A as

$$y_A := y_Z^{h(ID_A, r_A)} \cdot r_A \pmod p \tag{1}$$

By this computation (1) the obtained key y_A is not authentic, as the public parameter r_A could have been modified by an attacker. Thus its authenticity has to be verified *implicitly* by the proper use of the corresponding secret key x_A, which is only known to Alice. The key issuing protocol reaches trust level 1, as Z gets the knowledge of Alice's secret key x_A and thus might impersonate her.

3.2 Secure key issuing protocol

The above protocol is modified to reach trust level 3 by using a weak blind Schnorr signature, introduced in [HMP95a]. The certification authority chooses $\tilde{k} \in_R \mathbf{Z}_q^*$ as before and computes $\tilde{r}_A := \alpha^{\tilde{k}_A} \pmod p$. She transmits $\tilde{r}_A$ to Alice, who chooses a random $a \in_R \mathbf{Z}_q^*$ and computes $r_A := \tilde{r}_A \cdot \alpha^a \pmod p$. Alice sends ID_A and r_A to the authority who computes the signature parameter $\tilde{s}_A := x_Z \cdot h(ID_A, r_A) + \tilde{k}_A$. This value $\tilde{s}_A$ is transmitted to Alice who obtains her secret key $x_A := \tilde{s}_A + a \pmod q$. The tuple $(r_A, x_A) := \mathcal{S}(x_Z, ID_A)$ is a signature on her identity. Her corresponding public key is computed as

$$y_A := \alpha^{x_A} \equiv y_Z^{h(ID_A, r_A)} \cdot r_A \pmod p. \tag{2}$$

The secret key x_A is hidden to the certification authority Z, as it is blinded by the random value a. Thus, the protocol reaches trust level 3. As only Z is capable to issue valid self-certified keys, the existence of two different valid keys for the same users (e.g. in the case, when Z impersonates Alice) proves, that the authority was cheating. Thus such fraud is detectable by the users.

3.3 Pseudonymous self-certified keys

Alice identifies herself to the certification authority Z to obtain her keypair, but remains anonymous during the use of her self-certified keys. This is possible due to the use of a pseudonym PS_A that Alice uses instead of her real identity ID_A. The pseudonym is not linkable to ID_A but still all actions she performs are traceable, as long as she uses the same pseudonym for them. To obtain the key issuing protocol, we use the blind Schnorr signature scheme [Okam92]. The certification authority Z uses a different certified keypair $(\tilde{x}_Z, \tilde{y}_Z)$ in this protocol to distinguish it from the two protocols above. This avoids fraud of any user, who chooses another user's identity as his pseudonym in order to obtain an indistinguishable self-certified key which he might then use to impersonate this user.

Z chooses a random number $\tilde{k} \in_R \mathbf{Z}_q^*$ and computes $\tilde{r}_A := \alpha^{\tilde{k}_A} \pmod p$. She transmits $\tilde{r}_A$ to Alice, who chooses random $a, b \in_R \mathbf{Z}_q^*$, computes $r_A := \tilde{r}_A \cdot \alpha^a \cdot y_Z^b \pmod p$, $e_A := h(PS_A, r_A)$ and $\tilde{e}_A := e_A + b \pmod q$. Alice sends $\tilde{e}_A$ to the authority, who then computes the signature $\tilde{s}_A := \tilde{x}_Z \cdot \tilde{e}_A + \tilde{k}_A$. The value $\tilde{s}_A$ is transmitted to Alice, who computes her secret key $x_A := \tilde{s}_A + a \pmod q$. Her corresponding public key is computed as $y_A := \alpha^{x_A} \equiv \tilde{y}_Z^{h(PS_A, r_A)} \cdot r_A \pmod p$. The scheme is correct, as

$$\alpha^{x_A} \equiv \alpha^{\tilde{s}_A + a} \equiv \alpha^{\tilde{x}_Z \cdot \tilde{e}_A + \tilde{k}_A + a} \equiv \tilde{y}_Z^{e_A + b} \cdot \tilde{r}_A \cdot \alpha^a \equiv \tilde{y}_Z^{h(PS_A, r_A)} \cdot r_A \pmod p. \tag{3}$$

3.4 Fair pseudonymous self-certified keys

An extension of the pseudonymous key issuing protocol is the assistance of a trustee (a group of trustees) into the protocol, who get(s) knowledge of the relation between ID_A and PS_A. This can be done by registering PS_A at the trustee and obtaining a certificate. This certificate has to be presented together with PS_A. Thus the trustee is able to reveal this relation in the case of a justified suspect against the owner of PS_A to allow law enforcement.

4 HIERARCHICAL SELF-CERTIFIED KEYS

In the X.509 directory the certification authorities and the users are arranged in an authentication tree [CCIT88]. This tree can be described as a tuple $T =< V, E >$. The vertices V are related to the *entities* and the edges E to the *issue of certificates*. Each entity (except the root) has one father but might have several children. The computation of a public key is done recursively along several vertices in the tree.

The authentic public key of the root entity Z_0 of the tree is known by all users in the system. In the following we describe how entity Z_i issues the public key for entity Z_{i+1} using the basic key issuing protocol for clearness. In practice, it should be replaced by the secure key issuing protocol or even the pseudonymous key issuing protocol.

1. *Initialization:* Entity Z_0 chooses prime numbers p, q with $q|(p-1)$, a generator α and his secret key $x_0 \in \mathbf{Z}_q$. Then he computes his public key $y_0 := \alpha^{x_0} \pmod p$.
2. *Key issuing protocol:* Entity Z_i, $i \geq 1$, possesses the triple (ID_i, r_i, s_i) issued by entity 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}. r_{i+1} is transmitted publicly to Z_{i+1} while s_{i+1} must be transmitted confidentially.
3. *Hierarchical computation of public keys:* The public key of entity Z_i is computable by any user who knows all public parameters (ID_j, r_j), for $j \in [1 : i-1]$, and the basic key y_0. He computes

$$\begin{aligned} y_i &\equiv \alpha^{x_i} \equiv y_{i-1}^{h(ID_i, r_i)} \cdot r_i \equiv (y_{i-2}^{h(ID_{i-1}, r_{i-1})} \cdot r_{i-1})^{h(ID_i, r_i)} \cdot r_i \\ &\equiv ((\cdots((y_0^{h(ID_1, r_1)} \cdot r_1)^{h(ID_2, r_1)} \cdot r_2)\cdots)^{h(ID_{i-1}, r_{i-1})} \cdot r_{i-1})^{h(ID_i, r_i)} \cdot r_i \\ &\equiv y_0^{\prod_{j=1}^{i} h(ID_j, r_j)} \cdot \prod_{j=1}^{i} r_j^{\prod_{k=j+1}^{i} h(ID_k, r_k)} \pmod p. \end{aligned} \tag{4}$$

The computation needs $i+1$ exponentiations, which can be realized using a multi-exponentiation in less than $4/3$ the effort of a single exponentiation using a large database of pre-computed products [YeLL94]. If the user already knows an authentic public key $y_j, j \geq 1$, from a previous computation, he can stop the recurrence at level j, which simplifies the computation of (4).

Security considerations

We have to distinguish between *insider* and *outsider attacks*. The insider knows (several) secret keys and wants to modify the public data in a way, that he gets knowledge of more secret keys. This kind of attack is always possible, if the secure key issuing protocol from section 3.2 isn't used.

The *outsider* doesn't have knowledge of any secret key of the certification authorities and wants to modify the public (non-authentic) data in a way, that he is able to compute the secret key of entity Z_j Therefore, he can choose the values k_i, r_i, for $1 \leq i \leq j$ himself and try to solve the equation (4). He can either first choose x_i and then compute k_j, r_j or vice versa. In both cases he will end up with the equation

$$x_i \equiv x_0 \cdot \prod_{j=1}^{i} h(r_j, ID_j) + \sum_{j=1}^{i} \left(k_j \cdot \prod_{k=j+1}^{i} h(r_k, ID_k) \right) \pmod q, \tag{5}$$

which he can't solve without the knowledge of the fixed secret key x_0.

5 USER CONTROLLED KEY PROGRESSION

Besides the hierarchical progression of keys, a user might want to change his keys from time to time. This allows him to renew his keypairs frequently and reduces the risk that he uses compromised keypairs. He might switch his key whenever he has any suspicion. Thus an attacker who knows one secret key should not be able to reveal other keys from that one. We present a basic solution to fit these requirements. More sophisticated approaches with detailed security analysis can be found in [Pete96].

1. *Initialization:*
   The certification authority Z_0 chooses large primes p, q with $q|(p-1)$ and a generator α of a multiplicative subgroup of order q. Then it generates his keypair (x_Z, y_Z). User Alice obtains her *basic* keypair $(x_{A,0}, y_{A,0})$ by one of the above key issuing protocols.
2. *Key progression:*
   Alice can switch her keys either after a fixed, pre-determined time period of length l or after an arbitrary chosen time interval. In the first setting we assume, that she uses the keypair $(x_{A,t}, y_{A,t})$ in the interval $I_t := [t \cdot l : (t+1) \cdot l]$. In the second setting it is even possible that she switches the key after each communication to gain *maximal security.* Alice chooses n random pairs $k_{A,i} \in_R \mathbf{Z}_q^*$ $r_{A,i} := \alpha^{k_{A,i}} \pmod p, i \in [1:n]$. The switching is done analogue to the computation of the hierarchical self-certified public keys. The secret keys $x_{A,i}, i \in [1:n]$ are computed by using the basic secret key $x_{A,0}$. The basic key itself is not used for any communication and is thus not vulnerable to any attack (except tampering from its device). The progression is defined by

   $$x_{A,t} := x_{A,0} \cdot h(ID_A, r_{A,t}) + k_{A,t} \pmod q. \tag{6}$$

3. *Computation of the public key:*
   The public key $y_{A,t}$ is computed using Alice identity ID_A, her basic public key $y_{A,0}$ and the (non-authentic) parameter $r_{A,t}$ by

   $$y_{A,t} := y_{A,0}^{h(ID_A, r_{A,t})} \cdot r_{A,t} \pmod p \tag{7}$$

Security considerations

Suppose, that the secret key $x_{A,0}$ is never compromised. An attacker can't benefit from the knowledge of arbitrary many secret keys $x_{A,i}, 1 \leq i \leq j$, to compute any other secret key $x_{A,j}$, as there exists no relation among these keys except their orign from the same basic key $x_{A,0}$. The computation of $x_{A,0}$ from the knowledge of any $x_{A,i}$ is impossible, as equation (6) contains two unknowns $x_{A,0}$ and $k_{A,i}$. Thus $x_{A,0}$ is undetermined by $x_{A,i}$.

If the secret key $x_{A,0}$ would be compromised by tampering, the authenticity of the self-certified keys is no longer given as an attacker knowing $x_{A,0}$ could publish a tuple (k_C, r_C) and compute his own secret key using equation (6).

6 COMPARISON WITH CERTIFICATE BASED KEYS

We compare the benefits of self-certified keys with those of certificate based keys.

1. *Determine a public key without an authentic public directory:*
   This property is satisfied only partially. Assuming, that the user identities are known to all participants, the (non-authentic) guarantees r have still to be distributed among the users, such that everyone is able to compute the public keys of other users. For this purpose an infrastructure, like a public directory, is needed. The advantage is, that no authentic information is distributed via this infrastructure which simplifies its implementation.
2. *Non-repudiation of public keys:*
   One disadvantage of self-certified keys is their repudiability. For example, if the verification of a digital signature fails using a self-certified public key it is uncertain, whether the signature or the public key is incorrect. Contrary, in the case of a positiv verification the correctness of *both*, the signature and the key is assured, which might be used as evidence of the key's authenticity in future verifications.
3. *Efficient verification of a single certificate:*
   The *explicit* verification of a self-certified public key is not possible, as explained. Its *implicit* verification needs one exponentiation, which might be combined with other computations in the environment where the key is used, e.g. for signature verification. Thus often the exponentiation doesn't need to be counted separately, if it melts together with other exponentiations or can be done with little extra computation.
4. *Efficiency in verification of hierarchical certificates vs. batch signatures:*
   One advantage of hierarchical self-certified keys is that their computation is almost as efficient as the computation of a single self-certified key. If we use certificate based keys, all certificates have to be verified recursively throughout the hierarchy, which can't be accumulated e.g. by using a batch verification mechanism [NMVR94], as batch verification only applies for many signatures generated under the *same* public key. This is not the case in a certification tree, where each certificate is generated using the public key of the father entity.
5. *Importance of key propagation:*
   If the basic secret key of a user is assumed not to be compromised, one might ask, if it is useless to employ key propagation, as the user might always use the uncompromised key for his computations ? Indeed, there is a difference, as a secret key that is used in a cryptographic protocol might be objective to adaptively chosen message attacks (e.g. in a signature or encryption scheme). This is impossible for a protected key that is never directly used in any protocol but serves only for generation of other secret keys.
6. *Security of self-certified keys:*
   The security of the self-certified public keys against existential forgery can be reduced to the security of the underlying signature scheme. For the Schnorr signature scheme it was shown, that in the random oracle model, existential forgery under an adaptively chosen message attack is equivalent to the discrete logarithm problem [PoSt96].

To summarize: Self-certified keys offer *no* structural advantage over certificate based keys, they offer a concept of equal possibilities for which many useful applications are known. These applications are also realizeable using certificate based keys but not in an equally efficient and elegant manner.

7 APPLICATIONS

During the last years many applications of self-certified keys have been investigated, several of them in the context of *delegation*. There have been proposals for delegation of rights [DiHP96], delegation to sign messages [MaUO96], delegation to decrypt messages [MaOk97], delegation to give a vote, delegation to spend money and several others. They are briefly reviewed and improvements are discussed.

7.1 Proxy signatures

A proxy signature allows a designated (group of) person(s), called proxy signer(s) to sign on behalf of the original signer. It should satisfy several requirements, for which we refer to [MaOk96]. Among them are

Strong unforgeability: The proxy signer can create valid proxy signatures for the original signer. Any other third party, including the original signer, is unable to forge a proxy signature.

Strong identifiability: Anyone can identify the proxy signer from a proxy signature.

Strong undeniability: The proxy signer can't repudiate the creation of a valid signature against anyone later.

An efficient scheme, using a variant of the basic key issuing protocol in section 3.1 to generate the proxy signer's keypair, has been proposed in [MaUO96, MaOk96]. It suffers from the fact, that the original signer gets knowledge of the secret key of the proxy signer and thus doesn't satisfy the strong unforgeability, strong distinguishability and strong undeniability properties.

This drawback was repaired in [MaOk96] by adding a second signature using the proxy signers own secret key, which is not very elegant. A better solution would be, to substitute the key issuing protocol by the secure key issuing protocol described above. Then the original signer doesn't get knowledge of the proxy keys and is no longer able to sign on behalf of the proxy signer. To illustrate the solution, we take the certification authority in section 3.2 as original signer and *Alice* as proxy signer. We choose a conventional signature scheme $(\mathcal{S}, \mathcal{V})$ for signing. Then, the protocol looks as follows:

1. *Proxy generation:* The proxy keypair (x_A, y_A) is generated by Z using the protocol in section 3.2. Each user knowing y_Z, ID_A and r_A is able to compute her public key as

$$y_A := y_Z^{h(ID_A, r_A)} \cdot r_A \pmod{p}. \tag{8}$$

2. *Signing by the proxy signer:* Alice generates a signature $\sigma := \mathcal{S}(x_A, m)$ on message m using her proxy secret key x_A. The tuple $(m, \sigma, ID_A, r_A, y_Z)$ is the signed message.
3. *Verification of a proxy signature:* A verifier checks the validity of the signed message by checking

$$\mathcal{V}(y_Z^{h(ID_A, r_A)} \cdot r_A \pmod{p}, m, \sigma) \stackrel{?}{=} true. \tag{9}$$

Possible modifications:

- If the original signer wants to delegate a group G of proxy signers $U_1, \ldots, U_t \in G$ to sign on his behalf, he can include their identities in the hash function h, e.g. $h(ID_1, \ldots, ID_t, r_G)$. The value r_G has to be computed, such that all t proxy signers participate in the computation by using a threshold signature scheme.
- If the original signer doesn't want to include the identity of the proxy signer into the computation, the secure key issuing protocol might be executed by using $h(r_A)$ instead of $h(ID_A, r_A)$ to hide the identity to the verifiers.

7.2 Proxy cryptosystems

Proxy cryptosystems allow a receiver of an encrypted message (the original decryptor) to transform its ciphertext into one for a designated *proxy decryptor*. Once the ciphertext transformation has been executed, the proxy decryptor can compute the plaintext in place of the receiver. A proxy cryptosystem should satisfy two conditions [MaOk97]:

Transformation: Given a ciphertext c_U for an original decryptor U, only he or the creator of c_U are able to transform it into a ciphertext c_P for the proxy decryptor P.

Authorization: Given a ciphertext c_P of an encrypted message m, m can be computed from a proxy p or from information computed from p in polynomial time. Without the knowledge of p, m can't be extracted from c_P.

An efficient solution for a proxy cryptosystem has been proposed in [MaOk97] using the Agnew, Mullin, Vanstone signature scheme to issue the proxy [AgMV90] and the ElGamal cryptosystem [ElGa85]. We modify this scheme by using the Schnorr signature scheme for key issuing. In our notion, Z acts as *original decryptor* and Alice as *proxy decryptor*. The protocol is the following:

1. *Proxy generation:* The proxy keypair (x_A, y_A) is generated by Z as in the last section. It can be computed by all users knowing y_Z, ID_A and r_A as in equation (8).
2. *Encryption by the sender:* A document m is encrypted by choosing a random value $k_c \in_R \mathbf{Z}_q^*$, computing $c_1 := \alpha^{-k_c} \pmod p, c_2 := y_Z^{k_c} \cdot m \pmod p$ and $c_3 := r_A^{k_c} \pmod p$. The value c_3 is an additional value that allows the proxy decryptor Alice to decrypt the message instead of Z. The ciphertext (c_1, c_2, c_3) is send to the receiver Z.
3. *Decryption by the proxy decryptor:* If Z doesn't want to decrypt (c_1, c_2) himself, he forwards the triple (c_1, c_2, c_3) to Alice. Alice decrypts the message by computing

$$\begin{aligned}&(c_1^{x_A} \cdot c_3)^{h(ID_A, r_A)^{-1}} \cdot c_2 \equiv (c_1^{x_Z \cdot h(ID_A, r_A) + k_A} \cdot r_A^{k_c})^{h(ID_A, r_A)^{-1}} \cdot c_2 \\ &\equiv (y_Z^{-k_c \cdot h(ID_A, r_A)})^{h(ID_A, r_A)^{-1}} \cdot c_2 \equiv y_Z^{-k_c} \cdot y_Z^{k_c} \cdot m \equiv m \pmod p.\end{aligned} \tag{10}$$

Instead of allowing exactly one proxy decryptor to decrypt a message m, it is also possible that the proxy key x_A is *shared* among many proxy decryptors. In this case the issuing of the proxy might be modified, such that Alice's identity ID_A is no longer included in the computation of y_A. Furthermore, it is possible to use the secure key issuing protocol from section 3.2 to avoid that Z gets knowledge of Alice's secret key. This might be of interest, if Alice uses the same keypair for several purposes.

7.3 Delegation of rights

Delegation of rights is the process whereby a principal in a distributed environment authorizes another principal to act on his behalf. The *delegating principal* issues the *delegated principal* a delegation token (see [DiHP96] for details). This token consists of the identity of the delegating principal, privilege attributes and restrictions of the delegation. If the delegated principal is not able to execute the task given by the delegator, he might re-delegate it to another principal by issuing a new delegation token. Then there must exist relation between these two token, to avoid misuse by replaying old token or substituting them with tokens obtained from other delegations. This problem has been solved very efficiently by applying the concept of *hierarchical self-certified keys*. Each delegated principal obtains a delegation keypair (comparable to a self-certified keypair) from the precedent delegator, which he uses himself in the case of re-delegation to issue a keypair for his successor. By this method all delegation keys are related and can't be replaced by others on the delegation path. This solves an important problem in the context of delegation in a very efficient manner [DiHP96].

7.4 Electronic voting

Electronic voting schemes can be used to realize elections via the internet. The first electronic voting scheme has been proposed by Chaum [Chau81] using a MIX-net channel for anonymous communications and pseudonyms for user anonymity. Using this approach in combination with pseudonymous self-certified keys allows to obtain an efficient secure voting scheme. Such a scheme should satisfy the following properties [FuOO92]:

Completeness: All valid votes are counted correctly.
Soundness: The dishonest voters can't disrupt the voting.
Privacy: All votes must be secret.
Unreusability: No voter can vote twice.
Eligibiliy: No one, who isn't allowed to vote can vote.
Fairness: Nothing must affect the vote.
Verifiability: No one can falsify the result of the voting.

The model of an electronic voting scheme consists of voters, administrators and a public board. The voters and administrators communicate through an anonymous channel. An efficient solution, satisfying the above properties has been proposed in [HMP95b]. However, this solution employs the need of a blind multisignature scheme. This is avoided in the following by changing the order of the anonymous and the non-anonymous phases in the protocol. We also use an encryption scheme with verifiable multi-decryption as described in [HMP95b].

To describe the scheme we use the following notation: integer t_j is related to one candidate, e.g. his number on a public list, $(\mathcal{S}, \mathcal{V})$ is a conventional signature scheme, $(\mathcal{E}, \mathcal{D})$ is a probabilistic cryptosystem, where $\mathcal{D}$ is the decryption function with verifiable multi-decryption. The protocol is the following:

1. *Voting slip issuing phase:* Each legal voter V_i registers at the administrators by proving his identity and obtains a pseudonymous self-certified keypair (x_i, y_i) using a variant of

the pseudonymous key issuing protocol in section 3.3, where the blind Schnorr signature scheme is replaced by a blind multisignature scheme [HMP95b].

2. *Voting phase:* Voter V_i uses his secret key x_i to sign the vote t_i, e.g. $\sigma_i := \mathcal{S}(x_i, t_i)$. The vote is encrypted as $\mathcal{E}(t_i)$ and send together with the encrypted signature $\mathcal{E}(\sigma_i, y_i)$ via an anonymous channel to the administrators.*
3. *Decryption phase:* After collecting all encrypted votes, the administrators decrypt together the votes t_i and the signatures σ_i, y_i and publish them on the public board. As they use an encryption scheme with verifiable decryption, no administrator can cheat and invalid votes are correctly identified.
4. *Claiming phase:* Each voter checks that all signatures σ_i, y_i on the public board are valid. If a voter V_j recognizes, that his tuple (t_j, σ_j, y_j) wasn't published on the board, he re-sends it anonymously to a third person, who checks (1) that σ_j is a valid signature on t_j and (2) that y_j doesn't already appear on the board. The validity of y_j is checked implicitly by verifying σ_j. If both conditions are satisfied the board will be corrected.
5. *Counting phase:* The administrators, as well as each individual user, can now compute and publish the result of the election. Only the votes t_i with valid signatures σ_i, y_i are counted for this occasion.

The scheme satisfies the above properties. Unfortunately, it doesn't prevent the blackmailing or "buying of votes". A coercer who taps the communication of a voter can force him to give a predetermined vote t_i. He is able to verify that the "correct" vote is encrypted by checking $\mathcal{E}(t_i)$, after forcing the voter to use predetermined random numbers for the probabilistic encryption. A solution to overcome this problem, is to use a tamper-resistant hardware to generate $\mathcal{E}(t_i)$, which can't be manipulated.

7.5 Fair electronic cash

Anonymous electronic cash systems should satisfy the following properties:

Unforgeability: Only authorized entities like banks are able to issue valid digital coins.
Untraceability: The relationship between a digital coin and a user is untraceable.
Unlinkability: Different coins spent by the same user are unlinkable.
Framing: No user or shop can be falsely incriminated.

These properties are achieved by the simple on-line payment system [Chau82, Chau87] outlined in figure 1a. The user obtains a blind signature $\sigma_B := \mathcal{S}(x_B, C_y)$ on his coin C_y for which he knows the pre-image C_x. He spends this coin by generating a signature $\sigma_C := \mathcal{S}(C_x, m)$ on a random message m of the shop, using C_x as secret key.

In this payment system, the generation of the signatures σ_B and σ_C can be chained by generating C_x as *pseudonymous self-certified key* which is used to generate the proxy signature σ_C (see sections 3.3 and 7.1). This gains the shop and the bank respectively over 40% of their computational cost for verifying both signatures, as they need one multi-exponentiation instead of two single exponentiations [YeLL94].

Nevertheless, the system is not secure against theft, e.g. by hacking, or extortion of the user's secret key C_x or more serious the bank's secret key x_B. In order to prevent these attacks the system should satisfy the following additional properties [FuOk96, JaYu96]:

*By using an encryption scheme with multi-decryption it is assured that only all administrators together can decrypt the votes [HMP95b].

User-tracing: The bank and a trustee cooperate to match a spent coin to the user.
Coin-tracing: The bank and a trustee cooperate to compute information that allows the matching of a coin when it is spent or deposited.
Extortion-tracing: The bank and a trustee cooperate in order to compute information that allows the matching of a coin when it is spent or deposited.

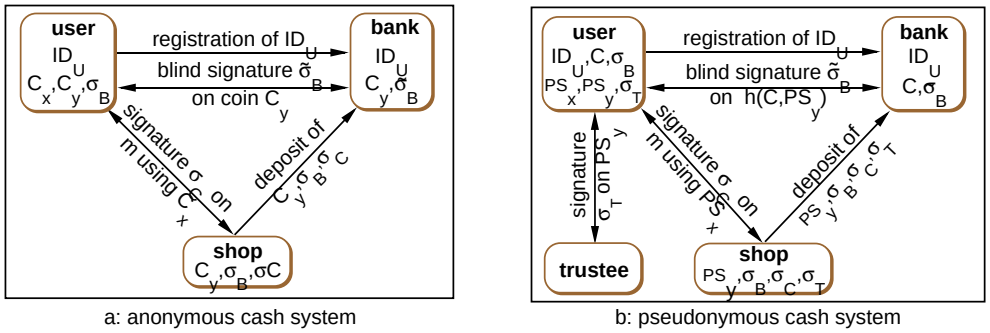

Figure 1 Principles of anonymous and pseudonymous cash systems

These additional properties can be achieved by registering a pseudonymous keypair (PS_x, PS_y) at a trustee, who generates $\sigma_T := \mathcal{S}(x_T, PS_y)$, which is used in the payment protocol as outlined in figure 1b [FuOk96, PePo97]. The user spends a coin C by generating a signature $\sigma_C := \mathcal{S}(PS_x, C, m)$ on a random message m given by the shop. Hence, the trustee can revoke the anonymity of C by linking PS_y to a user's identity. It can be shown, that such a system together with several black- and whitelists prevents all kinds of extortion attacks [PePo97].

In this system the generation of the signatures σ_T and σ_C can be chained by issuing PS_x as self-certified key by the trustee[†] and using it to generate the proxy signature σ_C hereafter (see section 7.1). As in the basic scheme, this gains the shop and the bank about 40% of their computational costs for verifying both signatures [YeLL94]. Additionally, if the coin C is issued as pseudonymous self-certified keypair (C_x, C_y) as above, the verification of all three signatures $\sigma_B, \sigma_T, \sigma_C$ can be done by a single multi-exponentiation by the shop and the bank, which gains each of them about 60% of their computations compared with three single exponentiations.

7.6 Authentic key exchange

The exchange of authentic session keys is one of the main directives in public key cryptography. It is quite easy for two users to compute a common Diffie-Hellman session key [DiHe76] from the knowledge of each others self-certified public key. If they use the same public keys during each session, they will obviously obtain the same session key each time. Thus this simple key exchange protocol offers no *amortized security* [YaSh89], which means that an attacker who knows an old session key can profit from this knowledge to obtain the actual session key of the users. On the other hand the method offers also no *forward secrecy* [DiOW92], that means if the attacker knows an old secret key of one user, he is able to compute the actual session key.

[†] the signature parameter r_T of σ_T is published, $PS_x := s_T$ is transferred confidentially to the user.

User Alice	User Bob
$x_{A,t} := x_{A,0} \cdot h(ID_A, r_{A,t}) + k_{A,t}$	$x_{B,t} := x_{B,0} \cdot h(ID_B, r_{B,t}) + k_{B,t}$
$y_{B,t} := y_{B,0}^{h(ID_B, r_{B,t})} \cdot r_{B,t} \pmod p$	$y_{A,t} := y_{A,0}^{h(ID_A, r_{A,t})} \cdot r_{A,t} \pmod p$
$K_{A,t} := y_{B,t}^{x_{A,t}} \pmod p$	$K_{B,t} := y_{A,t}^{x_{B,t}} \pmod p$
$K_t := h(K_{A,t})$	$K_t := h(K_{B,t})$

Table 1 Non interactive session key

Here, we present a new scheme that offers *both types* of security but allows the computation of a session key in a non-interactive manner. We achieve this result by using the user-controlled key progression, introduced in chapter 5. Using this approach, the security of the key exchange is enhanced, as every session key is only used once. The corresponding user keys are also used only at this time and are switched afterwards. We present the protocol in table 1, where t is the session number between the users (or if preferred, the number of a time interval of fixed length). Alice and Bob obtain the session key

$$K_t := h(y_{B,t}^{x_{A,t}} \pmod p) \equiv h(\alpha^{x_{A,t} \cdot x_{B,t}} \pmod p) \equiv h(y_{A,t}^{x_{B,t}} \pmod p) \equiv h(K_{B,t}), \tag{11}$$

where h is a suitable collision-resistant hash function. Key confirmation is done implicitly during the use of the session key e.g. in an encryption scheme.

Security considerations

The session key K_t offers amortized security with respect to previous session keys $K_i, i \leq t$. An attacker, who knows them cannot compute K_t efficiently. First, he has to invert the one-way hash function h to obtain the values $K_{A,i}$ for all known session keys K_i. Suppose, that he gets the knowledge of this pre-images for several of them by asking an oracle to invert the function for him. Second, he must be able to compute the key:

$$\begin{aligned} K_{A,t} &:= y_{B,t}^{x_{A,t}} \equiv \alpha^{x_{A,t} \cdot x_{B,t}} \equiv \alpha^{(x_{A,0} \cdot h(ID_A, r_{A,t}) + k_{A,t})(x_{B,0} \cdot h(ID_B, r_{B,t}) + k_{B,t})} \\ &\equiv \alpha^{(x_{A,0} \cdot x_{B,0}) \cdot h(ID_A, r_{A,t}) h(ID_B, r_{B,t})} \cdot \alpha^{x_{A,0} \cdot h(ID_A, r_{A,t}) k_{B,t}} \cdot \alpha^{x_{B,0} \cdot h(ID_B, r_B) k_{A,t}} \cdot \alpha^{k_{A,t} k_{B,t}} \\ &\equiv K_{A,0}^{h(ID_A, r_{A,t}) h(ID_B, r_{B,t})} \cdot y_{A,0}^{h(ID_A, r_{A,t}) k_{B,t}} \cdot r_{A,t}^{h(ID_B, r_{B,t}) x_{B,0}} \cdot r_{A,t}^{k_{B,t}} \pmod p. \end{aligned}$$

To be successfully, he needs the knowledge of the pre-image of the virtual "session key" $K_{A,0}$, which is never been used during any communication, and the corresponding public keys $y_{A,0}$ and $y_{B,0}$ were never revealed. Third, the attacker needs to know the secret values $k_{B,t}$ and $x_{B,0}$ (or $k_{A,t}$ and $x_{A,0}$). These values are only known to Alice and Bob, who use them as secret parameters for key progression. Even under the assumption that the attacker got knowledge of some previous secret key $x_{A,i}, x_{B,j}$, as assumed in the forward secrecy model, he is still unable to compute $K_{A,t}$, as he doesn't know $K_{A,0}$ and $k_{A,i}$ or $k_{B,j}$.

8 CONCLUSION

We have presented several extensions of self-certified keys, which make them more competitive compared with certificate-based keys. Then we outlined various applications, such as

proxy signatures, proxy cryptosystems, delegation of rights, electronic voting, cash systems and efficient key exchange protocols. Due to space limitations, these applications were only discussed briefly. For a detailed description, including their underlying trust and security models, the reader is referred to the cited articles.

REFERENCES

[AgMV90] G.B.Agnew, R.C.Mullin, S.A.Vanstone, (1990), Improved digital signature scheme based on discrete exponentiation, *Electronics Letters*, Vol. 26, pp. 1024–1025.

[Chau81] D.Chaum, (1981), Untraceable electronic mail, return addresses, and digital pseudonyms, *Communications of the ACM*, Vol. 24, No. 2, February, pp. 84–88.

[Chau82] D.Chaum, (1982), Blind signatures for untraceable payments, *Advances in Cryptology:* Proc. Crypto '82, Plenum Press, pp. 199–203.

[Chau87] D.Chaum, (1989), Privacy protected payments: Unconditional Payer and/or Payee Anonymity, *Smart Card 2000: The future of IC Cards*, North-Holland, pp. 69–92.

[CCIT88] CCITT, (1989), Recommendation X.509 : The Directory–Authentication Framework, *Blue Book - Melbourne*, Geneve, pp. 48–81.

[DiHe76] W.Diffie, M.Hellman, (1976), New directions in cryptography, *IEEE Transactions on Information Theory*, Vol. IT-22, No. 6, November, pp. 644–654.

[DiOW92] W.Diffie, P.van Oorschot, M.Wiener, (1992), Authentication and Authenticated Key Exchanges, *Designs, Codes and Cryptography*, Vol. 2, pp. 107–125.

[DiHP96] Y.Ding, P.Horster, H.Petersen, (1996), A new approach for delegation using hierarchical delegation tokens, *Proc. 2nd Conference on Computer and Multimedia Security*, Chapman & Hall, pp. 128–143.

[ElGa85] T.ElGamal, (1985), A public key cryptosystem and a signature scheme based on discrete logarithms, *IEEE Trans. on Information Theory*, Vol.30, No.4, pp. 469–472.

[FuOO92] A.Fujioka, T.Okamoto, K.Ohta, (1993), A practical secret voting scheme for large scale elections, LNCS 718, Proc. Auscrypt '92, Springer, pp. 244–260.

[FuOk96] E.Fujisaki, T.Okamoto, (1996), Practical Escrow Cash System, LNCS 1189, *Proc. 1996 Cambridge Workshop on Security Protocols*, Springer, 1997, pp. 33–48.

[GiPa90] M.Girault, J.-C.Pailles, (1990), Un schema basé sur l'identité permettant l'authentification Zero-Knowledge et l'échange de clé authentifié, *Esorics '90*, 12 pages.

[Gira91] M.Girault, (1991), Self-Certified Public Keys, LNCS 547, *Advances in Cryptology:* Proc. Eurocrypt '91, Springer, pp. 490–497.

[Günt89] C.Günther, (1989), An identity based key exchange protocol, LNCS 434, *Advances in Cryptology:* Proc. Eurocrypt '89, Springer, pp. 29–37.

[HoMP94] P.Horster, M.Michels, H.Petersen, (1994), Meta-ElGamal signature schemes, *Proc. 2. ACM Conference on Computer and Communications security*, pp. 96–107.

[HMP95a] P.Horster, M.Michels, H.Petersen, (1995), Hidden signature schemes based on the discrete logarithm problem and related concepts, *Proc. Communications and Multimedia Security*, Chapman & Hall, pp. 162–177.

[HMP95b] P.Horster, M.Michels, H.Petersen, (1995), Blind multisignature schemes and their relevance to electronic voting, *Proc. 11th Annual Computer Security Applications Conference*, IEEE Press, pp. 149–156.

[JaYu96] M.Jakobsson, M.Yung, (1996), Revocable and Versatile Electronic Money, *Proc. 3rd ACM Conference on Computer and Communications Security*, pp. 76–87.

[MaUO96] M.Mambo, K.Usuda, E.Okamoto, (1996), Proxy Signatures for delegating signing operations, *3rd ACM Conf. on Computer and Communication Security*, pp. 48–57.

[MaOk96] M.Mambo, E.Okamoto, (1996), Proxy Signatures: Delegation of the Power to Sign Messages, *IEICE Trans. Fundamentals*, Vol. E79-A, No. 9, Sep., 11 pages.

[MaOk97] M.Mambo, E.Okamoto, (1997), Proxy Cryptosystems: Delegation of the Power to Decrypt Ciphertexts, *IEICE Trans. Fundamentals*, Vol. E80-A, No. 1, January.

[NMVR94] D.Naccache, D.M'Raïhi, S.Vaudenay, D.Raphaeli, (1995), Can D.S.A. be improved ? Complexity trade-offs with the digital signature standard, LNCS 950, *Advances in Cryptology:* Proc. Eurocrypt'94, Springer, pp. 77–85.

[Okam92] T.Okamoto, (1992), Provable secure and practical identification schemes and corresponding signature schemes, LNCS 740, *Advances in Cryptology:* Proc. Crypto '92, Springer, pp. 31–53.

[Pete96] H.Petersen, (1996), Digitale Signaturverfahren auf der Basis des diskreten Logarithmusproblems und ihre Anwendungen, *Dissertation* (Ph.D. thesis), University of Technology Chemnitz-Zwickau, Shaker Verlag, 277 pages.

[PePo97] H.Petersen, G.Poupard, (1997), Efficient scalable fair cash with off-line extortion prevention, *Technical Report* LIENS-97-07, Ecole Normale Supérieure, 33 pages.

[PoSt96] D.Pointcheval, J.Stern, (1996), Security Proofs for Signatures, LNCS 1070, *Advances in Cryptology:* Proc. Eurocrypt'96, Springer, pp. 387–398.

[Schn89] C.P.Schnorr, (1989), Efficient identification and signatures for smart cards, LNCS 435, *Advances in Cryptology:* Proc. Crypto '89, Springer, pp. 239–251.

[Sham84] A. Shamir, (1984), Identity-Based Cryptosystems and Signature Schemes, LNCS 196, *Advances in Cryptology:* Proc. Crypto'84, Springer, pp. 47–53.

[SoNa92] S.von Solms, D.Naccache, (1992), Blind signatures and perfect crimes, *Computers & Security*, Vol. 11, pp. 581–583.

[YaSh89] Y.Yacobi, Z.Shmuely, (1990), On key distribution systems, LNCS 435, *Advances in Cryptology:* Proc. Crypto '89, Springer, pp. 344–355.

[YeLL94] S.-M.Yen, C.-S.Laih, A.K.Lenstra, (1994), Multi-exponentiation, *IEE Proc.-Comput. Digit. Tech.*, Vol. 141, No. 6, November, pp. 325–326.

9 BIOGRAPHY

Patrick Horster received his doctoral degree from Aachen University of Technology (RWTH Aachen), and finished his habitilation at the University of Technology Hamburg Harburg. After several deputy professorships, he got a chair in Theoretical Computer Science and Information Security at the University of Technology Chemnitz in 1994. In March 1997 he accepted a chair in System Security at the University of Klagenfurt. The focus of his current research activities is on design and analysis of cryptographic protocols in general and digital signatures in particular. Apart from cryptography, his research interests include system security, and smart card technology in medical health care systems.

Holger Petersen received his diploma in Computer Science at the University of Karlsruhe in 1993 and his doctoral degree at the University of Technology Chemnitz-Zwickau in July 1996. Since then, he is a postdoctoral researcher in the *Groupe de Recherche en Complexité et Cryptographie* at the Ecole Normale Supérieure in Paris. His main interests are digital signature schemes, their applications and electronic cash systems.