Provably Secure Distributed Schnorr Signatures and a (t, n) Threshold Scheme for Implicit Certificates

D.R. Stinson and R. Strobl
Certicom Corporation
5520 Explorer Drive
Mississauga ON, L4W 5L1
Canada

January 23, 2001

Abstract

In a (t, n) threshold digital signature scheme, t out of n signers must co-operate to issue a signature. We present an efficient and robust (t, n) threshold version of Schnorr's signature scheme. We prove it to be as secure as Schnorr's signature scheme: i.e., existentially unforgeable under adaptively chosen message attacks. The signature scheme is then incorporated into a (t, n) threshold scheme for implicit certificates. We prove the implicit certificate scheme to be as secure as the distributed Schnorr signature scheme.

1 Introduction

Traditional certificates contain a signature on some data, usually a public key and an identity string. To issue a traditional certificate, a Certification Authority (CA) first verifies the authenticity of this data and then simply issues a digital signature on it. The certificate is therefore as secure as the signature scheme: certificates cannot be forged because signatures cannot be forged.

When issuing implicit certificates, the situation is somewhat different. Implicit certificates also contain some data, usually some public reconstruction data and an identity string, but no public key or signature. The public key itself must be computed from the public reconstruction data and the public key of the CA who issued the certificate. Clearly, the advantage of implicit certificates is their size: they only contain some public reconstruction data, where as traditional certificates contain instead a public key and a digital signature. A survey of various types of implicit certificates is given in [9].

In contrast to traditional certificates, where the security lies directly on the underlying signature scheme, there are special security issues concerning implicit certificates. In general, any public reconstruction data and identity string, together with a CA's public key, would yield a public key. However, it should be hard to choose the public reconstruction data and compute the private key corresponding to the implied public key, without knowing the CA's private key. Another issue is that – since one usually uses a slightly

modified signature scheme to issue a certificate – one has to make sure that no information about the CA's or the user's private key is leaked.

We first present a distributed Schnorr signature scheme and prove it to be as secure as the non distributed version, i.e., existentially unforgeable under adaptively chosen message attacks. Second, this scheme is incorporated into the construction of a distributed implicit certificate scheme.

Our digital signature threshold scheme is based on two primitives: Pederson's Verifiable Secret Sharing Scheme and Pederson's multi-party protocol to generate a random shared secret [8, 6]. These primitives are briefly discussed in Section 2. In Section 3 we recall Schnorr's signature scheme [12]. Then we propose in Section 4 a (t, n) threshold version of this signature scheme. We prove the security of the scheme in Section 5, adapting the proof techniques used in [5]. The non-distributed implicit certificate scheme is introduced in Section 6. The (t, n) threshold version of this scheme is presented in Section 7, and a security proof is presented in Section 8.

In all proofs, we use the random oracle model as described in [1]. For all protocols we assume a synchronous communication model, where all players are connected via private channels and a global broadcast channel.

2 Secret Sharing Schemes

2.1 Parameters

We use elliptic curve notation for the discrete logarithm problem. Suppose q is a large prime and G, H are generators of a subgroup of order q of an elliptic curve E. We assume that E is chosen in such a way that the discrete logarithm problem in the subgroup generated by G is hard, so it is infeasible to compute the integer d such that $G = dH$.

2.2 Shamir's Secret Sharing Scheme

In a (t, n) secret sharing scheme, a dealer distributes a secret s to n players $P_1, \cdots, P_n$ in such a way that any group of at least t players can reconstruct the secret s, while any group of less than t players do not get any information about s. In [13], Shamir proposes a (t, n) threshold secret sharing scheme as follows. In order to distribute $s \in Z_q$ among $P_1, ..., P_n$ (where $n < q$), the dealer chooses a random polynomial f over Z_q of degree at most $t - 1$ satisfying $f(0) = s$. Each participant P_i receives $s_i = f(i)$ as his share.

There is one and only one polynomial of degree at most $t - 1$ satisfying $f(i) = s_i$ for t values of i. Therefore, an arbitrary group $\mathcal{P}$ of t participants can reconstruct the polynomial $f()$ by using Lagrange's interpolation formula:

$$f(u) = \sum_{i \in \mathcal{P}} f(i)\omega_i(u) \ , \text{ where } \ \omega_i(u) = \prod_{\substack{j \in \mathcal{P} \\ j \neq i}} \frac{u - j}{i - j} \bmod q.$$

Since it holds that $s = f(0)$, the group $\mathcal{P}$ can reconstruct the secret directly, using the formula

$$s = f(0) = \sum_{i \in \mathcal{P}} f(i)\omega_i \ , \text{ where } \ \omega_i = \omega_i(0) = \prod_{\substack{j \in \mathcal{P} \\ j \neq i}} \frac{j}{j - i} \bmod q.$$

Each ω_i is non-zero and can be easily computed from public information. Note that the constant term of a polynomial of degree at most $t-1$ is not given through $t-1$ equations of the form $f(i) = s_i$. Furthermore, each possible value for the constant term is equally possible. A coalition of $t-1$ players can therefore neither compute the secret nor get any information about it.

2.3 Verifiable Secret Sharing Scheme

A Verifiable Secret Sharing Scheme (VSS) prevents the dealer from cheating. In a VSS, each player can verify his share. If the dealer distributes inconsistent shares, he will be detected. Pedersen presented a non-interactive VSS in [7] which we will use in this paper. His scheme is as follows.

Assume the dealer has a secret $s \in Z_q$ and a random number $s' \in Z_q$, and is committed to the pair (s, s') through public information $C_0 = sG + s'H$. The secret s can be shared among $P_1, ..., P_n$ as follows.

The dealer performs the following steps

1. Choose random polynomials

$$f(u) = s + f_1 u + \cdots + f_{t-1} u^{t-1}, \;\; f'(u) = s' + f'_1 u + \cdots + f'_{t-1} u^{t-1}$$

   where $s, s', f_j, f'_j \in Z_q$. Compute $(s_i, s'_i) = (f(i), f'(i))$ for $i \in \{1, ..., n\}$.

2. Send (s_i, s'_i) secretly to player P_i for $1 \leq i \leq n$.

3. Broadcast the values $C_j = f_j G + f'_j H$ for $1 \leq j \leq t-1$.

Each player P_i performs the following steps

1. Verify that

$$s_i G + s'_i H = \sum_{j=0}^{t-1} i^j C_j. \tag{1}$$

   If this is false, broadcast a *complaint* against the dealer.

2. For each complaint from a player i, the dealer defends himself by broadcasting the value $(f(i), f'(i))$ that satisfies the checking equation (1).

3. Reject the dealer if

   - he received more than t complaints in step 1, or
   - he answered to a complaint in step 2 with values that violate Eq. (1).

Pedersen proved that any coalition of less than t players cannot get any information about the shared secret, provided that the discrete logarithm problem in E is hard (see [7]).

2.4 Generating a random secret

For the key generation phase of our scheme, it is necessary to generate a random shared secret in a distributed way. The early protocol proposed by Feldman [4] has been shown to have a security flaw, and a secure protocol has been proposed in [11]. We will use the secure protocol for our schemes and recall it in the following.

Suppose a trusted dealer chooses r, r' at random, broadcasts $Y = rG$ and then shares r using among the players P_i using Pedersen's VSS as described above. We would like to achieve this situation without a trusted dealer. This can be achieved by the following protocol (see [11] for more details).

Each player P_i performs the following steps

1. Each player P_i chooses $r_i, r_i' \in Z_q$ at random and verifiably shares (r_i, r_i'), acting as the dealer according to Pedersen's VSS described above. Let the sharing polynomials be $f_i(u) = \sum_{j=0}^{t-1} a_{ij} u^j$, $f_i'(u) = \sum_{j=0}^{t-1} a_{ij}' u^j$, where $a_{i0} = r_i, a_{i0}' = r_i'$, and let the public commitments be $C_{im} = a_{im}G + a_{im}'H$ for $i \in \{0, ..., t-1\}$.

2. Let $H_0 := \{P_j | P_j$ is not detected to be cheating at step 1$\}$. The distributed secret value r is not explicitly computed by any party, but it equals $r = \sum_{i \in H_0} r_i$. Each player P_i sets his share of the secret as $s_i = \sum_{j \in H_0} f_j(i) \bmod q$, and the value $s_i' = \sum_{j \in H_0} f_j'(i) \bmod q$.

3. Extracting $Y = \sum_{j \in H_0} r_i G$: Each player in H_0 exposes $Y_i = s_i G$ via Feldman's VSS (see [4]):

   3.1. Each player P_i in H_0 broadcasts $A_{ik} = a_{ik}G$ for $k \in \{0, ..., t-1\}$.

   3.2. Each player P_j verifies the values broadcast by the other players in H_0. Namely, for each $P_i \in H_0$, P_j checks if

   $$f_i(j)G = \sum_{k=0}^{t-1} j^k A_{ik}. \tag{2}$$

   If the check fails for an index i, P_j *complains* against P_i by broadcasting the values $(f_i(j), f_i'(j))$ that satisfy Eq. (1) but do not satisfy Eq. (2).

   3.3. For players P_i who received at least one valid complaint, i.e., values which satisfy Eq. (1) but do not satisfy Eq. (2), the other players run the reconstruction phase of Pedersen's VSS to compute $r_i, f_i(\cdot), A_{ik}$ for $k = 0, ..., t-1$ in the clear[1]. All players in H_0 set $Y_i = r_i G$.

After the executing this protocol, the following equations hold [11]:

$$\begin{aligned} Y &= rG \\ f(u) &= r + a_1 u + ... + a_{t-1} u^{t-1}, \text{ where } a_i = \sum_{j \in H_0} a_{ji}, \quad \text{and} \\ f(i) &= s_i. \end{aligned}$$

[1] Every player in H_0 simply reveals his share of r_i. Each player can then compute r_i by choosing t shares that satisfy Eq. (1)

For convenience, we introduce the following notation for this protocol:

$$(s_1, ..., s_n) \quad \overset{(t,n)}{\longleftrightarrow} \quad (r|Y, a_iG, H_0),\ i \in \{1, ..., t-1\}.$$

This notation means that s_j is player P_j's share of the secret r for each $j \in H_0$. The values a_iG are the public commitments of the sharing polynomial $f(\cdot)$ (they can be computed using public information), and (r, Y) forms a *key pair* (i.e., r is a private key and Y is the corresponding public key). The set H_0 denotes the set of players that have not been detected to be cheating. In the further protocols, we do not need the values $C_i, Y_j, C_{ji}, s'_j, r'$ for $j \in \{1, ..., n\}, i \in \{0, ..., t-1\}$ and therefore we omit these values in the short notation.

3 Schnorr's Signature Scheme

In [10], Schnorr introduced the following signature scheme. Let (x, Y) be a user's key pair, let m be a message and let G be a generator of an elliptic curve group having prime order q. Then a user generates a Schnorr signature on the message m as follows.

1. Select $e \in Z_q$ at random
2. Compute $V = eG$
3. Compute $\sigma = e + h(m, V)x \bmod q$
4. Define the signature on m to be (V, σ)

A verifier accepts a signature (V, σ) on a message m if and only if $\sigma \in Z_q$ and

$$\sigma G = V + h(m, V)Y$$

Schnorr signatures were shown to be existentially unforgeable under adaptively chosen message attacks in the random oracle model, using the forking lemma in [10], provided that the discrete logarithm problem is hard in the group generated by G.

4 A (t, n) Threshold Signature Scheme

In this section, we propose a robust and efficient (t, n) threshold digital signature scheme for Schnorr signatures. We use the primitives presented in Section 2.

Our protocol consists of a key generation protocol and a signature issuing protocol. Let $P_1, ..., P_n$ be a set of signers and let G be a generator of an elliptic curve group of order q.

4.1 Key Generation Protocol

All n signers have to co-operate to generate a public key, and a secret key share for each P_j. They generate a random shared secret according to the protocol presented in Section 2.4. Let the output of the protocol

be

$$(\alpha_1, ..., \alpha_n) \quad \overset{(t,n)}{\longleftrightarrow} \quad (x|Y, b_iG, H_0),\ i \in \{1, ..., t-1\}.$$

For each $j \in H_0$, α_j is the secret key share of P_j, and will be used to issue a partial signature for the key pair (x, Y).

4.2 Signature Issuing Protocol

Let m be a message and let h be a one-way hash function. Suppose that a subset $H_1 \subseteq H_0$ wants to issue a signature. They use the following protocol:

1. If $|H_1| < t$, stop. Otherwise, the subset H_1 generates a random shared secret as described in Section 2.4. Let the output be

$$(\beta_1, ..., \beta_n) \quad \overset{(t,n)}{\longleftrightarrow} \quad (e|V, c_iG, H_2),\ i \in \{1, ..., t-1\}.$$

2. If $|H_2| < k$, stop. Otherwise, each $P_i \in H_2$ reveals

$$\gamma_i = \beta_i + h(m, V)\alpha_i.$$

3. Each $P_i \in H_2$ verifies that

$$\gamma_k G = V + \sum_{j=1}^{t-1} c_j k^j G + h(m, V) \left(Y + \sum_{j=1}^{t-1} b_j k^j G \right) \text{ for all } k \in H_2.$$

   Let $H_3 := \{P_j | P_j \text{ not detected to be cheating at step 3}\}$.

4. If $|H_3| < t$, then stop. Otherwise, each $P_i \in H_3$ selects an arbitrary subset $H_4 \subseteq H_3$ with $|H_4| = t$ and computes σ satisfying $\sigma = e + h(m, V)x$, where

$$\sigma = \sum_{j \in H_4} \gamma_j \omega_j \text{ and } \omega_j = \prod_{\substack{h \neq j \\ h, j \in H_4}} \frac{h}{h - j}.$$

   The signature is (σ, V). To verify the signature, the same formula as in Schnorr's scheme applies:

$$\sigma G = V + h(m, V)Y \text{ and } \sigma \in Z_q.$$

Remarks

(1) The formula used in step 4 to compute σ holds because of the following: Let

$$F_3(u) := F_2(u) + h(m, V)F_1(u).$$

Then it follows that

$$F_3(0) = F_2(0) + h(m, V)F_1(0) = e + h(m, V)x = \sigma.$$

Therefore, by using Lagrange's formula (Section 2.2), the formula holds.

(2) This scheme is robust, i.e., a corrupt signer who does not follow the protocol (by distributing inconsistent shares) will be detected. The random shared secret protocol has been proven to be robust in [11]. The validity of the $\{\gamma_i\}$ is verified at step 3.

(3) The scheme can easily be modified so that a *trusted combiner* calculates the signature, instead of the players. The γ_i's would be sent secretly to the trusted combiner, who proceeds with the verification and the signature generation. In such a scenario, the players would not be able to generate a signature without the combiner.

(4) The only property required by the underlying secret sharing scheme is that it must be homomorphic. This signature scheme could therefore be generalized to non-threshold access structures by using a suitable linear general access structure secret sharing scheme.

5 Security

5.1 Notion of Security

In this section, we show that the proposed (t, n) threshold signature scheme is as secure as Schnorr's signature scheme, i.e., existentially unforgeable under adaptively chosen message attacks in the random oracle model.

We define an adaptively chosen message attack against our (t, n) threshold scheme as follows. An adversary $A_{DistSchnorr}$ is allowed to have the signature issuing protocol executed by any t or more signers to compute signatures on messages of his own choice. He also might corrupt up to $t-1$ arbitrary players. $A_{DistSchnorr}$ then tries to forge a new signature from the signatures he obtained in this way and from his view, where the *view* is everything that $A_{DistSchnorr}$ sees in executing the key generation protocol and the signature issuing protocol.

Let $A_{NormSchnorr}$ be a successful adversary that can break (in the sense of an existential forgery under adaptively chosen message attack) Schnorr's scheme (denoted by $D_{NormSchnorr}$); and let $A_{DistSchnorr}$ be a successful adversary that can break the distributed Schnorr scheme (denoted by $D_{DistSchnorr}$) presented in this paper. To proof the security of our scheme, we will show that given $A_{NormSchnorr}$, one can construct an adversary $A_{DistSchnorr}$, and visa versa. This implies that $D_{DistSchnorr}$ is as secure as $D_{NormSchnorr}$ is.

The basic idea of how to construct $A_{NormSchnorr}$ given the adversary $A_{DistSchnorr}$, a public key Y and a signing oracle goes as follows. $A_{NormSchnorr}$ simulates the roles of the uncorrupted players during all stages of $D_{DistSchnorr}$ – i.e., from the key generation protocol that outputs Y up to the signature issuing protocols for $A_{DistSchnorr}$'s chosen message attack – and lets them interact with $A_{DistSchnorr}$ (see Section 5.3). Because $A_{DistSchnorr}$ cannot distinguish what he sees (i.e., his view) during this simulation from what he would see during a real run of $D_{DistSchnorr}$, he will succeed and output a valid forgery, and therefore so will $A_{NormSchnorr}$.

The next section explains precisely what a view is. We also explain how to build a simulator *SIM* that simulates the honest players during the generation of a distributed random shared secret such that it produces for an arbitrary but given public key Y a view that is indistinguishable for the adversary from a view that would have resulted from real players during a real run of the same protocol outputting Y. This simulator is then used later as a subroutine of a simulator for the adversary's entire view of our threshold signature scheme.

5.2 View

During an arbitrary multi-party protocol, a player will chose values on his own, see public broadcast values and receive private values. We define his view of the protocol to consist of all these values. Notice that in order to simulate the view for a player one does not have to simulate the values which the player chooses on his own.

In the following, we will analyze the adversary's view during the generation of a random shared secret. In particular, the goal is to build a simulator *SIM* that succeeds in the following game. Let B be the index set of corrupted players. The corrupted players P_i for $i \in B$ first run the protocol with real players such that the public value of the random shared secret outputs a random value Y. Now we run the protocol again, but instead of communicating with the real players, the players P_i for $i \in B$ communicate with the simulator. This simulator will now produce messages exactly as the real players do, such that the public value of the random shared secret is Y, and further, the adversary controlling players P_i for $i \in B$ cannot distinguish this simulated view from the view resulting from the real players.

When generating a distributed random shared secret, as explained in Section 2.4, the view of a player P_i would be the following:

the sharing polynomials	$f_i(\cdot), f'_i(\cdot)$
the temporary shares	$f_j(i), f'_j(i)$ for $j \in H_0$
the public commitments	C_{jm}, A_{jm} for $j \in H_0, k \in \{0, ..., t-1\}$
answers on a valid complaint against P_l	$(f_l(j), f'_l(j))$ for $j \in \{1, ..., n\}$,

and the content of his random tape. If an adversary corrupts P_i and P_j, then the adversary's view is $\{\text{view of } P_i\} \cup \{\text{view of } P_j\}$.

Definition 1 *Suppose that a set H_0 of players compute a random shared secret on input (q, G) and produce output Y. Let $\tilde{A}$ be an adversary that corrupts up to $t-1$ players. Let $view(\tilde{A}, G, q, Y)$ denote the view of the adversary for this protocol. Let $VIEW(\tilde{A}, G, q, Y)$ be the random variable induced by $view(\tilde{A}, G, q, Y)$*[2].

Lemma 1 *For any probabilistic polynomial time adversary $\tilde{A}$ there exists a probabilistic polynomial time simulator* SIM *that can compute a random variable* $\text{SIM}(G, q, Y)$ *which has the same probability distribution as $VIEW(\tilde{A}, G, q, Y)$.*

Proof of Lemma 1 Assume that $\tilde{A}$ corrupts players P_i for $i \in B = \{1, ..., t-1\}$. Further, let B' be the index set that denotes the player who publishes inconsistent values A_{im}. Then, $view(\tilde{A}, G, q, Y)$, when generating a random shared secret, is as follows, assuming $H_0 = \{P_1, ..., P_n\}$:

1. The content of the random tape of $\tilde{A}$
2. $f_i(\cdot), f'_i(\cdot)$ for $i \in B$

[2] view(.) contains random variables and static values. VIEW(.) can be regarded as the interpretation of view(.) as one large bit string, so it is basically a random variable.

3. $f_j(i), f'_j(i)$ for $j \in H_0, i \in B$
4. C_{jm} for $j \in H_0, m \in \{0, ..., t-1\}$
5. A_{jm} for $j \in H_0, m \in \{0, ..., t-1\}$
6. $(f_i(j), f'_i(j))$ for $j \in \{1, ..., n\}, i \in B'$

Now we show how to construct a simulator *SIM* that can act in the protocol as the real players, such that the resulting view has the same probability distribution (we use the same simulator as in [11]). Note that *SIM* does not have to compute the sharing polynomials (2) itself since they are chosen by the adversary. The same holds for the content of the random tape (1) which is part of the adversary's internal state that does not have to be simulated.

1. (3, 4) Perform step 1 of the protocol on behalf of the uncorrupted players $P_t, ..., P_n$ exactly as specified in the protocol. This includes receiving and processing the information sent privately and publicly from corrupted players to honest ones.After this step, *SIM* knows all polynomials $f_i(\cdot), f'_i(\cdot)$ for $i \in H_0$ (this holds also for $i \in H_0 \cap B$, since *SIM* received enough consistent shares from these parties to compute their polynomials). In particular, *SIM* knows all the shares $f_i(j), f'_i(j)$, the coefficients a_{ik}, b_{ik} and the public values C_{ik}.
2. (5) When extracting the values $r_i G$, the simulator acts as follows:
   - Compute $A_{ik} = a_{ik} G$ for $i \in H_0 \setminus \{n\}, k \in \{0, ..., t-1\}$
   - Compute $A_{n0} = Y - \sum_{i \in H \setminus \{n\}} A_{i0}$
   - Compute $A_{nk} = \lambda_{k0} A_{n0} + \sum_{i=1}^{t-1} \lambda_{ki} f_n(i) G$ for $k \in \{1, ..., t-1\}$, where λ_{ki}'s are the Lagrange interpolation coefficients of the set H_0.
   - Broadcast A_{ik} for $i \in H_0, k \in \{0, ..., t-1\}$
3. (6) To handle the messages resulting from complaints, *SIM* acts as follows:
   - Perform for each uncorrupted player the verifications of Eq. (2) on the values A_{ik} for $i \in B$, broadcast by the players controlled by the adversary. If the verification fails for some $i \in B, j \in H_0 \setminus B$, broadcast a complaint $(f_i(j), f'_i(j))$. (Notice that the corrupted players can publish a valid complaint only against one another, and there will be no complaints against an honest player that is simulated by *SIM*).
   - For each valid complaint against P_i, perform the reconstruction phase of Pedersen's VSS to compute r_i and Y_i in the clear.

After step 1, the polynomials $f_i(\cdot), f'_i(\cdot)$ for $i \in H_0 \setminus B$ are chosen at random. All associated values $(C_{ik}, f_i(j), f'_i(j), a_{ik}, b_{ik})$ therefore have the exact same probability distribution as in a real run of the protocol.

The broadcasted values A_{ik} are all uniformly random since the corresponding a_{ik} are random. This holds also for the specially computed A_{nk} for $k \in \{0, ..., t-1\}$, since, for each such coefficient, there is at least one random value it depends on. Notice that the fact that these A_{nk}'s are not consistent with the corresponding a_{nk}'s does not appear in the adversary's view: he never sees the a_{nk}'s but only the consistent public commitments of these values.

During the handling of complaints (step 3) there can only be valid complaints against a corrupted server. To reconstruct r_i, *SIM* has to reveal the values $f_i(j), f_i'(j)$ for $j \in H_0 \setminus B$. But *SIM* knows all the polynomials $f_i(\cdot), f_i'(\cdot)$ for $i \in H_0 \setminus B$. Therefore, *SIM* has only to broadcast these values, which will always be consistent with the adversary's view.

A more detailed analysis of the distribution can be found in [11]. The computed view, and the induced random variable $SIM(\tilde{A}, G, q, Y)$, has the same probability distribution as $VIEW(\tilde{A}, G, q, Y)$. □

5.3 Unforgeability

In this section, we will show how to reduce the distributed Schnorr signature scheme to the regular Schnorr signature scheme, and visa versa. This implies that the security of the two schemes is identical.

Definition 2 *Let $A_{NormSchnorr}$ be a probabilistic polynomial time adversary who can ask a signer for valid signatures. By $A_{NormSchnorr}(G, q, Y)$ we denote a random variable which specifies the probability of the event that $A_{NormSchnorr}$ queries $(m_1, m_2, ...,)$ to the signer and outputs $(\tilde{m}, \tilde{\sigma}, \tilde{V})$ (on input (G, q, Y)). The probability is taken over all the coin tosses of $A_{NormSchnorr}$ and the signer.*

Definition 3 *Let $A_{DistSchnorr}$ be a probabilistic polynomial time adversary who can corrupt up to $t-1$ players. He also may have $\geq t$ arbitrary signers issue a signature upon his request. By $A_{DistSchnorr}(G, q|Y)$[3] we denote the random variable that has the probability distribution of $A_{DistSchnorr}$ asking for signatures on $(m_1, m_2, ...,)$ (on input (G, q)) and finally computing $(\tilde{m}, \tilde{\sigma}, \tilde{V})$ under the condition that the key generation protocol outputs Y. The probability is taken over all the coin tosses of $A_{DistSchnorr}$ and the signers.*

Theorem 1 *For any adversary $A_{NormSchnorr}$ against $D_{NormSchnorr}$, there exists an adversary $A_{DistSchnorr}$ against $D_{DistSchnorr}$ such that*

$$Pr[A_{DistSchnorr}(G, q|Y) = (m_1, ..., (\tilde{m}, \tilde{\sigma}, \tilde{V}))] = Pr[A_{NormSchnorr}(G, q, Y) = (m_1, ..., (\tilde{m}, \tilde{\sigma}, \tilde{V}))].$$

(Proof) We show how to construct $A_{DistSchnorr}$ given the adversary $A_{NormSchnorr}$. Suppose the key generation protocol of $D_{DistSchnorr}$ generates Y. $A_{DistSchnorr}$ feeds (G, q, Y) and the content of the random tape of $A_{NormSchnorr}$ into $A_{NormSchnorr}$ and starts $A_{NormSchnorr}$. Whenever $A_{NormSchnorr}$ asks for a signature on a message m, $A_{DistSchnorr}$ has some t signers execute the signature issuing protocol for m and returns the signature (σ, V) to $A_{NormSchnorr}$. Thus, $A_{NormSchnorr}$ can perform his chosen message attack. $A_{DistSchnorr}$ outputs $(\tilde{m}, \tilde{\sigma}, \tilde{V})$ if $A_{NormSchnorr}$ outputs $(\tilde{m}, \tilde{\sigma}, \tilde{V})$. □

Theorem 2 *For any adversary $A_{DistSchnorr}$ against $D_{DistSchnorr}$, there exists an adversary $A_{NormSchnorr}$ against $D_{NormSchnorr}$ such that*

$$Pr[A_{NormSchnorr}(G, q, Y) = (m_1, ..., (\tilde{m}, \tilde{\sigma}, \tilde{V}))] = Pr[A_{DistSchnorr}(G, q|Y) = (m_1, ..., (\tilde{m}, \tilde{\sigma}, \tilde{V}))].$$

[3] $A_{DistSchnorr}(G, q|Y)$ is different from $A_{DistSchnorr}(G, q, Y)$. It contains not only the values G, q, Y, but also $A_{DistSchnorr}$'s view from the key generation protocol. For $A_{NormSchnorr}$ this view is empty, while for $A_{DistSchnorr}$ this is not the case (since he can corrupt $t-1$ signers)

(Proof) We show how to construct $A_{NormSchnorr}$ given the adversary $A_{DistSchnorr}$. Informally, we will show how $A_{NormSchnorr}$ can simulate – with the help of a signing oracle (used in the chosen message attack assumption) – the role of the honest players in $D_{DistSchnorr}$ for a given public key Y. Because $A_{DistSchnorr}$ cannot distinguish this simulation, it will be successful and output a forgery which is a forgery in $D_{NormSchnorr}$, too.

For simplicity, assume $A_{DistSchnorr}$ corrupts players $1, ..., t-1$. Using the techniques described in Lemma 1, $A_{NormSchnorr}$ lets *SIM* execute the key generation protocol for the given public key Y. Next, $A_{NormSchnorr}$ runs $A_{DistSchnorr}$. Whenever $A_{DistSchnorr}$ requests a signature for m_i, $A_{NormSchnorr}$ asks a signer and provides $A_{DistSchnorr}$ with the signature (m_i, σ_i, V_i). $A_{NormSchnorr}$ also has to provide $A_{DistSchnorr}$ with the values he sees during the signature issuing protocol. These values include, in particular, the view resulting from generating a random shared secret and all the $\{\gamma_i\}$. Again, $A_{NormSchnorr}$ lets *SIM* interact with $A_{DistSchnorr}$ during the generation of a random shared secret. As a side effect, *SIM* (and therefore also $A_{NormSchnorr}$) knows $\alpha_1, ..., \alpha_{t-1}, \beta_1, ..., \beta_{t-1}$ and can compute $\gamma_1, ..., \gamma_{t-1}$. Finally, $A_{NormSchnorr}$ computes γ_t as follows. Recall from Section 4.2 that we have

$$\sigma_i = \sum_{j=1}^{t} \gamma_j \omega_j, \text{ where } \omega_j = \prod_{\substack{h=1 \\ h \neq j}}^{t} \frac{h}{h-j}.$$

Hence, γ_t is computed as

$$\gamma_t = \frac{\sigma_i - \sum_{j=1}^{t} \gamma_j \omega_j}{\omega_t}.$$

Now $A_{NormSchnorr}$ feeds $\{\gamma_1, ..., \gamma_t\}$ to $A_{DistSchnorr}$. Since $A_{DistSchnorr}$ now has his whole view, he can perform his adaptive chosen message attack. $A_{NormSchnorr}$ outputs $(\tilde{m}, \tilde{\sigma}, \tilde{V})$ if $A_{DistSchnorr}$ outputs $(\tilde{m}, \tilde{\sigma}, \tilde{V})$. □

6 The Implicit Certificate Scheme

To motivate the (t, n) threshold scheme for implicit certificates, we give a short overview of the non-distributed version of this scheme ([3]). In [3], security proofs for this scheme in the random oracle model are given.

Assume a CA with the key pair (x, Y) issues an implicit certificate to a user. The operation of the scheme is as follows.

1. The user generates a random integer $c \in Z_q$ and computes $V = cG$. Further, he sends V to the CA.
2. The CA authenticates the user. Together, the CA and the user determine an identifier string I_u (containing the user's identity and other information such as, for example, a serial number for the certificate).
3. The CA chooses a random integer $e \in Z_q$, and computes $C = V + eG$ and $\sigma = e + h(I_u, C)x$. Further, the CA sends (I_u, C, σ) to the user.
4. The user computes his private key $SK_u = c + s \bmod q$, and verifies the certificate by checking that following equation holds: $SK_u = C + h(I_u, C)Y$.

When a verifier wants to compute the user's public key from the certificate (I_u, C), following formula applies: $PK_u = C + h(I_u, C)Y$. Note that the equation used to compute σ is exactly Schnorr's signing equation. The only difference from Schnorr's signature scheme is the construction of the point C. Here, this point contains an additive component that the user provides. This is necessary to guarantee that only the user knows his secret key.

7 (t, n) Threshold Scheme for Implicit Certificates

In this section, we incorporate the distributed Schnorr signature scheme into a (t, n) threshold scheme for implicit certificates in the same way as was done in Section 6. In such a scheme, n players $P_1, ..., P_n$, called the *shareholders*, represent a CA with public key PK_0. A group of t shareholders together can reconstruct SK_0 and issue an implicit certificate. Any coalition of less than t shareholders does not have any information about SK_0.

Our scheme consists of three steps. First, the shareholders representing the CA have to generate a key pair. Everybody will know the value of PK_0, while only a coalition of at least t shareholders shall be able to recover SK_0 or issue certificates. Second, the shareholders issue a certificate to a user. Finally, the user verifies if the certificate is valid.

In Section 8, we will give a proof that the presented scheme is as secure as the Schnorr signature scheme. This means that if an adversary could forge an implicit certificate *and* know the corresponding private key, he could also forge a Schnorr signature.

7.1 Key Generation Protocol

We would like to generate a random shared secret SK_0 such that each shareholder P_i who follows the protocol holds a share s_i in this key. Moreover, a coalition of less than t players cannot get any information about SK_0.

This situation corresponds exactly to the generation of a shared secret, as described in Section 2.4. Using the notation introduced in Section 2.4, the situation is as follows:

$$(\alpha_1, ..., \alpha_n) \overset{(t,n)}{\longleftrightarrow} (SK_0 | PK_0, b_i G, H_0),\ i \in \{1, ..., t-1\}.$$

7.2 Certificate Issuing Protocol and Public Key Reconstruction

Suppose a subset $H_1 \subseteq H_0$ wants to issue an implicit certificate.

1. The user selects a random number c_u and sends $V_u = c_u G$ to the shareholders. V_u is called the public request value of the user.

2. If $|H_1| < t$, stop. Otherwise, H_1 generates a random shared secret as shown in Section 2.4. Let the public output be

$$(\beta_1, ..., \beta_n) \overset{(t,n)}{\longleftrightarrow} (e | V, c_i G, H_2),\ i \in \{1, ..., t-1\}.$$

3. If $|H_2| < k$, stop. Otherwise, each $P_i \in H_2$ computes $C = V + V_u$ and reveals

$$\gamma_i = \beta_i + h(I_u, C)\alpha_i. \tag{3}$$

4. Each $P_i \in H_2$ verifies that

$$\gamma_l G = V + \sum_{j=1}^{t-1} c_j i^j G + h(I_u, C) \left(Y + \sum_{j=1}^{t-1} b_j i^j G \right) \text{ for all } l \in H_2. \tag{4}$$

   Let $H_3 := \{P_j | P_j \text{ not detected to be cheating at step 3}\}$.

5. If $|H_3| < t$ stop. Otherwise, each $P_i \in H_3$ selects an arbitrary group $H_4 \subseteq H_3$ with $|H_3| = t$ and computes σ satisfying $\sigma = e + h(I_u, C)x$ by

$$\sigma = \sum_{j \in H_4} \gamma_j \omega_j, \text{ where } \omega_j = \prod_{\substack{h \neq j \\ h,j \in H_4}} \frac{h}{h - j}. \tag{5}$$

   The implicit certificate is (σ, C). At least t shareholders send the implicit certificate to the user.

6. The user computes his private key SK_u as $SK_u = c_u + \sigma$ and verifies the correctness of the certificate by the following equation:

$$SK_u G = C + h(I_u, C)Y \text{ and } \sigma \in Z_q. \tag{6}$$

To reconstruct the public key of the user from the implicit certificate, we use following formula:

$$\sigma PK_u = C + h(I_u, C)Y. \tag{7}$$

Remark A corrupt shareholder might send a wrong certificate $\tilde{\sigma}$ to the user. Since t shareholders send their certificates to the user, the user got at least one valid certificate (since there is at least one honest shareholder among t shareholders). To identify the valid certificate, the user simply checks for each σ if equation (6) holds.

8 Security

8.1 Correctness

We have to verify that the private key SK_u computed by the user corresponds to the public key PK_u implied by the implicit certificate (formula 7). Thus, we have to verify that following formula holds:

$$SK_u G \stackrel{!}{\equiv} C + h(I_u, C)PK_0. \tag{8}$$

Let P ($|P| = t$) be a group of shareholders which have not been detected to be cheating when issuing the certificate. Then we have

$$\begin{aligned} SK_u G &\overset{(5)}{=} (c_u + \sum_{i \in P} \gamma_i \omega_i) G \\ &\overset{(3)}{=} c_u G + \left(\sum_{i \in P} (\beta_i + h(I_u, C)\alpha_i) \right) G\omega_i \\ &= V_u + \sum_{i \in P} (\beta_i \omega_i G + \alpha_i \omega_i h(I_u, C) G) \\ &= V_u + V + h(I_u, C) PK_0 \\ &= C + h(I_u, C) PK_0 \quad \square \end{aligned}$$

8.2 Detectability

We have to verify that every shareholder not following the protocol will be detected.

Key Generation During key generation, we use the protocol described in [11]. This protocol has already been proven to be robust, i.e., players not following the protocol will be detected.

Certificate Issuing First, the players generate a distributed secret with Pedersen's protocol (which is proved to be detectable). Second, they reveal $\{\gamma_i\}$, but these values are verified through equation (4). Finally, they send the calculated certificate to the user. By verifying equation (6), the user can identify the correct certificates.

8.3 Notion of Security in the Random Oracle Model

We assume that we are in the random oracle model (i.e., the hash function is modelled as a random function; see [1]). Let (SK_{CA}, PK_{CA}) be the key pair of the CA (represented through shareholders in case of the distributed implicit certificate scheme). An implicit certificate scheme is *secure* if the following two properties hold:

unforgeability It is hard for an adversary who does not know CA's secret key to forge implicit certificates in such a manner that the adversary knows the corresponding private key

non-impersonating It is hard for CA to obtain the requester's private key provided that the requester followed the protocol.

The term "hard" means that there is no polynomial-time adversary who can solve the task with non-negligible probability. These conditions must hold for adversaries defined as follows.

We define a forging adversary A_f as a probabilistic, polynomial-time turing machine which, on input PK_{CA} does the following:

- it may watch other entities requesting and receiving implicit certificates from the CA

- it may request implicit certificates from the CA
- finally, it produces an implicit certificate and the corresponding private key in time t and with probability p.

We define an impersonating adversary A_i as a probabilistic, polynomial-time turing machine which, on input (PK_{CA}, SK_{CA}) does the following:

- it may act as a CA and issue implicit certificates to requesting entities
- it can produce an implicit certificate and the corresponding private key in time t and with probability p.

An adversary A_f (respectively, A_i) is successful if t is polynomial and p is non-negligible.

8.4 Unforgeability

Let (x, Y) be the (SK, PK) key pair of the CA (represented through shareholders in case of the implicit certificate scheme). Let $D_{NormSchnorr}$ denote Schnorr's signature scheme and $A_{NormSchnorr}$ be a successful adversary against it as defined earlier in Section 5.1. We define a successful adversary $A_{DistCert}$ against the implicit certificate scheme $D_{DistCert}$ as a successful forging adversary as defined in Section 8.3.

One can show that a successful adversary $A_{NormSchnorr}$ is equivalent to a successful adversary $A_{DistCert}$, in the sense that each of them can construct the other one. This implies that the distributed implicit certificate scheme is as secure as Schnorr's signature scheme.

The same proof technique as was used for the distributed Schnorr signature scheme can be applied in a straightforward way. That is, one can show how to simulate the view of the given adversary without knowing the private key of the shareholders. Since the adversary cannot distinguish a simulated view from an actual view, he will perform his attack and output a forgery. This forgery can then be used to construct the other adversary.

8.5 Non-impersonating

By proving the unforgeability of our scheme, we implicitly proved that the user does not learn the shareholders' private key shares. We also have to show that the shareholders do not learn the user's private key and impersonate the user. But it follows directly from the scheme that if the shareholders could compute the user's private key, then they could compute discrete logarithms. □

8.6 Further Issues

Consider the scenario where a digital signature on a certain message and an implicit certificate authenticating the according verification key are sent to a user. Even though we proved that it is hard to forge an implicit certificate without knowing the CA's secret key such that one knows the corresponding private key, we did not prove that it is hard to forge a digital signature and an implicit certificate such that the public key implied by the certificate just validates the signature.

This is not an issue with traditional certificates. However, whenever implicit certificates are used to authenticate a public key for some application, a specific security proof for the particular application is necessary. For example, in [2], a proof is given in the random oracle model that it is secure to use implicit certificates as authentication for public keys that verify Schnorr signatures.

9 Acknowledgments

We would like to thank Simon Blake-Wilson for his ideas on the general concepts. We also would like to thank Mingua Qu for reviewing the security proofs and for pointing out the special security issues that arise in the context of implicit certificates.

10 Summary

Based upon various secret sharing primitives and Schnorr's signature scheme, we have presented an implicit certificate scheme, a (t, n) threshold signature scheme, and a (t, n) threshold scheme for implicit certificates. All schemes are efficient, robust and provably secure in the random oracle model.

From a practical point of view, implicit certificate schemes have the following drawbacks. We suggest these points as open research problems.

- The implicit certificate schemes itself generate a key pair for the user. Therefore, the schemes cannot be used to generate a public reconstruction data for a given key pair of the user. To the best of our knowledge, no scheme based on the elliptic curve discrete logarithm problem exists that can issue an implicit certificate for a given key pair.
- The implicit certificate schemes produce a key pair which is defined over the same group as the CA's key pair is. Therefore, the security parameters for the certified public keys are always inherited from the certifying CA. This might not always be desirable in practice.

[1] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *First Annual ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[2] D. Brown. Implicitly certifying signatures securely. manuscript.

[3] R. Gallant D. Brown and S. Vanstone. Provably secure implicit certificate schemes. In *Proc. Financial Cryptography '01*, to appear.

[4] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proc. 28th FOCS*, pages 427–437, 1987.

[5] C. Park and K. Kurosawa. New elgamal type threshold digital signature scheme. *IEICE Trans.*, E79-A:86–93, 1996.

[6] T.P. Pedersen. Distributed provers with applications to undeniable signatures. *Lecture Notes in Computer Science (Eurocrypt '90)*, 473:221–238, 1991.

[7] T.P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. *Lecture Notes in Computer Science (Crypto '91)*, 576:129–140, 1992.

[8] T.P. Pedersen. A threshold cryptosystem without a trusted party. *Lecture Notes in Computer Science (Eurocrypt '91)*, 547:522–526, 1992.

[9] L. Pintsov and S. Vanstone. Postal revenue collection in the digital age. In *Proc. Financial Cryptography '00*, 2000.

[10] D. Pointcheval and J. Stern. Security proofs for signature schemes. *Lecture Notes in Computer Science (Eurocrypt '96)*, 1070:387–399, 1996.

[11] H. Krawczyk R. Gennaro, S. Jarecki and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Lecture Notes in Computer Science (Eurocrypt '99)*, 1592:295–310, 1999.

[12] C.P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4:161–174, 1991.

[13] A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.