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

Douglas R. Stinson[1,2] and Reto Strobl[3]

[1] Certicom Corporation
5520 Explorer Drive
Mississauga ON, L4W 5L1, Canada
[2] Department of Combinatorics and Optimization
University of Waterloo
Waterloo Ontario, N2L 3G1, Canada
dstinson@cacr.math.uwaterloo.ca
[3] Department of Computer Science
Swiss Federal Institute of Technology
CH-9478 Zurich, Switzerland
rstrobl@iiic.ethz.ch

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

Threshold Cryptography. Threshold cryptography addresses the issue of performing cryptographic tasks such as signing, encrypting/decrypting etc. in a distributed way. For example, in a (t, n) threshold signature scheme, any set of t players can issue a signature for an arbitrary message while any set of less than t players cannot issue a signature at all (see [8] for a threshold DSS signature scheme). We refer to [4, 9] for a detailed survey of threshold cryptography.

Certificates. Certificates provide a way to authenticate data, usually a public key. They can be realized using different techniques.

Traditional certificates contain an identity string, the public key, and a signature on these values. To issue a traditional certificate, a Certification Authority (CA) first verifies the authenticity of the public key and the identity string and then 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.

V. Varadharajan and Y. Mu (Eds.): ACISP 2001, LNCS 2119, pp. 417–434, 2001.

Implicit certificates contain an identity string, but neither an explicit public key nor a signature. Instead, they contain public reconstruction data. The public key itself must be computed from the public reconstruction data and the public key of the *CA* who issued the certificate. The advantage of implicit certificates is their size: Besides the identity string, they only contain public reconstruction data, whereas traditional certificates contain a public key *and* a digital signature. This is particularly useful in bandwidth constrained environments such as wireless communication and digital postmarks. A survey of various types of implicit certificates and applications is given in [13].

In contrast to traditional certificates, where the security lies directly on the underlying signature scheme, there are special security issues concerning implicit certificates. In particular, 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.

Our Work. 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.

In all proofs, we assume 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.

Organization. Our digital signature threshold scheme is based on Pedersen's Verifiable Secret Sharing scheme [12] and a multi-party protocol to generate a random shared secret [7]. These primitives are briefly discussed in Section 2. In Section 3 we recall Schnorr's signature scheme [15]. 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 [10]. 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.

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 $P_1, \ldots, P_n$ 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 [16], Shamir proposes a (t, n) threshold secret sharing scheme as follows. In order to distribute $s \in Z_q$ among $P_1, \ldots, P_n$ (where $n < q$), the dealer chooses a random polynomial $f(\cdot)$ over Z_q of degree at most $t-1$ satisfying $f(0) = s$. Each player 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 $f(\cdot)$ by Lagrange's interpolation as follows:

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

Since it holds that $s = f(0)$, the group $\mathcal{P}$ can reconstruct the secret as follows:

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

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 defined through $t-1$ equations of the form $f(i) = s_i$. Furthermore, since the dealer chooses $f(\cdot)$ uniformly at random, every value for the constant term is equally probable. 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 scheme, each player can verify his share. If the dealer distributes inconsistent shares, he will be detected. Pedersen presented a VSS scheme in [11] 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 $P_1, \ldots, P_n$ as follows.

The dealer performs the following steps

1. Choose random polynomials

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

   where $s, s', f_k, f'_k \in Z_q$ for $k \in \{1, \ldots, t-1\}$. Compute $(s_i, s'_i) = (f(i), f'(i))$ for $i \in \{1, \ldots, n\}$.
2. Send (s_i, s'_i) secretly to player P_i for $i \in \{1, \ldots, n\}$.
3. Broadcast the values $C_k = f_k G + f'_k H$ for $k \in \{1, \ldots, t-1\}$.

Each player P_i performs the following steps

1. Verify that
$$s_i G + s'_i H = \sum_{k=0}^{t-1} i^k C_k. \tag{1}$$
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 values $f(i), f'(i)$ that satisfy (1).
3. Reject the dealer if
   - he received at least t complaints in step 1, or
   - he answered to a complaint in step 2 with values that violate (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.

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 [5] has been shown to have a security flaw, and a secure protocol has been proposed in [7], which we will use for our schemes. We recall it in the following.

Suppose a trusted dealer chooses r, r' at random, broadcasts $Y = rG$ and then shares r among the players P_i using Pedersen's VSS scheme. We would like to achieve this situation without a trusted dealer. This can be achieved by the following protocol.

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 secret shares (r_i, r'_i), acting as the dealer according to Pedersen's VSS scheme. Let the sharing polynomials be $f_i(u) = \sum_{k=0}^{t-1} a_{ik} u^k$, $f'_i(u) = \sum_{k=0}^{t-1} a'_{ik} u^k$, where $a_{i0} = r_i, a'_{i0} = r'_i$, and let the public commitments be $C_{ik} = a_{ik} G + a'_{ik} H$ for $k \in \{0, \ldots, t-1\}$.
2. Let H_0 be the index set of players not detected to be cheating at step 1. The distributed secret value r is not explicitly computed by any player, 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 $Y = \sum_{j \in H_0} r_j G$: Each player in H_0 exposes $Y_i = s_i G$ via Feldman's scheme:
   (a) Each player P_i for $i \in H_0$ broadcasts $A_{ik} = a_{ik} G$ for $k \in \{0, \ldots, t-1\}$.
   (b) Each player P_j verifies the values broadcast by the other players in H_0. In particular, every player P_i for $i \in H_0$, 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 (1) but do not satisfy (2).

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

After executing this protocol, the following equations hold:

$$Y = rG$$
$$f(u) = r + a_1 u + \cdots + a_{t-1} u^{t-1}, \text{ where } a_k = \sum_{j \in H_0} a_{jk} \text{ for } k \in \{1, \ldots, t-1\}$$
$$f(j) = s_j \text{ for } j \in H_0.$$

In [7] this scheme has been proven to be robust under the assumption that $t \leq \frac{n}{2}$, i.e., if less than t players are corrupted, the values computed by the honest players satisfy the above equations.

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

$$(s_1, \ldots, s_n) \overset{(t,n)}{\longleftrightarrow} (r|Y, a_k G, H_0), \ k \in \{1, \ldots, t-1\}.$$

This notation means that s_j is player P_j's share of the secret r for each $j \in H_0$. The values $a_k G$ 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 Y_j, C_{jk}, s_j', r' for $j \in H_0, k \in \{0, \ldots, t-1\}$ and therefore we omit these values in the short notation.

3 Schnorr's Signature Scheme

In [14], Schnorr introduced the following signature scheme. Let (x, Y) be a user's key pair, let m be a message, let $h(\cdot)$ be a one-way hash function, 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

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

[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 (1).

In [14], Schnorr signatures were shown to be existentially unforgeable under adaptively chosen message attacks in the random oracle model, using the forking lemma, 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 $P_1, \ldots, P_n$ be the set of players issuing a signature and let G be a generator of an elliptic curve group of order q.

4.1 Key Generation Protocol

All n players 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

$$(\alpha_1, \ldots, \alpha_n) \overset{(t,n)}{\longleftrightarrow} (x|Y, b_k G, H_0),\ k \in \{1, \ldots, t-1\}.$$

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 $h(\cdot)$ be a one-way hash function. Suppose that the players with index set $H_1 \subseteq H_0$ 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

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

2. If $|H_2| < t$, stop. Otherwise, each P_i for $i \in H_2$ reveals

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

3. Each P_i for $i \in H_2$ verifies that

$$\gamma_j G = V + \sum_{k=1}^{t-1} c_k j^k G + h(m||V) \left(Y + \sum_{k=1}^{t-1} b_k j^k G \right) \text{ for all } j \in H_2. \quad (3)$$

Let H_3 be the index set of players not detected to be cheating at step 3.

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

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

The signature is (σ, V). The signature can be verified as in Schnorr's original scheme:

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

Remarks.

1. The scheme can easily be modified such 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.
2. 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 a suitable linear general access structure secret sharing scheme as for example [17].

4.3 Correctness

We have to show that the signature σ computed in Step 4 is the Schnorr signature on m, i.e., $\sigma = e + h(m||V)x \bmod q$. Let $F_1(\cdot)$ be the sharing polynomial of the key generation protocol ($\alpha_i = F_1(i)$ for $i \in H_0$), and let $F_2(\cdot)$ be the sharing polynomial implied by the generated random shared secret in Step 1 ($\beta_i = F_2(i)$ for $i \in H_1$). Furthermore, let $F_3(\cdot) := F_2(\cdot) + h(m||V)F_1(\cdot)$. Since $\gamma_i = F_3(i)$ for $i \in H_3$, it follows from Lagrange's interpolation formula that the players compute $\sigma = F_3(0)$. We can now argue as follows:

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

4.4 Robustness

We have to show that if less than t players are corrupted, the scheme always produces a valid signature. We assume that $t \leq \frac{n}{2}$.

From the robustness property of the protocol to generate a random shared secret it follows that every honest player P_i computes correct values $\alpha_i, \beta_i, \gamma_i$. Because there are at least t honest players, and because they can identify the correct γ_i by verifying (3), it follows directly by the correctness property that the honest players will always compute a valid certificate.

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. She also might corrupt up to $t-1$ arbitrary players. $A_{DistSchnorr}$ then tries to forge a new signature from the signatures she obtained in this way and from her *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 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 her view during this simulation from her view during a real run of $D_{DistSchnorr}$, she will succeed and output a valid forgery, and therefore so will $A_{NormSchnorr}$. Indistinguishability is used in the sense of the traditional notion of polynomial indistinguishability of two probability distributions as specified in [6].

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 during a real run of the protocol which outputs Y. This simulator is then used later as a subroutine of a simulator that computes the adversary's entire view of our threshold signature scheme.

5.2 View

During an arbitrary multi-party protocol, a player will choose 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 honest 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 honest players, the players P_i for $i \in B$ communicate with the simulator. This simulator will now produce messages exactly as the honest players do, such that the public value of the random shared secret is Y, and furthermore, the adversary controlling players P_i for $i \in B$ cannot distinguish this simulated view from the view resulting from the honest 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:

$$\begin{aligned}
\text{the sharing polynomials} \quad & f_i(\cdot), f_i'(\cdot), \\
\text{the temporary shares} \quad & f_j(i), f_j'(i) \text{ for } j \in H_0, \\
\text{the public commitments} \quad & C_{jk}, A_{jk} \text{ for } j \in H_0, k \in \{0, \ldots, t-1\}, \\
\text{answers on a valid complaint against } P_l \quad & f_l(j), f_l'(j) \text{ for } j \in H_0,
\end{aligned}$$

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 $SIM(G, q, Y)$ 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 $i \in B = \{1, \ldots, t-1\}$. Furthermore, let B' be the index set that denotes the players who publish inconsistent values A_{ik}. Then, $view(\tilde{A}, G, q, Y)$, when generating a random shared secret, is as follows:

1. The content of the random tape of $\tilde{A}$
2. $f_i(\cdot), f_i'(\cdot)$ for $i \in B$
3. $f_j(i), f_j'(i)$ for $j \in H_0, i \in B$
4. C_{jk} for $j \in H_0, k \in \{0, \ldots, t-1\}$
5. A_{jk} for $j \in H_0, k \in \{0, \ldots, t-1\}$
6. $f_i(j), f_i'(j)$ for $j \in H_0, i \in B'$

[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 a random variable.

We show how to construct a simulator *SIM* that can act in the protocol as the honest players, such that the resulting view has the same probability distribution (we use the same simulator as in [7]). 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 honest players $P_t, \ldots, P_n$ 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 $i \in H_0$. 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} for $i, j \in H_0$ and $k \in \{0, \ldots, t-1\}$.
2. (5) When extracting the values $r_i G$, the simulator acts as follows:
   - Compute $A_{ik} = a_{ik} G$ for $i \in H_0 \setminus \{n\}, k \in \{0, \ldots, t-1\}$
   - Compute $A_{n0} = Y - \sum_{i \in H_0 \setminus \{n\}} A_{i0}$
   - Compute $A_{nk} = \lambda_{k0} A_{n0} + \sum_{i=1}^{t-1} \lambda_{ki} f_n(i) G$ for $k \in \{1, \ldots, t-1\}$, where λ_{ki}'s are the Lagrange interpolation coefficients of the set H_0.
   - Broadcast A_{ik} for $i \in H_0, k \in \{0, \ldots, t-1\}$
3. (6) To handle the messages resulting from complaints, *SIM* acts as follows:
   - For each honest player, verify the values A_{ik} for $i \in B$ by checking (2). 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 scheme 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 $k \in \{0, \ldots, t-1\}$, 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: She 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 [7]. 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 $(m_1, m_2, \dots)$ 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 t or more 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 $(m_1, m_2, \dots)$ (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*

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

Proof. We show how to construct $A_{DistSchnorr}$ given $A_{NormSchnorr}$. 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*

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

[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 $A_{DistSchnorr}$. In particular, 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.

Let $\mathcal{B}$ be the index set of players corrupted by $A_{DistSchnorr}$. Using the simulator described in Section 5.2, $A_{NormSchnorr}$ lets *SIM* execute the key generation protocol for the given public key Y. Note that $A_{NormSchnorr}$ knows after this simulation the values α_i for $i \in \mathcal{B}$. Next, $A_{NormSchnorr}$ runs $A_{DistSchnorr}$. Whenever $A_{DistSchnorr}$ requests a signature for a message 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 she sees during the signature issuing protocol. These values include the view resulting from generating the random shared secret e and the values γ_i for $i \in H_2 \setminus \mathcal{B}$. To compute these values, $A_{NormSchnorr}$ lets *SIM* interact with $A_{DistSchnorr}$ during the generation of the random shared secret e. After this simulation $A_{NormSchnorr}$ knows β_i for $i \in \mathcal{B}$ and can compute γ_i for $i \in \mathcal{B}$. Finally, $A_{NormSchnorr}$ computes γ_j for $j \in H_2 \setminus \mathcal{B}$ as follows. W.l.o.g. we assume that $\mathcal{B} \cap H_2 = t - 1$. Then we have for every $j \in H_2 \setminus \mathcal{B}$ (see Section 4.2):

$$\sigma_i = \sum_{k \in \mathcal{B} \cup \{j\}} \gamma_k \omega_k, \text{ where } \omega_k = \prod_{\substack{l \neq k \\ l \in \mathcal{B} \cup \{j\}}} \frac{l}{l-k}.$$

Hence, γ_j is computed as

$$\gamma_j = \frac{\sigma_i - \sum_{k \in \mathcal{B}} \gamma_k \omega_k}{\omega_j}.$$

$A_{NormSchnorr}$ feeds γ_j for $j \in H_2 \setminus \mathcal{B}$ to $A_{DistSchnorr}$. Since $A_{DistSchnorr}$ now has her whole view, she can perform her 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. 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, and let $h(\cdot)$ be a one-way hash function. 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: $SK_u = C + h(I_u||C)Y$.

The user's public key can be computed from the certificate (I_u, C) as follows: $PK_u = C + h(I_u||C)Y$. 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 $P_1, \ldots, P_n$ represent a CA with public key PK_0. A group of t players together can reconstruct SK_0 and issue an implicit certificate. Any coalition of less than t players do not have any information about SK_0.

Our scheme consists of three steps. First, the players 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 players shall be able to recover SK_0 or issue certificates. Second, the players issue a certificate to a user. Finally, the user verifies if the certificate is valid.

7.1 Key Generation Protocol

We would like to generate a random shared secret SK_0 such that each player 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:

$$(\alpha_1, \ldots, \alpha_n) \overset{(t,n)}{\longleftrightarrow} (SK_0|PK_0, b_kG, H_0),\ k \in \{1, \ldots, t-1\}.$$

7.2 Certificate Issuing Protocol and Public Key Reconstruction

Suppose the players with index set $H_1 \subseteq H_0$ want to issue an implicit certificate.

1. The user selects a random number c_u and sends $V_u = c_uG$ to the players.

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

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

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

$$\gamma_i = \beta_i + h(I_u||C)\alpha_i. \tag{4}$$

4. Each P_i for $i \in H_2$ verifies that

$$\gamma_j G = V + \sum_{k=1}^{t-1} c_k j^k G + h(I_u||C)\left(Y + \sum_{k=1}^{t-1} b_k j^k G\right) \text{ for all } j \in H_2. \tag{5}$$

Let H_3 be the index set of players not detected to be cheating at step 3.

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

$$\sigma = \sum_{j \in H_4} \gamma_j \omega_j, \text{ where } \omega_j = \prod_{\substack{l \neq j \\ l \in H_4}} \frac{l}{l-j}. \tag{6}$$

The implicit certificate is (σ, C), which every player sends to the user.

6. At most $t-1$ of the certificates the user receives may be incorrect. To identify the correct certificates, the user computes his private key SK_u as $SK_u = c_u + \sigma$ and verifies

$$SK_u G = C + h(I_u||C)Y \text{ and } \sigma \in Z_q. \tag{7}$$

The public key of the user can be computed from the implicit certificate as follows:

$$\sigma PK_u = C + h(I_u||C)Y. \tag{8}$$

7.3 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, i.e., we have to verify that following holds:

$$SK_u G = C + h(I_u||C)PK_0. \tag{9}$$

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

$$\begin{aligned} SK_u G &\overset{(6)}{=} (c_u + \sum_{i \in \mathcal{P}} \gamma_i \omega_i) G \\ &\overset{(4)}{=} c_u G + \left(\sum_{i \in \mathcal{P}} (\beta_i + h(I_u||C)\alpha_i) \right) G\omega_i \\ &= V_u + \sum_{i \in \mathcal{P}} (\beta_i \omega_i G + \alpha_i \omega_i h(I_u||C) G) \\ &= V_u + V + h(I_u||C) PK_0 \\ &= C + h(I_u||C) PK_0. \end{aligned}$$

7.4 Robustness

We have to show that if less than t players are corrupted, the scheme always produces a valid certificate that is accepted by the user. We assume that $t \leq \frac{n}{2}$.

From the robustness property of the protocol to generate a random shared secret it follows that every honest player P_i computes correct values $\alpha_i, \beta_i, \gamma_i$. Because there are at least t honest players, and because they can identify the correct γ_i by verifying (5), it follows directly by the correctness property that the honest players will always compute a valid certificate. Finally, the user can identify a valid certificate by verifying (7).

8 Security Analysis

8.1 Notion of Security

Let (SK_{CA}, PK_{CA}) be the key pair of the CA. An implicit certificate scheme is *secure* if the following two properties hold:

unforgeability It is hard for an adversary who does not know the 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 the CA to obtain the user's private key provided that the user 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.

8.2 Unforgeability

Let (x, Y) be the (SK, PK) key pair of the CA. 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 8.1.

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 players. Since the adversary cannot distinguish a simulated view from an actual view, she will perform her attack and output a forgery. This forgery can then be used to construct the other adversary.

8.3 Non-impersonating

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

8.4 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 Conclusion

We have presented a (t, n) threshold version of Schnorr's signature scheme, and a (t, n) threshold scheme for implicit certificates. Both schemes are efficient, robust and provably secure in the random oracle model.

From a practical point of view, the implicit certificate scheme has the following drawbacks:

- The scheme itself generates a key pair for the user. Therefore, it cannot be used to generate an implicit certificate for a given key pair of the user.
- The scheme produces 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*.

Acknowledgments

This work was done while I was visiting Certicom as an intern during the period August - November, 2000. I would like to thank Certicom for this opportunity. Furthermore, we would like to thank Simon Blake-Wilson for his ideas on the general concepts, Minghua Qu for reviewing the security proofs and for pointing out the special security issues that arise in the context of implicit certificates, and Dan Brown for all the instructive discussions about implicit certificates.

References

[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 *Financial Cryptography '01*, to appear.
[4] Y. G. Desmedt. Threshold cryptography. *European Trans. on Telecommunications*, 5(4):449–457, 1994.
[5] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th FOCS*, pages 427–437, 1987.
[6] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. on Computing*, 18/1:186–308, 1989.

[7] R. Gennaro S. Jarecki H. Krawczyk and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *Eurocrypt '99*, pages 295–310, 1999.
[8] S. K. Langford. Threshold DSS signatures without a trusted party. In *Crypto '95*, pages 397–409, 1995.
[9] E. Okamoto, G. Davida, and M. Mambo. Some recent research aspects of threshold cryptography. In *Workshop on Information Security Applications*, 1997.
[10] C. Park and K. Kurosawa. New elgamal type threshold digital signature scheme. *IEICE Trans.*, E79-A:86–93, 1996.
[11] T.P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Crypto '91*, pages 129–140, 1991.
[12] T.P. Pedersen. A threshold cryptosystem without a trusted party. In *Eurocrypt '91*, pages 522–526, 1991.
[13] L. Pintsov and S. Vanstone. Postal revenue collection in the digital age. In *Financial Cryptography '00*, 2000.
[14] D. Pointcheval and J. Stern. Security proofs for signature schemes. In *Eurocrypt '96*, pages 387–399, 1996.
[15] C.P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4:161–174, 1991.
[16] A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.
[17] M. van Dijk. A linear construction of secret sharing schemes. *Designs, Codes and Cryptography*, 12:161–201, 1997.