Distributed Provers with Applications to Undeniable Signatures

Torben Pryds Pedersen
Aarhus University, Computer Science Department
Ny Munkegade, DK-8000 Århus C, Denmark

Abstract

This paper introduces distributed prover protocols. Such a protocol is a proof system in which a polynomially bounded prover is replaced by many provers each having partial information about the witness owned by the original prover. As an application of this concept, it is shown how the signer of undeniable signatures can distribute part of his secret key to n agents such that any k of these can verify a signature. This facility is useful in most applications of undeniable signatures, and as the proposed protocols are practical, the results in this paper makes undeniable signatures more useful. The first part of the paper describes a method for verifiable secret sharing, which allows non-interactive verification of the shares and is as secure as the Shamir secret sharing scheme in the proposed applications.

1 Introduction

Undeniable signatures were introduced in [CvA90]. Briefly, an undeniable signature is a signature which cannot be verified without the help of the signer (see [CvA90] and [Cha91]). They are therefore less personal than ordinary signatures in the sense that a signature cannot be related to the signer without his help. On the other hand, the signer can only repudiate an alleged signature by proving that it is incorrect.

A manufacturer can use undeniable signatures to sign his products, such that someone who wants to verify the genuineness of a given product has to contact the manufacturer. This way the manufacturer can control the usage of his products.

Convertible undeniable signatures are undeniable signatures with the added property that the signer can convert all the undeniable signatures to ordinary signatures by releasing a part of the secret key, and selectively convertible undeniable signatures allow the signer to convert single signatures to ordinary signatures without affecting other undeniable signatures (see [BCDP90]).

In almost all applications of undeniable signatures that one can imagine, it might be a problem that only the signer can verify the signatures, because this

D.W. Davies (Ed.): Advances in Cryptology - EUROCRYPT '91, LNCS 547, pp. 221-242, 1991.

requires that he can always be reached. Since an undeniable signature does not prove anything in itself, it is very reasonable that a receiver of a signature demands that either the signer or an agent authorized by the signer is always willing to verify signatures. Such an agent can also be a big help to the signer, since a person signing many messages quite rapidly can be overburdened verifying signatures.

With convertible undeniable signatures it is possible for the signer to authorize an agent who can verify all signatures, and if the signatures are selectively convertible, agents can be authorized to verify single signatures. However, this requires that the signer trusts the agents completely.

If the signer does not (want to) trust single persons, he may want to authorize n agents such that verification requires at least k of these to cooperate. This facility was proposed by David Chaum and this paper shows how it can be achieved. The construction falls in two parts. First it is described how the signer can distribute the keys such that each agent can verify that he has received correct shares, and then it is shown how the agents can verify or deny signatures.

In Section 3 we present a scheme for distributing a secret s which uses the fact that information about s is known beforehand through a public key. This scheme is based on the secret sharing scheme by Shamir, which is shortly described in Section 2. After a short description of the undeniable signature scheme from [BCDP90] in Section 4, Section 5 shows how a number of agents can verify these signatures, and Section 6 discusses the possibility of using agents to deny signatures.

2 The Shamir Scheme

In this section the Shamir secret sharing scheme is briefly described and the properties needed in the following sections are summarized (see also [Sha79]).

A (k, n)-threshold secret sharing scheme is a protocol between $n+1$ players in which the ***dealer*** distributes partial information (***share*** or ***shadow***) about a ***secret*** (or ***key***) to n participants such that

- Any group of fewer than k participants cannot obtain any information about the secret.
- Any group of at least k participants can compute the secret in polynomial time.

Consider a field $\mathbb{F}$ and let the secret, s, be an element of $\mathbb{F}$. In order to distribute s among $P_1, \ldots, P_n$ (where $n < |\mathbb{F}|$) the dealer chooses a polynomial $f \in \mathbb{F}[x]$ of degree at most $k-1$ satisfying $f(0) = s$. Participant P_i receives $s_i = f(x_i)$ as his private share, where $x_i \in \mathbb{F} \setminus \{0\}$ is public information about P_i ($x_i \neq x_j$ for $i \neq j$).

Due to the fact that there is one and only one polynomial of degree at most $k-1$ satisfying $f(x_i) = s_i$ for k values of i, the Shamir scheme satisfies the definition of a (k, n)-threshold scheme. Any k persons $(P_{i_1}, \ldots, P_{i_k})$ can find f by

the formula:

$$f(x) = \sum_{l=1}^{k}(\prod_{h\neq l}\frac{x-x_{i_h}}{x_{i_l}-x_{i_h}})f(x_{i_l})$$
$$= \sum_{l=1}^{k}(\prod_{h\neq l}\frac{x-x_{i_h}}{x_{i_l}-x_{i_h}})s_{i_l} \qquad (1)$$

Thus

$$s = \sum_{j=1}^{k} a_j s_{i_j},$$

where $a_1, \ldots, a_k$ are given by

$$a_j = \prod_{h\neq j}\frac{x_{i_h}}{x_{i_h}-x_{i_j}}.$$

Thus each a_i is non-zero and can easily be computed from the public information.

3 Verifiable Secret Sharing

Now the Shamir scheme is applied in a setting where the secret is given as a discrete logarithm. The resulting scheme has the advantage that each participant can verify his share. In [P.F87] Feldman obtained a verifiable secret sharing scheme from the Shamir secret sharing scheme by broadcasting probabilistic encryptions of the polynomial used to generate the shares. Under the assumption that the encryption function is a "homomorphism" it was possible for each shareholder to verify his share without communicating with the other participants.

The scheme suggested by Feldman aims at hiding the secret to be distributed in a very strong sense. However this is not necessary if some information about the secret is known beforehand. We now present a scheme very similar to [P.F87] which is somewhat simpler due to the fact that the public information about the secret is used in the verification of the shares.

3.1 Verification using Discrete Logarithms

Throughout this paper p and q are large primes such that q divides $p-1$, and g generates the subgroup, G_q, of $\mathbb{Z}_p$ of order q. It is assumed that p, q and g are publicly known.

Assume the dealer has a secret $s \in \mathbb{Z}_q$ and is committed to s through a public key $h = g^s$. This secret can be distributed to $P_1, \ldots, P_n$, as follows:

PROTOCOL DISTRIBUTE

1. Compute shares s_i using the Shamir secret sharing scheme in the field $\mathbb{F} = \mathbb{Z}_q$ by first choosing a polynomial $f = f_0 + f_1 x + \ldots + f_{k-1}x^{k-1}$ over $\mathbb{Z}_q$ of degree $k-1$ satisfying $f(0) = s$ and then computing

$$s_i = f(x_i).$$

   Here x_i is public information about P_i as described above.

2. Send s_i ***secretly*** to P_i, and ***broadcast*** $(g^{f_i})_{i=1,\ldots,k-1}$ to all n participants.

Thus the dealer has to broadcast $k-1$ elements in G_q and to send secretly n elements in $\mathbb{Z}_q$.

The public information corresponding to the share s_i is denoted $h_i = g^{s_i}$. Thus h_i depends on s_i in the same way that h depends on s. A participant not knowing s_i can compute h_i as ($g^{f_0} = h$)

$$h_i = \prod_{j=0}^{k-1} (g^{f_j})^{x_i^j}.$$

When all shares have been distributed each participant verifies his share as follows

PROTOCOL VERIFY SHARE (at P_i)

1. Compute $h_l = \prod_{j=0}^{k-1} (g^{f_j})^{x_l^j}$ for all $l = 1, \ldots, n$.
2. Verify, that $h_i = g^{s_i}$.
3. If this is false broadcast s_i and stop.
   Otherwise accept the share.

The signer can always detect, if P_i falsely claims to have received a wrong share, and in that case he should start all over discarding P_i from future protocols.

If the signer follows the protocol, all honest agents will accept their shares, and $h_i = g^{s_i}$ will be publicly known for $i = 1, \ldots, n$. The next proposition shows, that no matter how the dealer computes the shares, any k participants who have accepted their shares can find s.

Proposition 3.1
Any k participants, who have followed PROTOCOL VERIFY SHARE and accepted, can find s.

Proof
Assume that the k participants are $P_1, \ldots, P_k$. It is sufficient to show that the unique polynomial f' of degree at most $k-1$ satisfying

$$f'(x_i) = s_i \qquad \text{for } i = 1, \ldots, k$$

also satisfies $f'(0) = s$. But

$$\prod_{j=0}^{k-1} (g^{f_j})^{x_i^j} = h_i = g^{s_i} = g^{f'(x_i)}$$

implies

$$\sum_{j=0}^{k-1} f_j x_i^j = f'(x_i) \bmod q$$

and the uniqueness of f' implies that $f'(0) = f(0) = s$. ■

Theorem 3.2 below shows that any number of participants can simulate the dealer perfectly no matter what shares they get. Thus fewer than k participants do not get any information about s which allows them to compute something that they could not have computed before the secret was distributed.

Theorem 3.2
Any l participants having shares $(s_{i_j})_{j=1,\ldots,l}$ can find $(g^{f'_j})_{j=0,\ldots,k-1}$, such that

$$f'(x) = f'_0 + f'_1 x + \ldots + f'_{k-1} x^{k-1}$$

is a random polynomial of degree at most $k-1$ satisfying

$$\begin{aligned} f'(0) &= s \\ f'(x_{i_j}) &= s_{i_j}, \qquad j = 1, \ldots, l. \end{aligned}$$

Proof
If $l \geq k$ the proposition is trivial as any k agents can find the polynomial used by the signer.

Now assume that $1 \leq l < k$ and let the l agents in question be $P_1, \ldots, P_l$. The l participants generate f'as follows:

1. Choose $k-1-l$ random "shares" $s_{l+1}, \ldots, s_{k-1}$ corresponding to the public information $x_{l+1}, \ldots, x_{k-1}$.
2. Find $g^{f'_i}$ for $i = 0, \ldots, k-1$, where the polynomial $f'(x) = f'_0 + \ldots + f'_{k-1} x^{k-1}$ satisfies $f'(x_i) = s_i$ for $i = 1, \ldots, k-1$ and $f'(0) = s$ (see below).

As $s_{l+1}, \ldots, s_{k-1}$ were chosen at random the (unknown) polynomial f' generated this way is completely random such that

$$\begin{aligned} f'(0) &= s \\ f'(x_j) &= s_j, \qquad j = 1, \ldots, l. \end{aligned}$$

It only remains to show how $(g^{f_i'})_i$ are found. The polynomial f' is going to satisfy $(s_0 = s,\ x_0 = 0)$:

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{k-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{k-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{k-1} & x_{k-1}^2 & \cdots & x_{k-1}^{k-1} \end{pmatrix} \begin{pmatrix} f_0' \\ f_1' \\ \vdots \\ f_{k-1}' \end{pmatrix} = \begin{pmatrix} s_0 \\ s_1 \\ \vdots \\ s_{k-1} \end{pmatrix}$$

This $k \times k$ matrix is a Van der Monde matrix, and it has an inverse, A, as $x_i \neq 0$ and $x_i \neq x_j$ for $i \neq j$. Thus

$$A \begin{pmatrix} s_0 \\ s_1 \\ \vdots \\ s_{k-1} \end{pmatrix} = \begin{pmatrix} f_0' \\ f_1' \\ \vdots \\ f_{k-1}' \end{pmatrix}.$$

Let

$$A = \begin{pmatrix} a_{00} & a_{01} & a_{02} & \cdots & a_{0,k-1} \\ a_{10} & a_{11} & a_{12} & \cdots & a_{1,k-1} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{k-1,0} & a_{k-1,1} & a_{k-1,2} & \cdots & a_{k-1,k-1} \end{pmatrix}$$

and note that $P_1, \ldots, P_i$ can find each a_{ij}. Thus $g^{f_i'}$ can be computed for $i = 0, \ldots, k-1$ by the formula ($g^{s_0} = \mathrm{h}$ is known)

$$\prod_{j=0}^{k-1} g^{s_j a_{ij}} = g^{f_i'}.$$

This proves the theorem. ∎

4 Convertible Undeniable Signatures

This section contains a short description of the selectively convertible undeniable signature scheme from [BCDP90].

Let p, q and g be as above. For all a and b in G_q with $b \neq 1$ the discrete logarithm of a to the base b is defined and denoted $\log_b(a)$. The simultaneous discrete logarithm problem is to decide given four elements a, b, c and d if $\log_b(a)$ equals $\log_d(c)$.

The private keys in the scheme are

$$KS1 = x \text{ and } KS2 = z, \quad 1 < x, z < q$$

and the public key is

$$KP = (p, q, g, y, u), \text{ where } y = g^x \text{ and } u = g^z.$$

Any receiver of the public key can easily verify that y and u generate G_q.

The signature on the message m is $sign(m) = (g^t, r, s)$, where (r, s) is the El Gamal signature on $M = g^t tzm \bmod q$ (in this product g^t is considered an element in $\mathbb{Z}_q$). That is

$$g^M = y^r r^s \bmod p.$$

As noted in [BCDP90] m should be hashed before signing, but the hash function is omitted here.

Given m and (T, r, s) both the signer and the verifier can compute $w = T^{Tm}$ and $v = y^r r^s$. The signer can prove that (T, r, s) is (not) a signature on m by proving that $\log_w(v) = \log_g(u)$ ($\log_w(v) \neq \log_g(u)$). The protocol in figure 1 is a variant of the proof system for simultaneous discrete logarithm in [Cha91] and shows how the signer can verify a signature.

P		V
		Choose $a, b \in \mathbb{Z}$ and compute $ch = w^a g^b$
	$\xleftarrow{ch}$	
Choose $r \in \mathbb{Z}$ and compute $h_1 = ch^r$ and $h_2 = h_1^z$		
	$\xrightarrow{(h_1, h_2)}$	
	$\xleftarrow{(a, b)}$	
Verify that $ch = w^a g^b$		
	$\xrightarrow{r}$	
		Verify, that $h_1 = (w^a g^b)^r$ and $h_2 = (v^a u^b)^r$

Figure 1: Proof that $\log_w(v) = \log_g(u)(= z)$.

The signer can convert all his signatures to ordinary signatures by releasing z. Alternatively, a signature (T, r, s) on the message m can be converted to a digital signature by releasing t such that $T = g^t$. Given t, a signature can be verified as follows:

1. Verify, that $T = g^t$.
2. Verify, that $(u^{mT})^t = y^r r^s$.

Anyone, who can solve the simultaneous discrete logarithm problem can obviously recognize valid signatures. In [BCDP90] it is argued that it is hard to verify a signature without the aid of the signer even if the signer verifies other signatures.

5 Distributed Prover Protocols

If the signer in an undeniable signature scheme is going away for a while, the receivers of signatures might request that a trusted third person gets the secret key of the scheme such that this person can verify signatures during the signer's absence. If the signer doesn't want to give away the secret, an obvious solution is to distribute it to n persons (agents) with a (k, n)-secret sharing scheme. When a verifier wants a signature to be verified or denied, he can ask k of these n persons for assistance.

Such an $(k+1)$-party protocol between the verifier and k agents will in general be called a ***distributed prover protocol*** (a distributed prover protocol differs from a multi-prover proof system (see [BOGKW88]) as the provers may have unlimited computing power in the latter case).

These k persons can help the verifier by first finding the secret and subsequently one of them can execute the usual verification/denial protocol. However, this would be against the intentions of the signer, because in that case the secret could just as well have been given to one person in the first place.

In this section we construct protocols that allow k persons to verify signatures ***without*** finding the secret. From a theoretical point of view it is possible to solve this problem, if the agents are allowed to talk with each other (for instance by using the techniques in [CDvdG88]). However, the distributed verification protocol presented in Section 5.2 is very efficient, and it does not require interaction between the agents. Thus secret communication is only needed in the setup phase when the secret key is distributed.

First the model is presented and a definition of security of a distributed prover protocol is given.

5.1 The Model

It is assumed that the n agents are selected such that at most $k-1$ of them will ever deviate from the prescribed protocols and try to find the secret key.

Each agent is modeled by a probabilistic polynomial time Turing machine having in addition to a computation tape and a random tape

- a tape for broadcasting messages (can be read by all other participants);
- a tape for common input (shared by all users);
- a tape for auxiliary input (can only be read by the owner);
- a tape for receiving secret messages from the signer (read only).

A cryptographic protocol is secure, if any polynomial (in a security parameter) number of executions of it does not enable one of the parties to do a computation afterwards, that he could not have done beforehand.

As a polynomial number of polynomial time Turing machines can be simulated by a single polynomial time Turing machine, it can be assumed that the verifier is the same in all executions of the protocol (although the verifier may behave differently in each execution). This automatically handles the situation where different verifiers cooperate.

Definition 5.1
An agent is ***honest*** if it follows the protocol in all executions. An agent who is not honest is called ***dishonest***.

For any verifier and any set of dishonest agents consider the following protocol:

1. Distribute the secret.
2. Repeat a polynomial number of times: The verifier and the dishonest agents select an input to the distributed protocol and a set of agents with whom the protocol is executed.

Following the ideas of [CDvdG88], where the security of general multi-party protocols is defined, we say

Definition 5.2
Let K_P be the public key of the original prover.
A distributed prover protocol is secure, if for every set of dishonest agents and every verifier, V^*, for every set of auxiliary inputs to the verifiers and the dishonest agents, there exists a probabilistic polynomial time machine, M, such that the output of M on input K_p is polynomially indistinguishable from the transcript of an execution of the above protocol.

This definition allows that an execution of the distributed protocol reveals some information about the shares of the honest agents as long as this information cannot be used to obtain new information about the original prover's secret key or the signatures.

5.2 Distributed Verification

Consider the case where the signer, S, has signed the message, m, using the random exponent, t. Thus $sign(m) = (T, r, s)$ where $T = g^t$ and (r, s) is the El Gamal signature on the product $Ttzm$ modulo q. S distributes the ability to verify this signature to n agents ($P_1, \ldots, P_n$ with corresponding public keys $x_1, \ldots, x_n$) by distributing t. As $T = g^t$ is part of the signature (and therefore not secret), the secret sharing scheme from Section 3 can be used:

PROTOCOL DISTRIBUTE SINGLE SIGNATURE

1. S broadcasts T to the n agents.
2. S distributes t using PROTOCOL DISTRIBUTE in section 3. Thus P_i gets the share $t_i = f(x_i)$, where f is a polynomial over $\mathbb{Z}_q$ of degree $k-1$ such that $f(0) = t$.
3. Each agent P_i executes PROTOCOL VERIFY SHARE from section 3.
4. S sends $H(m, r, s)$ to each agent, where H is a collision-free hash function.

After the execution of this protocol each P_i has a secret share t_i with corresponding public information $h_i = g^{t_i}$. In addition to these values each agent has a hash value of the signature and the signed message, which is used to decide if a signature should be verified.

When a person, V, asks k agents (say $P_1, \ldots, P_k$) to verify a signature (T', r', s') on a message m', the agents first have to make sure that they *are able to* verify it and then decide if the signature is correct. Let $a_1, \ldots, a_k$ be elements in $\mathbb{Z}_q$ such that

$$\sum_{i=1}^{k} t_i a_i = t.$$

As mentioned in Section 3, each a_i can be computed from $(x_i)_{i=1,\ldots,k}$.

PROTOCOL DECIDE

1. V and each P_i verify that $T' = \prod_1^k h_i^{a_i}$.
   If this is not the case, the agents can neither verify nor deny the signature.
2. Each P_i verifies that the signer has sent $H(m', r', s')$. If this is true, the agents agree to verify the signature and otherwise they tell V that they are not able to verify it.

The result of PROTOCOL DECIDE is *not* a proof that the signature is correct/false, because the decision is based on values that anyone could have produced.

$P_1, \ldots, P_k$ can verify a signature by executing (now $T' = T$)

PROTOCOL DISTRIBUTED VERIFICATION

1. P_i and V compute $w = u^{Tm'}$ and $v = y^{r'} r'^{s'}$.
2. $P_1, \ldots, P_k$ proves that $\log_w(v) = \log_g(T)$ as shown in figure 2.
3. V accepts the signature if and only if it accepts the proof.

An honest agent reveals w^{t_i} in an execution of the protocol, as it can be computed from $ch^{r_i t_i}$ when r_i is known. The following theorem shows, that an honest agent does not reveal more than this, and it is shown that the agents cannot verify an invalid signature.

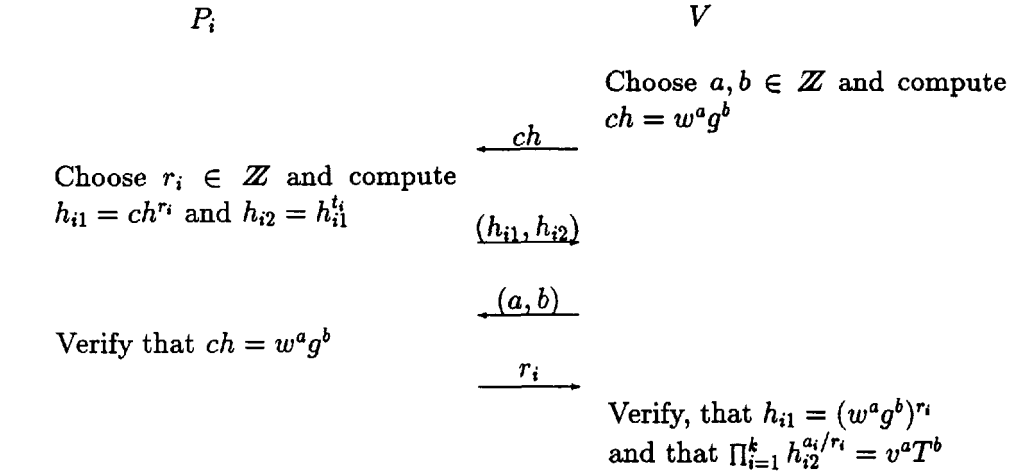

Figure 2: Distributed proof that $\log_w(v) = \log_g(T) (= \sum a_i t_i)$.

Theorem 5.3
The protocol in figure 2 satisfies

1. If $v = w^t$, the verifier accepts with probability 1.
2. If $v \neq w^t$, the verifier accepts with probability at most $\frac{1}{q}$ – even if the agents have unlimited computing power.
3. For any probabilistic polynomial time verifier V^* and for any set $D \subseteq \{1, \ldots, k\}$ of dishonest agents there is a machine $M_{V^*,D}$ running in expected polynomial time such that $M_{V^*,D}$ given
   - the common input (p, q, g, T, w, v), where $\log_g(T) = \log_w(v)$;
   - the auxiliary input of V^* (aux_{V^*});
   - the auxiliary input of the dishonest agents $((aux_i)_{i \in D})$; and
   - $(w^{t_i})_{i \in H}$ where $H = \{1, \ldots, k\} \setminus D$

   outputs a conversation having the same distribution as in real executions of the protocol.

Proof
If $v = w^t$ and the provers follow the protocol then

$$\begin{aligned} \prod_{i=1}^{k} h_{i2}^{a_i/r_i} &= \prod_{i=1}^{k} (ch^{r_i t_i})^{a_i/r_i} \\ &= \prod_{i=1}^{k} ch^{t_i a_i} \end{aligned}$$

$$= ch^t$$
$$= v^a T^b.$$

and therefore the verifier accepts.

If, on the other, hand there exists k agents, who can convince the verifier about a false claim with probability greater than q^{-1}, then there is a strategy for the prover in the corresponding single prover protocol (see figure 1), which makes the verifier accept with probability greater than q^{-1}. This is a contradiction (see [BCDP90]).

The third property can be proven by a standard simulation. Let V^* and D be given and let $H = \{1, \ldots, k\} \setminus D$ be the set of honest agents. $M_{V^*,D}$ works as follows

1. V^* produces a challenge ch.
2. For the honest provers compute $h_{i1} = g^{e_i}$ and $h_{i2} = h_i^{e_i}$ where $e_i \in \mathbb{Z}_q$ is a random element.
   For the dishonest provers (h_{i1}, h_{i2}) is computed as in the protocol.
3. Get (a, b) from the verifier.
   If $ch \neq w^a g^b$ stop.
4. Rewind V^* and the provers in D to after ch was sent.
5. For $i \in H$ compute $h_{i1} = ch^{r_i}$ and $h_{i2} = ((w^{t_i})^a h_i^b)^{r_i}$.
   The dishonest provers compute (h_{i1}, h_{i2}) as usual.
6. Get (a', b') from the verifier.
   If $ch \neq w^{a'} g^{b'}$: goto (4).
   If $(a', b') \neq (a, b)$: find $\log_g(w) = \frac{b'-b}{a-a'} \bmod q$, simulate the honest provers perfectly and then stop.
   If $(a', b') = (a, b)$: give r_i to the verifier for $i \in H$, and for $i \in D$ compute r_i as usual. Then stop.

This machine runs in expected polynomial time, and its output has the same distribution as the conversations of a real execution, because the pairs (h_{i1}, h_{i2}) always have the same distribution as in real executions. ∎

As the verifier may execute the protocol several times on input, (w, v), the verifier may learn (w^{t_i}) for all $i \in H$, where $H = \{1, \ldots, n\} \setminus D$ denotes the set of honest agents. The following theorem shows that as long as the honest agents get randomly chosen shares, then no matter what secret shares the dishonest agents get, this information is completely useless with respect to any polynomial time computation of the verifier and the dishonest agents.

This is proven by giving for any verifier, V^* and any set of dishonest agents, D, a polynomial time machine, which on input $g, h, (t_i)_{i \in D}$ simulates the following perfectly (see Section 5.1):

1. The signer chooses random shares for $i \in H$ with the property that any k agents in $H \cup D$ can find t and publishes $h_i \in G_q$ for $i \in H$.
2. A polynomial (in $|q|$) number of times, V^* and the agents in D, choose a set of k agents among $H \cup D$. Then these k agents execute the protocol with V^* on input (w, v).

Note, that we have assumed that the same input will be used in all executions. The distributed protocol is secure, even if the agents get different inputs in each execution, but in the proposed application, the input will always be the same unless a collision of the hash-function has been found.

Theorem 5.4
For any probabilistic polynomial time verifier, V^*, and all sets of dishonest agents, D, all sets of shares, $(t_i)_{i \in D}$, there exists a probabilistic expected polynomial time machine, $M_{V^*,D}$, which on input $(p, q, g, h, (t_i)_{i \in D})$ and the auxiliary inputs of V^* and the dishonest agents simulates the above scheme perfectly.

Proof sketch
If $|D| \geq k$, the theorem is obviously true.

For $|D| < k$ the simulator is based on the same principles as the proof of Theorem 3.2 and works as follows:

1. Choose a subset $H' \subseteq H$, such that $|H' \cup D| = k - 1$.
   Choose random shares $t_i \in \mathbb{Z}_q$ for $i \in H'$ and let $h_i = g^{t_i}$ and $v_i = w^{t_i}$.
2. As in the proof of Theorem 3.2 compute g^{f_j} and w^{f_j} for $j = 0, \ldots, k-1$, where the polynomial

   $$f(x) = f_0 + \ldots + f_{k-1}x^{k-1}$$

   satisfies

   $$\begin{aligned} f(0) &= t \\ f(x_i) &= t_i \qquad \text{for } i \in H' \cup D \end{aligned}$$

3. Compute for $i \in H \setminus H'$ the public information

   $$h_i = g^{f(x_i)} = \prod_{j=0}^{k-1} (g^{f_j})^{x_i^j}$$

   and

   $$v_i = w^{f(x_i)} = \prod_{j=0}^{k-1} (w^{f_j})^{x_i^j}$$

   Let $t_i = f(x_i)$ (t_i is not known for $i \in H \setminus H'$).

4. Run V^* and the dishonest agents to choose a set of agents to execute the protocol.
5. Run the simulator from the proof of Theorem 5.3 giving it v_i for $i \in H$ as input.
6. Run V^* and the dishonest agents in order to decide whether to go to 4) or to stop.

By assumption the protocol is only executed a polynomial number of times and thus $M_{V^*,D}$ stops in expected polynomial time.

Furthermore, the output of $M_{V^*,D}$ has the same distribution as a a real execution of the protocol described above Theorem 5.4, because the h_i's and v_i's have the correct distribution and the simulation in the proof of Theorem 5.3 is perfect. ■

It has silently been assumed that all executions of the protocol are done sequentially. Due to the fact that the protocol in figure 1 is also perfect zero-knowledge when executed many times with different verifiers simultaneously, the distributed protocol is also secure, when executed in parallel (with different sets of agents).

In particular this implies that the verifier cannot use a transcript of an execution of PROTOCOL DISTRIBUTED VERIFICATION as a proof of the validity of a signature.

5.3 Generalizations

The simulator in the proof of Theorem 5.4 can be modified to handle different inputs to each execution of the protocol by computing the v_i's in step 5. By distributing z, the signer can therefore authorize agents, that are able to verify all the signatures.

This facility, however, requires that the agents are able to decide whether a given triple (T, r, s) is a signature on m or not. They can do this by performing a multi-party computation, whose output says if the signature is valid or not. In this case, care must be taken to prevent that a dishonest agent convinces the verifier of the result of this computation.

Alternatively, the signer could give the agents a list of hash values of signatures and then the agents make their decision based on this list. This requires, that the signer updates this list every time a new message is signed.

6 Distributed Denial

This section investigates the possibility of using the agents to deny signatures. It can be argued that this facility is not necessary, since denial of signatures is not expected to take place as often as verification. Furthermore, it is likely, that

denial will take place in court, and in that case it is more reasonable that the signer or a single agent authorized by the signer is present.

In spite of this, the following section suggests how a number of agents can prove that an alleged signature is false. Suppose (T, r, s) is a legal signature on m which has been distributed to the n agents as described above. Thus $t = \log_g T$ has been distributed to n agents and at some point k of these are asked to prove that a given triple (T', r', s') is not a signature on m', where T equals T'.

As the signer only uses T in one signature the agents know that the signature is invalid if $T' = T$ and the signer did not send $H(m', r', s')$ when the correct signature was distributed (see Section 5.2).

Using the techniques described below, agents sharing z can deny any (false) signature, but as discussed in Section 5.3 this requires that they are able to decide whether an arbitrary signature is correct or not.

6.1 Denial by the Agents

Assume that the signer has distributed T to $P_1, \ldots, P_n$ as described in Section 5.2. $P_1, \ldots, P_k$ can prove that (T, r, s) is not a signature on m as follows:

PROTOCOL DISTRIBUTED DENIAL

1. P_i and V compute $w = u^{Tm}$ and $v = y^r r^s$.
2. $P_1, \ldots, P_k$ proves that $\log_w(v) \neq \log_g(T)$ as shown in figure 3.
3. V accepts that the signature is false if and only if he accepts the proof.

Figure 3 shows how the agents can prove inequality of discrete logarithms. All participants in this protocol get p, q, g, T, $(h_1, \ldots, h_k)$ and (v, w) as common input, and the i'th prover gets $t_i = \log_g h_i$ as auxiliary input. The keys to the commitment scheme should be supplied by a trusted key authentication center.

Theorem 6.1
The protocol in figure 3 satisfies

1. If $\log_w(v) \neq \log_g(T)$ then V accepts with probability $1 - \frac{1}{q}$, if the agents follow the protocol.
2. If $\log_w(v) = \log_g(T)$ then no matter what k agents with unlimited computing power does, V accepts with probability at most $\frac{1}{q}$.

Proof
For $e \neq 0 \bmod q$

$$\begin{aligned} w_2 = v_1 &\Leftrightarrow v^e = (w^e)^x \\ &\Leftrightarrow v = w^t. \end{aligned}$$

P_i		V
Choose $e_i \in \mathbb{Z}_q$ and $r_i \in \{0,1\}^*$. Compute $w_{1i} = w^{e_i}$ and $\beta_i = E(w_{1i}, r_i)$.	$\xrightarrow{\beta_i}$	
When everybody has sent β_i, the commitments are opened.	$\xrightarrow{r_i}$	
Find all w_{1i} and compute $w_1 = \prod_1^k w_{1i}$ If $w_1 = 1$ stop. Compute $v_{1i} = v^{e_i}$		Find all w_{1i} and compute $w_1 = \prod_1^k w_{1i}$. If $w_1 = 1$ stop and reject the proof.
	$\xrightarrow{v_{1i}}$	
Prove to V that $\log_v(v_{1i}) = \log_w(w_{1i})$ as in figure 1		Execute the proof in figure 1 with P_i and verify that $\log_v(v_{1i}) = \log_w(w_{1i})$
Compute $v_1 = \prod_1^k v_{1i}$ Compute, $w_{2i} = w_1^{t_i}$		Compute $v_1 = \prod_1^k v_{1i}$
	$\xrightarrow{w_{2i}}$	
Prove that $\log_{w_1}(w_{2i}) = \log_g(h_i)$ as in figure 1 Compute $w_2 = \prod_1^k w_{1i}^{a_i}$		Compute $w_2 = \prod_1^k w_{1i}^{a_i}$ If one of the proofs is not accepted, stop. Otherwise: accept, if $w_2 \neq v_1$ and reject, if $w_2 = v_1$.

Figure 3: Distributed proof that $\log_w(v) \neq \log_g(T)$, where $T = \prod h_i^{a_i}$ and $h_i = g^{t_i}$. $E(\alpha, r)$ denotes a commitment to α using the random string $r \in \{0,1\}^*$.

Therefore the first claim follows from the fact that V accepts the proof if $w_1 \neq 1$, which happens with probability $1 - \frac{1}{q}$.

The second claim follows from the fact that a cheater in the proof system in figure 1 will succeed with probability at most $\frac{1}{q}$. ∎

The proof system in figure 3 is probably ***not*** auxiliary input zero-knowledge, because if the verifier does not cooperate with any provers, each P_i reveals

$$(v_1, w_1, w_{2i}) = (v^e, w^e, w_1^{t_i}),$$

which presumably cannot be constructed by a polynomial time machine which gets v and w as input (here $e = \sum_1^k e_i$, is unknown).

Despite this the following shows that it is very hard for the verifier to obtain any advantage by executing the protocol.

First notice, that the verifier cannot choose v and w freely, but they must be chosen on the form

$$v = y^r r^s \qquad \text{and} \qquad w = u^{TH(m)}$$

where H is a hash-function. Under the assumption that the image of m under H looks like a random string of bits, w looks like a random element of G_q. Furthermore, if the ElGamal scheme is secure, then v is the image of r and s under a one-way function, and thus it is hard for the verifier to control v and w. In Appendix A, it is argued that if the verifier cannot choose v and w better than at random, then the protocol is secure.

As it also seems to be very hard to exploit the knowledge of r, s, m and T, the only possibility for a cheating verifier is to execute the protocol several times with the same v and/or w (perhaps with different sets of agents). But due to the fact that w_1 is chosen (almost) at random in each execution, it seems hard to obtain any information by comparing different executions of the protocol.

7 Conclusion

This paper has introduced distributed prover protocols. This notion has many potential applications, and here we have shown how the signer of undeniable signatures can use it to authorize agents, such that any k of these agents can verify an undeniable signature without being able to sign new messages. Furthermore, it is shown that the agents can be authorized to verify either a single signature or all signatures produced by the signer.

In order to distribute the secret key to the agents, a variation of the verifiable secret sharing scheme proposed by Feldman has been presented. The resulting scheme is designed for situations where the secret key is "known" as the discrete logarithm and is optimal with respect to the secrecy of the key. We have mainly focused on the application of this scheme in distributed prover protocols, but it

can be used in any multi-party computation in which the secret input of each player (x_i) can be recognized through the public information (g^{x_i}, g).

Finally it has been shown how the agents can deny false signatures.

Acknowledgements

I wish to thank David Chaum for suggesting this problem and Ivan Damgård for many discussions about the proposed methods.

References

[BCDP90] J. Boyar, D. Chaum, I. Damgård, and T. Pedersen. Convertible undeniable signatures, 1990. To appear in the proceedings of Crypto'90.

[BOGKW88] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson. Multi-prover interactive proofs: How to remove intractability. In *Proceedings of the 20th Annual ACM Symposium on the Theory of Computing*, pages 113 – 131, 1988.

[CDvdG88] D. Chaum, I. Damgård, and J. van de Graaf. Multiparty computations ensuring privacy of each party's input and correctness of the result. In *Advances in Cryptology - proceedings of CRYPTO 87*, Lecture Notes in Computer Science, pages 87–119. Springer-Verlag, 1988.

[Cha91] D. Chaum. Zero-knowledge undeniable signatures. In *Advances in Cryptology - proceedings of EUROCRYPT 90*, Lecture Notes in Computer Science, pages 458 – 464. Springer Verlag, 1991.

[CvA90] D. Chaum and H. van Antwerpen. Undeniable signatures. In *Advances in Cryptology - proceedings of CRYPTO 89*, Lecture Notes in Computer Science. Springer Verlag, 1990.

[P.F87] P.Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proceedings of the 28th IEEE Symposium on the Foundations of Computer Science*, pages 427 – 437, 1987.

[Sha79] A. Shamir. How to share a secret. *CACM*, 22:612–613, 1979.

A The Distributed Proof of Different Discrete Logarithms

This appendix contains an analysis of the protocol in figure 3. An honest prover in a single execution of the protocol will be simulated by generating

$$\begin{aligned} v_{1i} &= v^{e_i} \\ w_{1i} &= w^{e_i} \\ w_{2i} &: \text{ a random element of } G_q \end{aligned}$$

where e is chosen at random in $\mathbb{Z}_q$. This simulation does not work against a verifier who (for example) knows a and b such that

$$v = h_i^b \qquad \text{and} \qquad w = g^a$$

because then w_{2i} must satisfy (remember $h_i = g^{t_i}$)

$$w_{2i}^{b/a} = w_1^{t_i b/a} = v_1$$

but this is very unlikely for a randomly chosen w_{2i}. However, as noted in Section 6.1 it is very hard for the verifier to choose such a pair (v, w) in PROTOCOL DISTRIBUTED DENIAL.

In the following it is therefore assumed that v and w are chosen uniformly at random in G_q. Consider an execution of the protocol between $P_1, \ldots, P_k$ and a verifier V^*, and assume that $D \subset \{P_1, \ldots, P_k\}$ is the set of dishonest provers. Let ci denote the common input to the protocol (omitting p, q and g). Then ci is on the form

$$(T, h_1, \ldots, h_k, v, w).$$

where

$$T = \prod_{i=1}^{k} h_i^{a_i} \qquad \text{and} \qquad \log_w v \neq \log_g T.$$

A polynomially bounded machine, $M_{V^*,D}$, with access to the dishonest provers and V^* simulates an honest prover as following on input ci:

1. Choose e_i at random and compute β_i.
2. Open β_i and when all k values of w_{1j} are known, compute w_1 as their product.
3. Compute $v_{1i} = v^{e_i}$ and when all k values of v_{1i} are known, prove that $\log_v(v_{1i})$ is equal to $\log_w(w_{1i})$. Finally compute $v_1 = \prod_{j=1}^{k} v_{1j}$.
4. Choose w_{2i} at random.
5. Simulate the "proof" that $\log_{w_1}(w_{2i}) = \log_g(h_i)$.

In step 5) the simulator is going to simulate a "proof" of a false claim as it is very unlikely that w_{2i} equals $w_1^{t_i}$. Lemma A.1 below shows that it is possible to simulate the proof system for equality of discrete logarithms in such a way that the simulator always stops in expected polynomial time (this is not the case for the "obvious" simulator). Furthermore, if the input to the simulator is correct, then the output of the simulator is statistically indistinguishable from executions of the protocol in figure 1.

$M_{V^*,D}$ generates (v_{1i}, w_{1i}) with the same distribution as executions of the protocol. The only difference between a simulated conversation and a transcript of a real execution of the protocol is that the simulator chooses w_{2i} at random. If this random w_{2i} cannot be distinguished from $w_1^{t_i}$ then the the simulation of the "proof" that $\log_{w_1} w_{2i} = \log_g h_i$ cannot be distinguished from an execution of a correct proof.

However, in order for the signature scheme to be undeniable it must be infeasible to decide if $\log_g T$ equals $\log_w v$ for randomly chosen v and w. As v, w and w_1 are chosen at random (almost) it is reasonable to assume that w_{2i} cannot be distinguished from $w_1^{t_i}$, and thus the simulations can not be distinguished from real executions.

Lemma A.1
Consider the proof-system in figure 1.
For any probabilistic polynomial time verifier V^* there is a machine M_{V^*} running in expected polynomial time on all inputs (p, q, g, u, v, w). If $\log_g(u) = \log_w(v)$, the output of M_{V^*} is statistically indistinguishable from a transcript of an execution of the protocol.

Proof
Let V^* be a polynomial time verifier. M_{V^*} does the following: (remember that $u = g^x$)

1. V^* produces a challenge ch.
2. Compute $h_1 = g^e$ and $h_2 = u^e$ where $e \in \mathbb{Z}_q$ is a random element.
3. Get (a, b) from the verifier.
   If $ch \neq w^a g^b$ stop.
4. Alternately execute one round of procedure A and B below until one of them stops:
   Procedure A:
   (a) Rewind V^* to after ch was sent.
   (b) Compute $h_1 = ch^r$ and $h_2 = (v^a u^b)^r$.
   (c) Get (a', b') from the verifier.
   If $ch \neq w^{a'} g^{b'}$: goto (a).
   If $(a', b') \neq (a, b)$: find $\log_g(w) = \frac{b-b'}{a'-a}$, simulate the protocol perfectly

and then stop.
If $(a', b') = (a, b)$: give r to the verifier and stop.

Procedure B:

(a) $count := 0$.
(b) Rewind V^* to after ch was sent.
(c) Compute $h_1 = g^e$ and $h_2 = u^e$, where $e \in \mathbb{Z}_q$ is chosen at random.
(d) Get (a', b') from the verifier.
(e) If $ch = w^{a'} g^{b'}$: count := count + 1.
(f) If $count < |q|$: goto (b);
Otherwise: stop.

Let $P(ch)$ be the probability that V^* sends (a, b) such that $ch = w^a g^b$ when given random pairs (h_1, h_2) satisfying $h_1^z = h_2$.

To show that M_{V^*} runs in expected polynomial time it is sufficient to show that the expected number of iterations of procedure B is polynomial. As each round of B is run independently of previous rounds, this number is

$$P(ch)\frac{|q|}{P(ch)} = |q|$$

Next it will be shown that the output of the simulator is statistically indistinguishable from executions of the protocol whenever the input satisfies $\log_g(u) = \log_w(v)$. If M_{V^*} stops in step 3, the generated output has the correct distribution.

If M_{V^*} stops, because procedure A stops before procedure B, there are two possibilities:

- The simulator has found $(a, b) \neq (a', b')$ such that $ch = w^a g^b = w^{a'} g^{b'}$. In this case the simulation is perfect.
- The simulator has generated the messages

$$h_1 = ch^r,\ h_2 = (v^a u^b)^r \text{ and } r \text{ randomly chosen.}$$

These messages have the same distribution as the messages in real executions.

Finally there is the possibility, that procedure B stops before A, in which case the output of the simulator differs from executions of the protocol. It will now be shown that this happens with negligible probability.

We say that A has success in a round, if it stops. Similarly B has success in one round if the verifier sends (a', b') such that $ch = w^{a'} g^{b'}$. A wins as soon as it has one success, and B wins if it has $|q|$ successes before A has had any. Let the outcome of one execution of a round of A and B be ($P = P(ch)$)

- α, if A has success. $Prob[\alpha] = P$.
- β, if B has success and A has not. $Prob[\beta] = P(1 - P)$.
- *discard*, if neither A nor B has success.

By performing many (independent) experiments with outcomes and probabilities as above and by removing all occurrences of *discard* we get a list of α and β. The probability that β occurs at a given place in the list is ($P > 0$ as the protocol did not stop in step 3)

$$P_\beta = \frac{P(1-P)}{P+P(1-P)} < \frac{1}{2}.$$

B only wins if the first $|q|$ elements in the list are β:

$$Prob[B \text{ wins}] = P_\beta^{|q|} < 2^{-|q|}$$

Thus a simulated conversation has the same distribution as in a real execution of the proof system except with probability less than $2^{-|q|}$. ■