Distributed Provers and Verfiable Secret Sharing Based on the Discrete Logarithm Problem

Torben Pryds Petersen

March 1992

Abstract (in Danish)

I et asymmetrisk ("public-key") krypto-system bar en person (A) en hemmelig nøgle, hvis tilhørende offentlige nøgle er tilgængelig for enhver. Siden sådanne systemer blev foreslået i 1976 (se [DH76]), har studiet af metoder, hvormed A kan overbevise andre om, at han virkelig kender denne hemmelige nøgle, spillet en stor rolle indenfor kryptografi.

I adskillige år kunne de fleste eksisterende sådanne autenticitets-systemer placeres i en af nedennævnte to klasser:

- Identifikations-systemer: modtageren bliver overbevist om A's identitet, men kan ikke nødvendigvis bevise overfor andre, at han har talt med A.

- Digitale signaturer: modtageren får en signatur fra A. Denne kan verificeres af enhver, som kender A's offentlige nøgle.

I 1988 blev der imidlertid foreslået et nyt paradigme for autenticitetsprotokoller — de såkaldte uafviselige signaturer (engelsk: "undeniable signatures", se [CA90]). Disse signaturer udgør en mellemting mellem identifikations-systemer og digitale signaturer, idet A giver modtageren en signatur, som kun kan verificeres med A's hjælp. Specielt kan modtageren ikke vise den til andre uden A's viden. På den anden side kan A ikke uden videre løbe fra signaturen, idet A for at benægte en signatur skal bevise, at den er falsk, hvilket er umuligt, hvis A tidligere har bevist overfor modtageren, at den er korrekt.

Der vil i dette arbejde blive præenteret en anden metode til konstruktion af uafviselige signaturer, som yderligere har den fordel, at A på ethvert tidspunkt kan vælge at ændre enten en enkelt eller samtlige signaturer til sædvanlige digitale signaturer. Der gives desuden en præcis definition af disse konvertible signaturer, og på basis af denne definition vises det, at de

eksisterer, hvis og kun hvis digitale signaturer eksisterer. Dette er en lille smule overraskende, idet konvertible uafviselige signaturer giver A flere muligheder end digitale signaturer.

I forbindelse med alle tre klasser af autenticitets-protokoller nævnt ovenfor er det interessant at undersøge, hvorledes A kan lade sig repræsentere af agenter. Sagt mere præcist kunne A på et tidspunkt ønske at autorisere $n \geq 1$ agenter således, at mindst k af disse ($1 \leq k \leq n$) må være til stede for at repræsentere A.

Denne mulighed er specielt interessant for konvertible, uafviselige signaturer, idet den tillader A at ansætte agenter, som kan hjælpe med at verificere signaturer. Det er dog også et interessant problem i forbindelse med identifikations-systemer samt digitale signaturer, idet løsningerne her kan anvendes for organisationer, hvis hemmelige nøgle er uddelt blandt medlemmerne. Brug af organisationens nøgle kræver herefter medvirken af et fast antal medlemmer (for eksempel mindst halvdelen).

Der kendes metoder til at dele en hemmelighed blandt n personer, således at mindst k af disse må være til stede for at genfinde hemmeligheden, hvorimod færre end k personer ikke kan finde noget information om nøglen. For at beskytte sig mod snyderi vil en agent, som modtager en del af hemmeligheden, dog ofte være interesseret i selv at forsikre sig om, at vedkommende har modtaget en korrekt del af hemmeligheden.

Denne afhandling beskriver to metoder til dette. Den første metode har den fordel, at færre end k personer ikke får noget (Shannon) information om hemmeligheden. Dette er særdeles nyttigt i forbindelse med hemmeligheder, som kræver en høj grad af sikkerhed. Den anden metode er konstrueret med henblik på asymmetriske krypto-systemer, idet den angiver, hvorledes en hemmelig nøgle med tilhørende offentlig nøgle kan uddeles blandt nogle agenter. I denne metode benyttes den offentlige nøgle ved verifikationen af de enkelte dele af den uddelte hemmelighed.

Hvis brug af en således uddelt nøgle kræver, at agenterne først finder den, og derefter bruger den på samme måde, som A ville have gjort, kan nøglen i princippet kun anvendes en gang, thi i så fald behøver disse agenter ikke at mødes fremover for at anvende nøglen.

Der præsenteres derfor protokoller, som tillader agenterne at repræsentere A uden, at nogen af dem herved bliver i stand til at forbedre sin mulighed for at bruge den hemmelige nøgle uden hjælp af mindst $k - 1$ andre agenter.

Der er givet protokoller, som gør det muligt for k agenter at

- identificere sig som A (det vil sige, at de viser, at de tilsammen kan beregne A's hemmelige nøgle);
- konstruere digitale signaturer på A's vegne;
- bevise gyldighed af A's uafviselige signaturer. A kan enten autorisere agenterne til at bevise gyldighed af enkelte signaturer eller samtlige A's uafviselige signaturer.

Det er også muligt at lave systemer, som tillader agenter at konstruere uafviselige signaturer på A's vegne, men disse behandles ikke her. Alle disse protokoller er baseret på den metode til uddeling af en hemmelighed, som udnytter, at en tilhørende offentlig nøgle er kendt.

For at være i stand til at vurdere sikkerheden af agenternes protokoller, er der givet en generel beskrivelse af den situation, hvor et antal agenter deler en nøgle og ønsker at bruge den i en protokol med en anden part. En sådan protokol kaldes et distribueret bevis (engelsk: "distributed proof"). Med udgangspunkt i sædvanlige definitioner af sikkerhed af kryptogranske protokoller defineres sikkerheden af en sådan protokol, og det vises, at de ovennævnte anvendelser er sikre. Yderligere præsenteres en generel konstruktion af distribuerede beviser, som kan bruges til at lave et sikkert distribueret bevis for medlemskab af sprog i NP.

I det ovennævnte er asymmetriske krypto-systemer udelukkende anvendt i forbindelse med autenticitets-protokoller. Disse systemer kan imidlertid også anvendes til hemmeligholdelse af en meddelelse, idet meddelelsen kan enchifreres med den offentlige nøgle, hvorefter kun personer med kendskab til den hemmelige nøgle kan læse meddelelsen. I afhandlingen vises, hvorledes agenter, som deler den hemmelige nøgle, kan dechifrere en givet chiffer-tekst. Denne anvendelse er specielt interessant for organisationer, hvor den hemmelige nøgle er uddelt blandt medlemmerne.

Contents

Chapter 1

Introduction

This thesis was written as part of my Ph.D. study at Aarhus University with Peter Landrock as supervisor. The main object is to study how a number of persons can replace a single person in cryptographic protocols. Some of the results presented here were developed in joint work with Joan Boyar, David Chaum and Ivan Damgård.

Consider a typical cryptographic scenario in which a person, A, has a pair of secret/public keys. This pair allows A to participate in various protocols in which she utilizes the fact that only she knows the secret key (e.g. identification protocols). In general it is necessary to know A's secret key in order to replace her in these protocols. This property is essential in some applications, but it also raises the question, what A should do, if she wants a protocol to be executed (correctly), but is prevented from participating? If A gives her secret key to an agent (e.g. her lawyer), this agent can represent A in all future protocols — even against A's will.

A better strategy for A would therefore be to authorize many agents and require that a certain number of these be present in order to represent A. Then many agents have to cooperate in order to cheat A. This solution is often used in cryptology, and several systems have been developed, which allow A to distribute a secret key among the agents, such that only certain subsets of these can later recover the key. However, the existing literature has not shown, how the agents should actually represent A in practice. In particular, it is desirable that the agents can do this without ever having to find A's secret. This work provides a general (theoretic) solution to this problem, and more efficient protocols are given for various situations.

All protocols in this work are based on the arithmetic in the field $GF(p)$, where p is a prime. Chapter 2 describes the notation which will be used, but it is assumed that the reader is familiar with the basic properties of this field. In order to avoid too many technical definitions, it is also assumed that the reader is familiar with the basic cryptographic notions and in particular the definitions of indistinguishable random variables, simulators and zero-knowledge (as given in [GMR89]).

The first and perhaps the most important problem that A faces when she wants to authorize a number of agents is to distribute her secret key among them. The first part of this work deals with this problem and Chapter 3 and 4 show how this can be done in different situations.

In Chapter 5 we consider the situation, where A uses her secret key to decipher ciphertexts. This problem has received some attention previously, and the main result of this chapter is the development of a scheme that allows the members of an organization to select a pair of secret/public keys to a crypto system such that only certain sets of members can later decipher messages that are encrypted under the organization's public key.

Chapter 6 introduces the general framework of distributed proofs, and as an application of this notion it is shown how A can be represented by agents in an identification and a digital signature scheme.

One of the primary objects of this work is the application of distributed proofs to (selectively) convertible undeniable signatures. David Chaum has recently suggested the notion of undeniable signatures, which are signatures that cannot be verified without the help of the signer. These signatures are briefly described in Chapter 7, and in Chapter 8 they are extended to (selectively) convertible signatures which are undeniable signatures with the added property that the ability to verify signatures is separated from the ability to construct new signatures. They are particularly suited for the framework considered here, as they allow the signer to authorize agents, which can verify his undeniable signatures without being able to construct new signatures. Chapter 9 describes how this can be done, and Chapter 10 concludes this work.

I wish to thank David Chaum for suggesting to use agents to verify undeniable signatures, and Joan Boyar for many discussions during my study. I also want to thank the external referee, Claude Crépeau, for reading this thesis very carefully and giving many suggestions for improvements. Special thanks to my supervisor, Peter Landrock, and to Ivan Damgård who has been a great help to me.

Chapter 2

Notation and Assumption

This chapter presents some notation which is used repeatedly in this work, and a few assumptions about problems related to that of computing discrete logarithms are formalized.

Notation

Whenever we say that an element is chosen at random in some set, A, we mean with respect to the uniform distribution and independently of everything else (unless otherwise stated). If A is finite, an element in A will therefore be selected with probability $\frac{1}{\#A}$, where $\#A$ in general denotes the cardinality of the (finite) set A.

For any string of bits $x \in \{0,1\}^*$ let $|x|$ be the length of x, and for any natural number $n \in \mathbb{N}$ let $|n|$ be the length of the binary representation of n.

Throughout this work p denotes a large prime. If all prime factors of $p-1$ are small, the discrete logarithm modulo p can be computed in time $O(|p|^2)$ (see [PH78]). We will therefore always require that $p-1$ has a large prime factor, and this factor will be denoted by q.

For such a pair of primes (p,q) the unique subgroup of $\mathbb{Z}_p^*$ of order q will be called G_q, and g will always be a generator of G_q. In the protocols that will be described later, it will sometimes be necessary to verify that an element $a \in \mathbb{Z}_p^*$ belongs to G_q. This is very easy to do as

$$\forall a \in \mathbb{Z}_p^* : a \in G_q \Leftrightarrow a^q = 1$$

For $a, b \in \mathbb{Z}_p^*$ the least non-negative integer, e, such that $b = a^e \bmod p$ is denoted $\log_a b$. If such an integer does not exist, $\log_a b$ is undefined, but due to the fact that any element $a \neq 1$ in G_q generates G_q, $\log_a b$ is always defined for such an a and $b \in G_q$.

The Euler totient function is denoted φ. Thus $\varphi(n)$ is the number of integers between 0 and n relatively prime to n for any $n \in \mathbb{N}$.

A function $f : \mathbb{N} \rightarrow \mathbb{R}_+ \cup \{0\}$ is called *negligible*, if for all $c > 0$:

$$f(n) < n^{-c}$$

for all n sufficiently large. An event, which occurs with probability $1 - f(n)$ where f is negligible, is said to have *overwhelming* probability.

Assumptions

As mentioned above it will be assumed that it is hard to compute discrete logarithms modulo p. This assumption has been widely used in cryptography during the last 15 years (see for example [DH76], [BM84], [EG85], [Bet88], [Sch90] and [BM91]).

In the following we shall modify the usual assumption about the intractability of computing discrete logarithms a little, as we shall assume that this problem is hard when $p-1$ has a single large prime factor (as mentioned above $p-1$ must have at least one large prime factor). However, this does not seem to be a stronger assumption than the usual assumption, as it is generally believed that such primes are among the hardest for computing discrete logarithms. Before stating the formal intractability assumptions we note that

$$\mathbb{Z}_p^* \cong G_p \times C_{\frac{p-1}{q}}$$

where $C_{\frac{p-1}{q}}$ is a cyclic group of order $\frac{p-1}{q}$. If q is much larger than $\frac{p-1}{q}$, the problem of computing discrete logarithms in $\mathbb{Z}_p^*$ can be reduced to that of computing discrete logarithms in G_q. Therefore we will state our assumption about the intractability of computing discrete logarithms in $\mathbb{Z}_p^*$ for the group G_q.

Consider a family of probabilistic polynomial size circuits $(C_n)_{n \in \mathbb{N}}$ aiming at computing $\log_g h$ for $h \in G_q$. C_n takes $4n$ bits as input where $n = |p|$

plus a number of random bits. Let $P_C(p, q, g)$ be the probability that

$$C_n(p, q, g, h) = \log_g h$$

when $h \in G_q$ is chosen at random. This probability is over the random bits of the circuit and the choice of h. Then it will be assumed that $P_C(p, q, g)$ is negligible as a function of $|q|$:

Assumption DLP

For all families $(C_n)_{n \in I\!N}$ as above, for all $c > 0$, for all sufficiently large primes p and q and all generators, g:

$$P_C(p, q, g) < |q|^{-c}$$

The Diffie-Hellman problem (DH) is that of computing g^{xy} when g^x and g^y are given. For primes, p, such that $\varphi(\varphi(p))$ has only small prime factors, this problem is equivalent to DLP (see [dB90]). However, in general it is not known whether it is possible to solve DH without being able to compute discrete logarithms.

Let $(C_n)_{n \in I\!N}$ be a family of probabilistic polynomial size circuits such that C_n has $5n$ input bits (plus the random bits) and n output bits and let

$$P_C(p, q, g) = Prob[C_n(p, q, g, g^x, g^y) = g^{xy}].$$

The probability is over the random choices of $x, y \in \mathbb{Z}_p^*$ and the random bits of the circuit.

Assumption DH

For all families $(C_n)_{n \in I\!N}$ as above, for all $c > 0$, for all sufficiently large primes p and q and all generators, g:

$$P_C(p, q, g) < |q|^{-c}$$

Another problem related to that of computing discrete logarithms is to recognize whether to pairs of elements $(g, h) \in G_q^2$ and $(a, b) \in G_q^2$ satisfies

$$\log_g h = \log_a b.$$

This problem is known as the simultaneous discrete logarithm problem (see [CEG87]), and it is easy to solve if one can compute discrete logarithms. However it is not known if the converse also holds.

Let $(C_n)_{n \in I\!N}$ be a family of probabilistic polynomial time circuits such that C_n has $6n$ input bits (plus the random bits) and one output bit.

Assumption SDL

For all families $(C_n)_{n \in I\!N}$ as above, for all $c, d > 0$, for all sufficiently large primes p and q ($n = |p|$) for all generators g and for at least a fraction $1 - |q|^{-d}$ of $a \in G_q$ ($a \neq 1$):

$$\begin{aligned} &|Prob[C_n(p,q,g,h,a,b) = 1 \mid \log_a b = \log_g h] - \\ &Prob[C_n(p,q,g,h,a,b) = 1 \mid \log_a b \neq \log_g h] \mid |q|^{-c}. \end{aligned}$$

The probabilities are over the random choices of $h, b \in G_q$ and the random bits of C_n.

This assumption says that an algorithm for solving the simultaneous discrete logarithm problem cannot do much better than trying to guess $\log_g h, \log_g a, \log_a b$ or $\log_h b$.

In the preceding assumptions we have suggested that the problem in question is hard for all pairs of sufficiently large primes. It is however sufficient that the problems are hard for all but a negligible fraction of the large primes, since the probability of generating a “bad” pair can be neglected in that case.

As for the generation of p and q it is sufficient in practice to use a probabilistic test of primality ([Rab80], [SS77] and [BDL91]). Furthermore, if one first generates q and determines p as the lest prime equivalent to 1 modulo q, then heuristics show that (see [Wag79])

$$p < q \log^2 q.$$

Thus $|p| \leq |q| + 2 \log|q|$, which is sufficient for the above assumptions.

Chapter 3

Verifiable Secret Sharing

The object of this chapter is to present a non-interactive verifiable secret sharing scheme in which no information about the secret is revealed to unauthorized groups. First, a formal definition of verifiable secret sharing is presented, and the history of such schemes is sketched. After a presentation of the scheme a few applications are described.

The results in this chapter are also described in [Ped91b]

3.1 Secret Sharing Schemes

Let $\mathcal{S}$ be a finite set of secrets, and let Γ be a set of subsets of $\{1, 2, \ldots, n\}$. A *secret sharing scheme* for the *access structure*, Γ, describes how a *dealer*, D, having a *secret*, $s \in \mathcal{S}$, can send information (called *shares*) to n participants (*shareholders*), $P_1, \ldots, P_n$ such that the following holds for all $A \subseteq \{1, \ldots, n\}$:

- If $A \notin \Gamma$ then $(P_i)_{i \in A}$ get no information about s.
- If $A \in \Gamma$ then the participants in A can compute s in polynomial time (in a security parameter).

Usually the first requirement refers to Shannon information (see [Sha48]), but it can also be relaxed to mean that the participants cannot compute (any bit of) s. Obviously, this definition only makes sense, if Γ is monotonely increasing:

$$\forall A, B \subseteq \{1, \dots, n\} : A \in \Gamma \wedge A \subseteq B \Rightarrow B \in \Gamma.$$

Here, we shall only be concerned with (k,n)-*threshold* schemes. For $1 \leq k \leq n$, such a scheme is defined by the access structure:

$$\Gamma_k = \{A \subseteq \{1, \dots, n\} \mid \#A \geq k\}.$$

The first published descriptions of secret sharing schemes were both threshold schemes (see [Sha79] and [Bla79]). These secret sharing schemes are in some sense equivalent (see [Kot85]), but in this work we shall only use Shamir's scheme which is briefly described now.

Consider a finite field, $I\!F$, such that each $s \in \mathcal{S}$ can be uniquely represented as an element in $I\!F$. Hence, each secret can be identified with an element in $I\!F$. In order to distribute $s \in I\!F$ among $P_1 \dots P_n$ (where $n < |I\!F|$) the dealer chooses a polynomial $f \in I\!F[x]$ of degree at most $k-1$ satisfying $f(0) = s$. Participant P_i receives $s_i = f(p_i)$ as his private share, where $p_i \in I\!F \setminus \{0\}$ is public information about P_i ($p_i \neq p_j$ for $i \neq j$).

Let $0 \leq l < k$ and consider l shares $s_{i_1}, s_{i_2}, \dots, s_{i_l}$. These shares contain no information about the secret, because for every $s \in I\!F$ there are exactly $|I\!F|^{k-l-1}$ polynomials of degree at most $k-1$, which maps 0 to s and p_{i_j} to s_{i_j} for $j = 1, 2, \dots, l$.

Furthermore, any k persons $(P_{i_1}, \dots, P_{i_k})$ can recover the secret by first finding f using the formula:

$$\begin{aligned} f(x) &= \sum_{j=1}^{k} (\prod_{h \neq j} \frac{x - p_{i_h}}{p_{i_j} - p_{i_h}}) f(p_{i_j}) \\ &= \sum_{j=1}^{k} (\prod_{h \neq J} \frac{x - p_{i_h}}{p_{i_j} - p_{i_h}}) s_{i_j} \end{aligned}$$

Hence

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

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

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

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

3.2 Verifying the Shares

Two types of cheating can occur in a secret sharing scheme. Either the dealer can send incorrect shares to some of the participants (i.e. a share different from that prescribed by the dealers distribution algorithm), or a participant can supply an incorrect share, when a group of participants are trying to recover the secret (see [BS88] for a further discussion of this).

When designing a secret sharing scheme one should be aware that both of these frauds can occur, because it cannot be assumed in general that the n participants and the dealer trust each other completely. Hence, one should use a *verifiable secret sharing scheme* (see [CGMA85]). This is a secret sharing scheme in which the participants are allowed to talk with each other and the dealer in order to verify the correctness of their shares. Furthermore, each participant must be able to prove that his share is correct when a number of participants are going to recover the secret.

A general communication model for verifiable secret sharing allows the dealer and each shareholder to

- send secret messages to each other participant; and
- broadcast messages to the other participants.

In this work we are interested in *non-interactive* verifiable secret sharing. In such a scheme the dealer is allowed to send a secret message to each participant and to broadcast messages, but the shareholders cannot talk to each other or the dealer when verifying a share. Each shareholder is only allowed to broadcast a single message (bit). Namely, whether the share is accepted or not.

In order to define verifiable secret sharing formally we first note that the definition of secret sharing schemes requires the existence of two polynomial time computable functions, *distribute* and *combine*, where P_i gets the share s_i of s, if

$$\textit{distribute}(r, s, p_1, p_2, \ldots, p_n) = (s_1, s_2, \ldots, s_n).$$

Here $r \in \{0,1\}^*$ is a string of random bits. For all $i_1, \ldots, i_k \in \{1, \ldots, n\}$ where $i_j \neq i_l$ for $j \neq l$ the following must hold:

$$\mathit{combine}[(p_{i_1}, s_{i_1}), \ldots, (p_{i_k}, s_{i_k})] = s.$$

Furthermore, it must be infeasible to compute s from fewer than k shares. In particular, the scheme is said to be unconditionally secure for the dealer, if fewer than k shares contain no Shannon information about s.

In a non-interactive verifiable secret sharing scheme, let *proof* be the message that D broadcasts to all participants. Thus in such a scheme

$$\mathit{distribute}(r, s, p_1, p_2, \ldots, p_n) = (s_1, s_2, \ldots, s_n, \mathit{proof}).$$

Let furthermore $\mathit{verify}(p_i, s_i, \mathit{proof})$ be a polynomial time computable predicate which P_i uses to decide whether his share should be accepted or not. Then the following must hold:

Definition 3.1
Let K be a positive integer.
Three polynomial time (in K) computable functions *distribute*, *combine* and *verify* constitute a non-interactive verifiable (k, n)-threshold scheme with security parameter K, if

1. For all secrets $s \in \mathcal{S}$, and for all $r \in \{0,1\}^*$. If

   $$\mathit{distribute}(r, s, p_1, p_2, \ldots, p_n) = (s_1, s_2, \ldots, s_n, \mathit{proof})$$

   then for all $i_1, i_2, \ldots, i_k \in \{1, 2, \ldots, n\}$ ($i_j \neq i_l$ for $j \neq l$)

   $$\mathit{combine}[(p_{i_1}, s_{i_1}), \ldots, (p_{i_k}, s_{i_k})] = s$$

   and for all $i \in \{1, 2, \ldots, n\}$:

   $$\mathit{verify}(p_i, s_i, \mathit{proof}) = 1.$$

   If $\mathit{verify}(p_i, s_i, \mathit{proof}) = 1$ then P_i accepts the share s_i and otherwise the share is rejected.

2. For all secrets $s \in \mathcal{S}$ and all $r \in \{0,1\}^*$, if

   $$\mathit{distribute}(r, s, p_1, p_2, \ldots, p_n) = (s_1, s_2, \ldots, s_n, \mathit{proof})$$

   then it is not feasible to find s from fewer than k of the s_i's and *proof*

3. For any polynomial time (in K) algorithm, A.

   If A on input $r \in \{0,1\}^*$ outputs n shares, $s_1, s_2, \ldots, s_n$ and a message *proof*, then the following holds.

   For all subsets S_1 and S_2 of $\{1, \ldots, n\}$ of size k such that

   $$\forall i \in S_1 \cup S_2 : \ \mathit{verify}(p_i, s_i, \mathit{poof}) = 1$$

   the following holds except with negligible (in K) probability:

   $$\mathit{combine}[(p_i, s_i)_{i \in S_1} = \mathit{combine}[(p_i, s_i)_{i \in S_2}].$$

   The probability is over the random choices of r.

A share is called *correct*, if it is accepted by *verify*.

Definition 3.1 does not refer to the secret when defining the correctness of a share. This is in accordance with the fact that a single participant has no information about s during the verification, and s could therefore be whatever the dealer claims. After the execution of the verification protocol the secret is defined as the value, which any k participants with correct shares will find when combining their shares. If the dealer succeeds in distributing inconsistent shares, this is not well-defined, but Definition 3.1 guarantees that the dealer will be caught almost always, if he tries to cheat.

This definition of verifiable secret sharing schemes allows that an infinitely powerful dealer distributes incorrect shares. We shall later return to this point.

3.3 Related Work

As previously mentioned the first two secret sharing schemes were published in 1979 ([Sha79] and [Bla79]), and since then much work has been put in to the investigation of such schemes (see [Sim90] for a list of references).

As for verifiable secret sharing, the first method that prevents cheating in a secret sharing scheme was described in [MS81]. Here an error-correcting code is applied so that the secret can be recovered even if some of the shares are incorrect (but the scheme does not allow a shareholder to obtain a proof that his share is correct). This is also true for Tompa and Woll's modification of the Shamir scheme presented in [TW86]. Here the set of legal secrets is a small subset of the fields $\mathbb{F}$, so that it is very unlikely that incorrect shares will produce a legal secret when combined.

There has been other suggestions for verifying that the participants obtain the correct secret when combining their shares, but the first method that allowed the participants to verify their shares *before* trying to recover the secret was presented in [CGMA85]. Unfortunately, this scheme requires exponential in k long messages.

Here we shall not mention all the interactive verifiable secret sharing schemes that have been constructed, but only remark that such schemes have been a very important tool in the construction of unconditionally secure protocols for multi-party computations (see [CCD88] and [BGW88]). Both of these schemes require that less than $\frac{n}{3}$ of the shareholders are dishonest, and they do not need the broadcast channel, because it can be simulated using a polynomial time protocol for Byzantine Agreement (see [LSP82] and [DS82]). In the scheme of [CCD88] the dealer has an exponentially small probability of cheating, whereas [BGW88] achieves zero error probability. The basic ideas of [BGW88] are reused by Micali and Rabin in [MR91] to obtain a scheme with the same properties, and which only needs an expected number of rounds.

It can be argued (see [CCD88]), that even if a broadcast channel is given, it is impossible to achieve an unconditionally secure scheme with zero error probability, if more than one third of the participants are dishonest. However, if the dealer is allowed an exponentially small probability of distributing inconsistent shares it is possible to construct a scheme, which allows up to $\frac{n}{2}$ dishonest participants (see [RB89]).

To the knowledge of this author, the first non-interactive verifiable secret sharing scheme was presented in [Fel87]. Here the dealer publishes probabilistic encryptions of the polynomial used to compute the shares, and due to a homomorphism property of the encryption scheme the shares can be verified. The scheme works for any probabilistic encryption scheme in which a number of bits (say l) are encrypted as the “hard-core” bits of a one-way function with homomorphic properties. As an example it is suggested to use the one-way function

$$f : x \mapsto \alpha^x \bmod p$$

where α generates $\mathbb{Z}_p^*$. This function has the property that

$$f(x + y) = f(x)f(y).$$

For $l = O(\log|p|)$ there is a function

$$pred : \{1, \dots, p-1\} \rightarrow \{0,1\}^l$$

such that $pred(x)$ is hard to compute given $f(x)$. Thus an l-bits message, m, is encrypted as $f(x)$ where $pred(x) = m$. In Section 4.1 we present a scheme which is very similar to that of Feldman when this one-way function is used. The reader is referred to this section and [Fel87] for more details, but we remark here that due to the encryption scheme, this scheme has the following properties

- Even an all powerful dealer cannot distribute inconsistent shares; and
- After a secret has been distributed its security depends on a computational assumption.

3.4 An Information-Theoretic Secure Scheme

Most literature dealing with secret sharing schemes requires that unauthorized groups of shareholders get no Shannopn information about the secret. This has the advantage that the security of the secret is independent of future developments in algorithms and hardware. It is therefore also important to be able to distribute a secret verifiably in a (k, n)-threshold scheme in such a way, that fewer than k shares contain no information about the secret.

This section presents such an information theoretic secure secret sharing scheme. The idea is to replace the encryption scheme in [Fel87] by a commitment scheme which is unconditionally secure for the committer. By a proper choice of the commitment scheme this also allows a more efficient scheme and it ensures that the Shamir scheme is used in a field. This is important as the correctness of the Shamir scheme depends on the fact that the polynomial is chosen over a field.

The Commitment Scheme

Let g and h be elements of G_q such that $\log_g h$ is unknown to all parties. These elements can either be chosen by a trusted center, when the system is initialized, or by (some of) the participants using a coin-flipping protocol.

The committer commits himself to an $s \in \mathbb{Z}_q$ by choosing $t \in \mathbb{Z}_q$ at random and computing

$$BC(s,t) = g^s h^t$$

(this is a slight variation of a scheme suggested in [BCP]). Such a commitment can later be opened by revealing s and t. The following theorem is very easy to prove and shows that $BC(s,t)$ reveals no information about s, and that the committer cannot open a commitment to s as $s' \neq s$ unless he can find $\log_g (h)$.

Theorem 3.2
$BC(s,t)$ is uniformly distributed in G_q for any $s \in \mathbb{Z}_q$ and for randomly uniformly chosen $t \in \mathbb{Z}_q$.
If $s, s' \in \mathbb{Z}_q$ satisfies $s \neq s'$ and $BC(s,t) = BC(s',t')$ for some $t, t' \in \mathbb{Z}_q$, then $t \neq t' \bmod q$ and

$$\log_g h = \frac{s - s'}{t' - t} \bmod q.$$

Even though it will not be used in the following we mention that it is quite easy to prove one's ability to open two commitments as the same value without revealing this value. Let namely

$$\beta = BC(s,t) \quad \text{and} \quad \beta' = BC(s,t')$$

where $t \neq t'$. Anyone who knows an r such that $\beta/\beta' = h^r$ can open β as s if and only if he can also open β' as s. By revealing $r = t - t'$ it is therefore possible to prove equality of the contents of two commitments. Furthermore, $t - t'$ does not contain any information about s.

It is not that easy to prove, that commitments to two different values really do contain different values. In particular, the proof of [BCC88] that two blobs contain different bits given a method of proving equality does not generalize to this commitment scheme. In [CHP91] a protocol for proving inequality of the content of two commitments is described.

Finally consider the efficiency of the commitment scheme. As mentioned in Chapter 2, p and q can presumably be constructed such that $p \leq q(\log q)^2$. Thus a commitment to $|q|$ bits requires at most $|q| + 2 \log |q|$ bits. Furthermore, by first computing the product gh a commitment to s can be computed in less than $2|q|$ multiplications modulo p or less than two multiplications pr. bit of s. Thus the commitment scheme is quite efficient with respect to the size of commitments as well as the amount of computation required.

The Scheme

For convenience we assume that i is the public information of P_i (i.e. $p_i = i$ for $1 \leq i \leq n$). By the fact that $\mathbb{Z}_q$ is a field, the dealer, D, can distribute $s \in \mathbb{Z}_q$ as follows:

1. D chooses $F \in \mathbb{Z}_q[x]$ of degree at most $k-1$ satisfying $F(0) = s$, and computes $s_i = F(i)$ for $i = 1, 2, \ldots, n$.

   Let $F(x) = F_0 + F_1 x + \ldots + F_{k-1}x^{k-1}$, where $F_0 = s$. D chooses $G_0, G_1, \ldots, G_{k-1} \in \mathbb{Z}_q$ at random and uses G_i when committing to F_i for $i = 0, 1, \ldots, k-1$. D broadcasts *proof* $= (E_0, \ldots, E_{k-1})$ where

   $$E_i = BC(F_i, G_i) \quad \text{for} \;\; i = 0, 1, \ldots, k-1.$$

2. Let $G(x) = G_0 + G_1 x + \cdots + G_{k-1}x^{k-1}$ and let $t_i = G(i)$ for $i = 1, \ldots, n$. Then D sends (s_i, t_i) secretly to P_i for $i = 1, 2, \ldots, n$.

When P_i has received his share, (s_i, t_i), he verifies that *proof* is of the right form and that $\mathit{verify}(i, (s_i, t_i), \mathit{proof}) = 1$ where

$$\mathit{verify}(i, (s_i, t_i), \mathit{proof}) = 1 \Longleftrightarrow BC(s_i, t_i) = \prod_{j=0}^{k-1} E_j^{i^j}.$$

Lemma 3.3
Let $S \subset \{1, 2, \ldots, n\}$ be a set of size k such that $\mathit{verify}(i, (s_i, t_i), \mathit{proof}) = 1$ for all $i \in S$. Then the k participants $(P_i)_{i \in S}$ can find a pair (s', t') such that $E_0 = g^{s'} h^{t'}$.

Proof
Let $S \subseteq \{1, 2, \ldots, n\}$ of size k be given, The participants in S first find the two unique polynomials F' and G' of degree at most $k-1$ satisfying

$$\begin{aligned} F'(i) &= s_i \\ G'(i) &= t_i \end{aligned}$$

for $i \in S$. Now let $h = g^d$. Then

$$g^{F'(i)+dG'(i)} = g^{s_i+dt_i} = BC(s_i, t_i)$$

for $i \in S$. Thus $(F' + dG')(x)$ is the unique polynomial of degree at most $k-1$ mapping i to $s_i + dt_i$. Let $E_j = g^{e_j}$. Then the polynomial

$$e(x) = \sum_{j=0}^{k-1} e_j x^j$$

satisfies $e(i) = s_i + dt_i$ for $i \in S$. Thus

$$e(x) = (F' + dG')(x)$$

and in particular

$$E_0 = g^{e(0)} = g^{F'(0)+dG'(0)} = g^{F'(0)} h^{G'(0)}.$$

Therefore it is sufficient to put $s' = F'(0)$ and $t' = G'(0)$. ■

The members in S do not have to find F' in order to find the secret. It is more efficient to use the formula

$$s = \sum_{i \in S} a_i s_i \quad \text{where} \quad a_i = \prod_{j \in S, j \neq i} \frac{j}{j-i}.$$

Note that they can also find G_0 by the formula

$$G_0 = \sum_{i \in S} a_i t_i$$

It follows from the above arguments that if the dealer follows the protocol then

- *verify*$(i, (s_i, t_i), \textit{proof}) = 1$ for all i; and
- s can be computed in polynomial time from any k shares.

This shows that the first requirement in Definition 3.1 is fulfilled. The next two theorems consider the remaining two requirements.

Theorem 3.4

Requirement 3) of Definition 3.1 is satisfied under assumption DLP.

Proof
Consider a dealer, who has constructed n shares $(s_i, t_i)_{i=1,\dots,n}$ and a message, *proof*. Let S and S' be two subsets of $\{1, 2, \dots, n\}$ such that all participants in S and S' have accepted their shares in the verification. By Lemma 3.3 above the members in S (S') can together find a pair (s, t) ((s', t')) such that $E_0 = BC(s, t)$ $(= BC(s', t'))$.

If $s \neq s'$ the dealer can therefore find $\log_g h$ by first finding S and S' (see below) and then computing $\log_g h$ as described in Theorem 3.2.

As the shares are consistent if and only if there is a polynomial, f, of degree at most $k - 1$ such that

$$f(i) = s_i \quad \text{for } i = 1, 2, \dots, n$$

the dealer can find S and S' as follows, if the shares are inconsistent:

```
1. Let $S := \emptyset$ and $i := 1$.
2. While $\#S < k$ and $i \leq n$ do:
   (a) if $(s_i, t_i)$ is correct: put $S := S \cup \{i\}$.
   (b) put $i := i + 1$.
3. If $i > n$: stop (at most $k$ correct shares).
4. Let $f$ be the unique polynomial of degree at most $k - 1$ such that
   $f(i) = s_i$ for all $i \in S$.
5. Let $S' \subset S$ such that $\#S' = k - 1$.
6. While $\#S' < k$ and $i \leq n$ do:
   (a) if $(s_i, t_i)$ is correct and $f(i) \neq s_i$: put $S' := S' \cup \{i\}$.
   (b) put $i := i + 1$.
7. If $\#S' = k$: output $S$ and $S'$.
   Otherwise output "no inconsistent shares".
```

■

As a consequence of Theorem 3.4 all the correct shares are consistent unless the dealer succeeds in finding $\log_g(h)$ *before* the last share has been sent. Furthermore, Theorem 3.2 implies, that a shareholder cannot supply an incorrect share, when k shareholders are going to recover the secret, unless he can find $\log_g h$. Note however, that the shareholder usually will have much more time to find $\log_g h$, than the dealer has.

The following theorem shows, that fewer than k participants get no Shannon information about the secret. Hence requirement 2 in Definition 3.1 is satisfied unconditionally. For any subset $S \subseteq \{1, 2, \ldots, n\}$, let $view_S$ denote the messages, that the members of S see:

$$view_S = (E_0, E_1, \ldots, E_{k-1}, (s_i, t_i)_{i \in S}).$$

Theorem 3.5
For any $S \subset \{1, 2, \ldots, n\}$ of size at most $k-1$ and any $view_S$

$$Prob[D \text{ has secret } s \mid view_S] = Prob[D \text{ has secret } s]$$

for all $s \in \mathbb{Z}_q$.

Proof
It is sufficient to prove the theorem in the case where S has size $k-1$. If $k-1$ parties get no information about s then neither does fewer than $k-1$ parties.

Let $S = \{1, \ldots, k-1\}$ and $view_S = (E_0, E_1, \ldots, E_{k-1}, (s_i, t_i)_{i=1,\ldots k-1})$. For every $s \in \mathbb{Z}_q$ there is exactly one $t \in \mathbb{Z}_q$ such that $E_0 = BC(s, t)$ and there is exactly one polynomial F of degree at most $k-1$ satisfying

$$\begin{aligned} F(0) &= s \\ F(i) &= s_i \qquad \text{for } i = 1, \ldots, k-1 \end{aligned}$$

and exactly one polynomial G of degree at most $k-1$ satisfying

$$\begin{aligned} G(0) &= t \\ G(i) &= t_i \qquad \text{for } i = 1, \ldots, k-1 \end{aligned}$$

Let $F(x) = s + F_1 x + \cdots + F_{k-1} x^{k-1}$ and $G(x) = t + G_1 x + \cdots + G_{k-1} x^{k-1}$. In order to show that $view_S$ does not contain any information about the secret it is sufficient to show that F and G satisfies

$$BC(F_i, G_i) = E_i \quad \text{for } i = 1, \ldots, k-1.$$

Similar to the proof of Lemma 3.3 this follows from the fact that there is one and only one polynomial, f, of degree at most $k-1$ satisfying $(s_0 = s, t_0 = t)$

$$g^{f(i)} = g^{s_i} h^{t_i}$$

for $i = 0, 1, \ldots, k-1$ and the polynomial $F + dG$ satisfies this for $d = \log_g h$.

$\blacksquare$

We state here without proof the following theorem (which can be proved by the same method as the proof of Theorem 4.3 in Section 4.1).

Theorem 3.6
There is a probabilistic polymomial time machine, M, which on input $S \subset \{1, 2, \ldots, n\}$ and $(s_i, t_i)_{i \in S}$ produces $view_S$ with the same distribution as the dealer does.

This theorem shows that any fixed set of at most $k-1$ shareholders can actually generate the messages received from the dealer with the same distribution. Theorem 3.6 is weaker than Theorem 3.5, because the latter says that no matter how the set $S \subseteq \{1, 2, \ldots, n\}$ of $k-1$ persons is generated, the shares $(s_i, t_i)_{i \in S}$ contain no information about s. Theorem 3.6 is included here because it shows that the secret sharing scheme can simulated and therefore it can be used in zero-knowledge protocols (see Section 6.3).

3.5 Efficiency and Security

In this sections the computational requirements of the scheme are estimated and the scheme is compared with that of [Fel87].

First consider the size of the secret shares. As a secret is $|q|$ bits long, the information rate (see [BD90]) is

$$\frac{\text{size of secret}}{\text{size of share}} = \frac{1}{2}$$

Ignoring the time needed to evaluate $F(x)$ and $G(x)$ (this is reasonable as the polynomials are only evaluated on small arguments), the dealer has to compute k commitments in order to distribute a share. This requires less than $2|q|k$ multiplications modulo p or approximately $2k$ multiplications pr bit of the secret.

The verification requires $k-1$ exponentiations modulo p and the computation of one commitment. This can be done in less than (again ignoring the computation of i^j for $j = 1, \ldots, k-1$)

$$2|q|(k-1) + 2|q| + (k-1) \approx (2|q|+1)k$$

multiplications. This is however, a pessimistic estimate as many of the exponents in the exponentiations are rather small (in particular, for P_1 they all equal 1).

For a moment return to Feldman's scheme. Using this scheme and a probabilistic encryption scheme based on discrete logarithms in $\mathbb{Z}_p$, the computational requirements when distributing an l-bits secret are very similar to the requirements in our scheme when distributing a $|q|$-bits secret (note that $|q| \approx 2^l$).

With respect to security the two schemes are dual to each other, because the encryption schemes used in [Fel87] only protects the secret under the assumption that the one-way function cannot be inverted. However, even an infinitely powerful dealer cannot distribute incorrect shares. In contrast, the new scheme protects the privacy of the secret unconditionally, but the correctness of the shares depends on a computational assumption.

Having these two secret sharing schemes it is natural to ask for a non-interactive scheme in which

- no information about the secret is revealed; and
- even an infinitely powerful dealer cannot compute inconsistent shares.

However, the following shows that such a scheme is impossible in the model which is used here. As before let *proof* denote all the information which the dealer broadcasts in a non-interactive secret sharing scheme, let s_i be the secret share which is sent to P_i, and let $\mathit{verify}(i, s_i, \mathit{proof})$ denote the verification predicate, which P_i computes in order to verify his share. Now consider $P_1, \ldots, P_{k-1}$ and assume that they have received correct shares. Let S_k be the set of shares which P_k will accept:

$$S_k(\mathit{proof}) = \{s_k \mid \mathit{verify}(k, s_k, \mathit{proof}) = 1\}.$$

As even an all powerful dealer cannot find inconsistent shares the following holds for all $s_k \in S_k$:

$$combine[(1, s_1), (2, s_2), \ldots, (k, s_k)] = s$$

for some secret s. This means that $P_1, \ldots, P_{k-1}$ can find the secret by guessing a secret share $s_k \in S_k$ and then combine their own shares with s_k.

In particular note that $S_k(\mathit{proof})$ is in *NP*, if *verify* can be computed in polynomial time. Thus $P_1, \ldots, P_{k-1}$ need "only" nondeterministic polynomial time in order to find the secret if the scheme is unconditionally secure for the shareholders. Similarly, a dishonest dealer can always distribute inconsistent shares in nondeterministic polynomial time, if the scheme reveals no information about the secret.

We close this section with a short remark on the models of adversarial participants. Feldman considers

- a *static model* in which the set of cheating participants is constant; and
- a *dynamic model* in which the cheating participants are chosen depending on *proof*.

Feldman used the ideas of zero-knowledge when defining security of secret sharing schemes and required that the dealer can be simulated. In order to prove security in the dynamic model according to this definition it was necessary to permute the shares, but Feldman conjectured that the scheme would also be secure without this complication. In our scheme we are able to prove security without permuting the shares because the proof is based on information theory rather than simulations. As the two schemes are quite similar, this seems to support Feldman's claim that it is not necessary to permute the shares in his scheme. Note that Theorem 3.6 says that our scheme is secure in the static model affording to Feldman's definition.

3.6 Computing on Shared Secrets

As mentioned in Section 3.3 verifiable secret sharing is an important tool in the construction of secure protocols for multiparty computations. In particular, the constructions in [BGW88] and [CCD88] both utilize the fact that it is easy to compute linear combinations of shared secrets. In this section we show that this is also true if the secret sharing scheme presented here is used, and an application of this property is given.

Linear Combinations

Assume that two secrets s' and s'' have been distributed as described in Section 3.4. In particular let (s'_i, t'_i) and (s''_i, t''_i) be P_i's share of s' and s'', respectively, and let $(E'_0, E'_1, \ldots, E'_{k-1})$ and $(E''_0, E''_1, \ldots, E''_{k-1})$ be the broadcast messages when the two secrets were distributed.

Each P_i can compute $(E_0, E_1, \ldots, E_{k-1})$ corresponding to a verifiable distribution of $s = s' + s'' \bmod q$ as

$$E_j = E'_j E''_j \quad \text{for } j = 0, 1 \ldots k-1.$$

Furthermore, P_i's select share, (s_i, t_i), of s is given by

$$\begin{array}{rcl} s_i & = & s'_i + s''_i \bmod q \\ t_i & = & t'_i + t''_i \bmod q \end{array}$$

It is easy to see that if both (s'_i, t'_i) and (s''_i, t''_i) are correct shares (satisfy *verify*) then (s_i, t_i) is also a correct share of s; i.e.

$$g^{s_i} h^{t_i} = E_0 E_1^i \ldots E_{k-1}^{i^{k-1}}.$$

If, instead, s is computed as $s = as' \bmod q$ for some $a \in \mathbb{Z}_q^*$, then P_i can compute his share (s_i, t_i) and $(E_0, E_1, \ldots, E_{k-1})$ as follows

$$\begin{array}{rcl} E_j & = & E'^{a}_j \quad \text{for } j = 0, 1, \ldots, k-1 \\ s_i & = & as'_i \bmod q \\ t_i & = & at'_i \bmod q \end{array}$$

Again, it is easy to see that

$$g^{s_i} h^{t_i} = E_0 E_1^i \ldots E_{k-1}^{i^{k-1}}.$$

In both of the above cases Lemma 3.3 implies that any k shareholders who have accepted their shares of s' and s'' can find a pair (s, t) such that

$$g^s h^t = E_0.$$

Furthermore, it is an immediate consequence of Theorem 3.5 that fewer than k persons have no information about s if s' and s'' are distributed correctly.

Choosing an Anonymous, Shared Secret

In [IS91] it was shown how to select and distribute a secret without a mutually trusted authority, who knows the secret and distributes it. In this section we show how to achieve the same goal with verifiable secret sharing by demonstrating how n participants can select a secret such that none of them knows it and distribute it verifiably among themselves in a (k, n) secret sharing scheme, where $2k - 1 \leq n$. It is not hard to generalize the proposed method to let l person $(2k - 1 \leq l \leq n)$ select and distribute the secret.

Let $P_1, \ldots P_n$ be the n persons who want to choose a secret and distribute it among themselves, and assume that each P_i can make digital signatures. As each participant is going to act as a dealer in the following, it is required that each P_i can broadcast messages and send secret messages to each P_j for $j \neq i$. The protocol for P_i is

1. Choose $s_{i0} \in \mathbb{Z}_q$ at random.
2. Distribute s_{i0} verifiably among $P_1, \ldots P_n$.
   Furthermore P_i signs each secret share and sends the signature with the share.
3. Verify all the received shares. If a share is incorrect, P_i publishes the share and its signature. Then P_i stops.
4. Compute the share (s_i, t_i) of $s = s_{10} + s_{20} + \cdots + s_{n0}$ and the corresponding public information $(E_0, E_1, \ldots, E_{k-1})$ as described above.

It follows from the arguments in the previous subsection that

- (s_i, t_i) is a correct share of s, if P_i has accepted all shares correctly; and
- any k participants can find a pair (s', t') such that $E_0 = E(s', t')$.

Let $S \subseteq \{1, 2, \ldots, n\}$ be a set of k participants. These k participants can force s to be any value they want, if they choose their s_{i0}'s depending on $(s_{i0})_{i \notin S}$ (which they can find).

In the following it is shown that s is uniformly distributed in $\mathbb{Z}_q$, and that fewer than k participants have no information about s.

By saying that a group of participants cooperate we mean, that they can put together all their information and agree on all messages that they send to other participants.

Theorem 3.7
If P_i chooses $s_{i0} \in \mathbb{Z}_q$ uniformly at random and at most $k-1$ of the other parties cooperate, then s is uniformly distributed in $\mathbb{Z}_q$.

Proof
Let $S \subseteq \{1, 2, \ldots, n\}$ be a set of at most $k-1$ cooperating participants and assume that the participants not in S follow the prescribed protocol ($i \notin S$).

First note that since $n \geq 2k-1$, each participant in S sends in step 2 at least k correct shares (oTherwise the protocol is stopped). Since each s_{i0} can be computed from k correct shares, each P_i is able to open his commitment to s_{i0}.

Since no set of at most $k-1$ participants (excluding P_i) get any information about s_{i0}, this implies that no P_j for $j \neq i$ can open his commitment to s_{i0} depending on the value of s_{i0}. As P_i chooses s_{i0} at random this implies, that

$$s = s_{10} + s_{20} + \cdots + s_{n0}$$

is uniformly chosen in $\mathbb{Z}_q$. ■

As before let view_S be the messages, which the participants in a subset S of $\{1, \ldots, n\}$ see. Thus view_S consist of all messages, which the participants in S receive plus the sequence of random bits they use during the protocol.

Let $\mathit{Prob}[s]$ denote the (a priori) probability that $s \in \mathbb{Z}_q^*$ will be chosen and let $\mathit{Prob}[s \mid \mathit{view}_S]$ denote the (a posteriori) probability that s has been chosen given view_S.

Theorem 3.8
For any $S \subset \{1, 2, \ldots, n\}$ of size at most $k-1$ and any view_S

$$\mathit{Prob}[s] \text{ is chosen } \mid \mathit{view}_S] = \mathit{Prob}[s]$$

for all $s \in \mathbb{Z}_q$, if the participants not in S follow the protocol.

Proof sketch
Under the assumptions in the theorem it follows from Theorem 3.7 that

$$\mathit{Prob}[s] = \frac{1}{q}$$

for each $s \in \mathbb{Z}_q$.

Given $S \subset \{1, 2, \ldots n\}$ of size $k-1$ and $view_S$. For any $s \in \mathbb{Z}_q$ there exists $q^{n-(k-1)-1} = q^{n-k}$ values of $(s_{j0})_{j \notin S}$ such that

$$s = s_{10} + s_{20} + \cdots s_{n0}$$

and, as in the proof of Theorem 3.5, for each of these values of s_{j0} $(j \notin S)$ there is exactly one choice of the F- and G-polynomials for each P_j, which gives the same messages from P_j in step 2. ■

Chapter 4

Secret Sharing in a Public-Key Scenario

The previous chapter showed how a secret can be distributed verifiably such that no information about the secret is revered to non-authorized groups. However, this high level of secrecy is not always needed. If, for example, the secret is the private key in a public key crypto-system, the corresponding public key may contain all Shannon information about the secret. Hence, there is no Shannon information to hide.

In this chapter it is shown how one can take advantage of the public key, when distributing the private key. More precisely we shall see how a secret $s \in \mathbb{Z}_q$ can be shared verifiably in a (k, n)-threshold scheme when the corresponding public key $h = g^s$ is known to all participants. This scheme will be used in Chapter 6 and 9. As in Section 3.6 this secret sharing scheme can also be used to let n participants select a private and public key and distribute the private key among themselves.

The scheme presented now is similar to that of [Fel87] (based on discrete logarithms) except that the encryption scheme is not needed. Furthermore, the shares are not permuted as the simple scheme gives sufficient security in the later applications to distributed proofs.

4.1 Verification using the Public Key

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 (i is the public information about P_i):

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 $f(0) = s$ (hence $f_0 = s$) and then computing

$$s_i = f(i).$$

   Let *proof* $= (g^{f_1}, \ldots, g^{f_{k-1}})$.

2. Send s_i *secretly* to P_i, and *broadcast proof* to all n participants.

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

When all shares have been distributed each P_i first verifies that *proof* has the correct form. Let *proof* $= (E_1, E_2, \ldots, E_{k-1})$. Then each P_i verifies that $\mathit{verify}(i, s_i, \mathit{poof}) = 1$ where

$$\mathit{verify}(i, s_i, \mathit{proof}) = 1 \iff g^{s_i} = h \prod_{j=1}^{k-1} E_j^{i^j}$$

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

$$h_i = h \prod_{j=1}^{k-1} E_j^{i^j}$$

If the dealer follows the protocol, all honest agents will accept their shares, and $h_i = g^{s_i}$ will be publicly known for $i = 1, 2, \ldots, n$. 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 4.1
Any k participants, who have accepted their shoes correctly, can find $\log_g h$.

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

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

also satisfies $f'(0) = s$. As before let *proof* $= (E_1, E_2, \ldots, E_{k-1})$ and let e_j be defined by $E_j = g^{e_j}$ for $j = 1, \ldots, k-1$. Then

$$h \prod_{j=1}^{k-1} E_j^{i^j} = h_i = g^{s_i} = g^{f'(i)}$$

implies

$$s + \sum_{j=1}^{k-1} e_j i^j = f'(i) \bmod q$$

for $i = 1, 2, \ldots, k$. Now the uniqueness of f' implies that $f'(0) = s$. ■

As the secret is uniquely determined from the public key, this lemma immediately implies that

Theorem 4.2
The verifiable secret sharing scheme above satisfies requirement 1) and 3) of Definition 3.1 under assumption DLP.

For the scheme in Section 3.4 it was shown that the dealer reveals no Shannon information about the secret, but as previously mentioned such a claim makes no sense here, because each paticipant has all Shannon information about s to begin with. Instead we will show that if a probabilistic polynomial time algorithm can find s after it has been distributed, then s can also be found in probabilistic polynomial time given h. To be more precise we will show, that any number of participants can simulate the dealer perfectly no matter what shares they get (the reader is referred to [GMR89] for a precise definition of simulations and perfect simulations). Thus fewer than k participants do not get any information about s, that allows them to compute something, which they could not have computed before the secret was distributed.

Theorem 4.3
Any l participants having shares $(s_{i_j})_{j=1,\ldots,l}$ can find $(g^{f'_j})_{j=0,\ldots k-1}$, such that

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

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

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

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

Assume that $1 \leq l < k$ and (in order to simplify the notation) that the l agents in question are $P_1, \ldots, P_l$. As in the proof in [Fel87] they can generate f' as follows:

1. Choose $k-1-l$ random “shares” $s_{l+1}, \ldots, s_{k-1}$ corresponding to the public information $l+1, \ldots, k-1$.

2. Find $g^{f'_i}$ for $i = 0, \ldots, k-1$, where the polynomial $f'(x) = f'_0 + \cdots + f'_{k-1}x^{k-1}$ satisfies $f'(i) = s_i$ 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

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

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

$$\begin{pmatrix} 1 & 0 & 0^2 & \cdots & 0^{k-1} \\ 1 & 1 & 1^2 & \cdots & 1^{k-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & k-1 & (k-1)^2 & \cdots & (k-1)^{k-1} \end{pmatrix} \begin{pmatrix} f'_0 \\ f'_1 \\ \vdots \\ f'_{k-1} \end{pmatrix} = \begin{pmatrix} s_0 \\ s_1 \\ \vdots \\ s_{k-1} \end{pmatrix}$$

This $k \times k$ matrix is a Van der Monde matrix, and it has an inverse, A. 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 $P_1, \ldots, P_l$ can find each a_{ij}. Thus $g^{f'_i}$ can be computed for $i = 0, \ldots, k-1$ by the formula ($g^{s_0} = h$ is known)

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

This proves the theorem. ■

This theorem is a somewhat stronger statement than just saying that it is possible to generate messages with the same distributions as in PROTOCOL DISTRIBUTE, because the simulator generates a predetermined set of shares. This means that there is no particularly "bad" set of shares, which makes it easier for the shareholders to find the secret. However, the theorem does not exclude that certain broadcasted messages allow fewer than k shareholders to find the secret, but the dealer will choose such messages with very small probability.

The above simulation does not work, if a probabilistic polynomial time algorithm is allowed to choose the cheating participants depending on *proof* (a dynamic adversary), because when the simulator generates *proof*, it has already selected a set of $k-1$ participants for which it knows the shares. Thus the simulator cannot compute the shoe of a participant outside this set without knowing s. This problem is solved in [Fel87] by permuting the shares, but in the applications in this work, this complication is not necessary.

4.2 Linear Combinations

In Section 3.6 we saw that it was easy to compute the shoes and the broadcasted information of linear combinations of shared secrets. The scheme presented in this chapter also has this property. Namely, let $s', s'' \in \mathbb{Z}_q$ be two distributed secrets and let

s'_i	be P_i's secret share of s'
s''_i	be P_i's secret share of s''
$(E'_1, \ldots, E'_{k-1})$	be the broadcasted proof for s'
$(E''_1, \ldots, E''_{k-1})$	be the broadcasted proof for s''

Then P_i's secret share of $s' + s''$ mod q is $s'_i + s''_i$ mod q and the broadcasted information is the pairwise product of the tuples described above.

Similarly, P_i's secret share of $s = as'$ mod q for $a \in \mathbb{Z}_q^*$ is as'_i mod q and the broadcasted information $(E_1, E_2, \ldots, E_{k-1})$ is given by

$$E_i = E_i'^{a} \qquad i = 1, 2, \ldots, k-1.$$

It is easy to see that these values correspond to correct shares if the original shares are correct.

4.3 Selecting and Distributing a Secret Key

We now show how $P_1, P_2, \ldots, P_n$ can select a pair of keys (s, h) where $h = g^s$ such that for a fixed parameter k $(1 \leq 2k-1 \leq n)$, the secret key is distributed in a (k, n)-threshold scheme. It is not-hard to generalize the protocols to the general scheme presented in [Fel87] and such that $l \geq 2k-1$ members select the secret key and distribute it to the n members of the group.

Until now we have silently assumed that p, q and g was given a priori. However, if this is not the case, they can easily be selected by $P_1, \ldots, P_n$ and in this case the key generation consists of the following three phases:

1. Select the primes p and q, and the generator g of the subgroup G_q of $\mathbb{Z}_p^*$.
2. Select the secret key $x \in \mathbb{Z}_q$ and the corresponding public key $h = g^x$.
3. Distribute x.

Each of these steps will be described in detail in the following.

Generating p, q and g

Due to the fact that p, q and g are public information, the purpose of this step of the key generation is to make the participants agree on these numbers.

Let $BC(m, r)$ denote a commitment to $m \in \{0, 1\}^*$ using the random string, r. Any commitment scheme will do, but we suggest using the commitment scheme described in Section 3.4 as it is very efficient and is based on assumption DLP. The keys to this commitment scheme can be supplied by a trusted party (for example by the key authentication center which is needed anyway).

Let *GenPrimes* be a deterministic algorithm running in polynomial time, which on input a random seed and a security parameter with high probability generates primes p and q and a generator, g, of G_q. As mentioned in Section 2, *GenPrimes* can use a probabilistic primality test and start by selecting q (see also [Gor84] for a description of these ideas). When p and q are found, it is easy to find g. Using this algorithm, the members can find p, q and g as follows:

1. Using the commitment scheme all members simultaneously flip l coins to generate a common random seed.
2. Each member runs *GenPrimes* on input the seed and the security parameter, which is a common input to the participants.

A cheating participant might deliberately obtain wrong values of p, q and g, but this problem can be solved by either requiring that a majority of the participants are honest or that they all agree on these values.

Selecting s and h

The keys are selected as follows:

1. For $i = 1$ to n: P_i chooses $s_{i0} \in \mathbb{Z}_q$ at random (uniform distribution) and computes $\sigma_i = g^{s_{i0}}$. Then a random string r_i is chosen and $C_i = BC(\sigma_i, r_i)$ is broadcasted to all members.
2. When all n members have broadcast a commitment each P_i opens C_i starting with P_n, P_{n-1} and so forth.
3. The public key, h, is computed as $h = \prod_{i=1}^{n} \sigma_i$.

Now all members know the public key, but they cannot find the secret key $s = \sum_{i=1}^{n} s_{i0}$ unless they all work together (or some of them can compute discrete logarithms).

The use of the commitment scheme and the fact that the commitments are constructed and opened in opposite orders guarantee that, if P_i trusts his own coins then the distribution of the secret key is polynomially indistinguishable from the uniform distribution.

Distributing the Secret Key

It is assumed that P_i can send secret messages to each P_j $(1 \leq i, j \leq n)$ and that P_i can broadcast messages. Furthermore, P_i must be able to make digital signatures.

The idea in this phase is that each P_i first distributes his part of the secret key, s_{i0}, verifiably using PROTOCOL DISTRIBUTE. When all n members have done this, each P_i can compute a verifiable share of

$$s = s_{10} + s_{20} + \cdots + s_{n0}$$

using the properties described in Section 4.2 and the technique presented in Section 3.6.

We will require that P_i signs h as a proof that he accepts h as the public key of the group. When all members have signed h, a key authentication center verifies the signatures, and if they are correct, it makes a certificate showing that h is the public key of the group.

For each share s_i of s let h_i denote g^{s_i}, and let $(E_1, E_2, \dots, E_{k-1})$ denote the public information corresponding to s. Then each P_j can compute each h_i as

$$h_i = h \prod_{j=1}^{k-1} E_j^{ij}$$

It follows from Section 4.2 that after completion of the key distribution, any k honest members can find the secret key, whereas fewer than k members do not get any computational advantage compared to the situation where they are given h and $\sigma_1, \sigma_2, \dots, \sigma_n$, on such that $h = \prod_1^n \sigma_i$.

Note furthermore, that all members have to complete the protocols and finally sign the public key in order to gain any advantage, because otherwise the protocols are restarted and a new secret key selected. In future applications of this protocol it can therefore be assumed that all honest members get the expected number of messages.

4.4 Generalizations

The protocols presented in this and the previous chapter can be extended to compartmented schemes. Such a scheme is an extension of threshold schemes in which the participants are split into a number of subgroups $S_1, \dots, S_m$, and computation of the secret requires participation of k_i members of S_i.

Furthermore, as previously mentioned not all members of the group have to take part in the selection of the keys (as long as they trust someone who does participate).

Chapter 5

Threshold Crypto Systems

The concept of group oriented cryptography was introduced in [Des88] as a means of sending messages to a group of people, such that only certain subsets of the members are able to read the message (decipher the ciphertext). The rules for who is allowed to decipher depend on the positions of the members within the organization.

The members of a group are said to be *known* if the sender has to know them (a public key for each member), and a group is called *anonymous* if there is a single public key for the group independently of the members. In general, an anonymous group has a much shorter public key than groups with known members, but it is difficult to handle the situation where a member leaves the group, as this usually requires a new secret key to be selected. Desmedt presents crypto systems for both types of groups in the case where deciphering requires the cooperation of all members.

Group oriented cryptography has been further studied in [Fra90], [DF90] and [Hwa91]. Frankel used in [Fra90] the organization of individuals in groups to reduce the problem of distributing and managing public keys. His solution required clerks at the sending as well as the receiving organization. Here, any access structure can be obtained as the clerks at the receiving organization decide, who is allowed to read the message. However the clerks belonging to the same organization can, by cooperating, decrypt messages.

In [DF90], Desmedt and Frankel modified the El Gamal public-key crypto system (see [EG85]) so as to work for a group of n people. This scheme has the property that the group has a single public keys and the corresponding secret key is shared among the members such that any k of them can decipher, but fewer than k persons cannot (a *treshold* crypto system). This scheme requires a trusted person to choose a secret key for the organization and distribute it to the members.

In [Hwa91] a somewhat different approach is taken Here the El Gamal public-key crypto system and the Shamir secret sharing scheme are used to encrypt a key to a conventional crypto system, which is then used to encrypt the message. This scheme does not require a trusted person to distribute the secret key — each individual has its own secret keys but deciphering still requires the cooperation of k out of n individuals. This scheme has the advantages that the sender can choose k, but the members of the group are no longer anonymous — the sender has to know a public key of each member of the group.

Here we shall only consider anonymous groups. This chapter applies the secret sharing scheme proposed in Section 4.3 to the threshold crypto system in [DF90]. This results in a scheme with the following properties:

- A trusted party, who selects and distributes the secret key can be avoided, as the members of the group can select and distribute the key themselves.

- Each member of the group can verify that his share of the secret key is correct.

- A trusted party is required in order to verify the validity of recovered plaintext, but it does not need any secret keys.

- The threshold scheme is as secure as El Gamal's public-key crypto system (see [EG85]).

The second of these properties is very important as the shares are no longer computed by a trusted party, and each member should therefore convince himself that they are computed correctly.

The first two improvements are quite easy to obtain, when the schemes from Section 4.3 are given. In the following we first give a brief description of the EL Gamal public-key crypto system. Then the protocol for deciphering is presented, and finally it is proven that the scheme is as secure as the El Gamal scheme.

The material in this chapter has been published in [Ped91c].

5.1 The El Gamal Public-Key Crypto System

In [EG85], El Gamal proposed a public-key crypto system with public key (p, q, g, h) (see Chapter 2) and secret key $s = \log_g h$. A message $m \in G_q$ is enciphered as

$$c = (g^r, mh^r) \text{ where } r \in \mathbb{Z}_q \text{ is chosen at random.}$$

Someone knowing s can decipher $c = (c_1, c_2)$ as follows:

$$m = c_2 c_1^{-s}.$$

Assumption DH implies that it is hard to decrypt c given only c and the public key. However, due to the homomorphism property this crypto system is vulnerable to a chosen ciphertext attack and m should therefore have sufficient redundancy (or a message authentication code should be appended to m), such that valid messages can be recognized.

5.2 The Threshold System

The following also works if a trusted party has selected and distributed the secret key, but we only consider the situation where a group of $n \geq 2k-1$ people $P_1, P_2, \ldots, P_n$ have selected a public key (g, h) as described in Section 4.3. Thus the secret key $s = \log_g h$ is distributed among $P_1, P_2, \ldots, P_n$ in a (k, n) secret sharing scheme. P_i's secret share is denoted s_i and each P_i $(i = 1, 2, \ldots, n)$ knows all the values $h_i = g^{s_i}$ for $i = 1, 2, \ldots, n$.

Anybody knowing the public key can send a message $m \in G_q$ secretly to the organization by sending the ciphertext

$$(c_1, c_2) = (g^r, mh^r).$$

Any k members of the group, say $P_1, P_2, \ldots, P_k$, can later recover the plaintext as follows (see also [DF90]):

PROTOCOL DECIPHER

1. Each P_i finds $a_1, a_2, \ldots, a_k$ such that $s = \sum_{i=1}^{k} a_i s_i$.
2. Each P_i computes $C_i = c_1^{a_i s_i}$ and sends C_i secretly to a trusted center.

3. The trusted party computes $m' = c_2(\prod_{i=1}^{k} C_i)^{-1}$.
   If m' is a valid messages, the trusted party publishes m'.
   Otherwise the trusted party announces that the message is invalid.

The participation of the trusted center is necessary in order to prevent the presently known chosen ciphertexts attacks against the El Gamal system.

PROTOCOL DECIPHER makes it possible for a dishonest member (say P_j) to prevent (c_1, c_2) from being deciphered by sending $C_j \neq c_1^{a_j s_j}$. This will probably result in the trusted party rejecting the plaintext.

This problem can be avoided by requiring that each member proves that C_i has been computed correctly. This can for example be done using the efficient proof system for simultaneous discrete logarithms presented in [Cha91] (this protocol is also described in Section 7.1, PROTOCOL SDL). Without going into the details here, it should be mentioned that the trusted center can execute this protocol with $P_1, P_2, \ldots, P_k$ simultaneously and reuse some of the computations in each execution. This parallel execution remains zero-knowledge and is very efficient.

When this proof system is added to PROTOCOL DECIPHER, the previously mentioned fraud is impossible. Therefore the only way for a dishonest participant to prevent a plaintext from being recovered is by stopping the execution of the protocol.

5.3 Security of the Scheme

The following theorem shows that if a set, D (for dishonest), of fewer than k persons can find the share of a person outside D in a chosen ciphertext attack, then the secret key in the original El Gamal scheme can be found by a similar attack. By arguments similar to the proof of Theorem 5.1 it can be shown that any attack against the threshold scheme can be converted into a similar attack against the El Gamal scheme.

For ease of exposition we shall identify D with the set of indices of the persons in D. For any subset $D \subset \{1, \ldots, n\}$, let $wiew_D$ denote the messages that the parties in D saw in the key distribution protocol

$proof_i$;	for $i = 1, 2, \ldots, n$
s_{ij};	for $i = 1, 2, \ldots, n; j \in D$
r_j;	for $j \in D$

where $proof_i$ is the message which P_i broadcasts when distributing s_{i0}, s_{ij} is P_j's share of s_{i0}, and $r_j \in \{0,1\}^*$ is the random bits used by P_j. Furthermore, the participants in D received $(\sigma_1, \sigma_2, \ldots, \sigma_n)$ as input to this protocol.

Theorem 5.1
Let D be a set of l participants ($l < k, l + k \leq n$) such that the participants in $M = \{1, \ldots, n\} \setminus D$ have followed the key selection and key distribution protocol.

If for some $t \in M$ there is a probabilistic polynomial time algorithm, A,

$$A(p, q, g, (\sigma_i)_{i=1,\ldots,n}, view_D)$$

such that for some set of s_{ij} ($i = 1, 2, \ldots, n$, $j \in D$), A outputs $\log_g(h_t)$ with probability, π, then there is a probabilistic polynomial time algorithm, B, such that $B(p, q, g, H)$ equals $\log_g(H)$ with probability π. The first probability is over the distribution of $view_D$ (when the $(s_{ij})_{j \in D}$ are fixed) and the coins of A, whereas the probability of B's success is over the random choice of H and the coins of B.

In addition, A may ask an oracle for the decryption of ciphertexts, and in that case B is also allowed to get ciphertexts deciphered with respect to the public key H.

Proof
In the following p, q and g are fixed. For $m, h \in G_q$ and $y \in \mathbb{Z}_q$ let $E(m, y, h) = (g^y, mh^y)$ denote the enciphering function in the El Gus public-key crypt0 system.

Let the members in D be $P_1, P_2, \ldots, P_l$ and let $t = n$. On input (p, q, g, H), algorithm B consists of two parts:

1. Generate the input to A.
2. Simulate A.

The first part is done as follows:

1. Invent $n - 1 - l$ "players" $(P_{l+1}, \ldots, P_{n-1})$, who are going to be honest (follow the protocol) and a n'th member P_n who does not follow the protocol, but nonetheless his behaviour cannot be distinguished from that of an honest player.

2. Simulate the key selection scheme.
   (a) For $l < i < n$ select $s_{i0} \in \mathbb{Z}_q$ at random, compute $\sigma_i = g^{s_{i0}}$. Let $\sigma_n = H$.

       For $1 \leq i \leq l$, P_i selects σ_i as he would do in an execution of the key seliction protocol.
   (b) Compute the public key $h = \prod_{i=1}^{n} \sigma_i$.
3. Simulate the key distribution protocol.
   (a) For $l < i < n$ choose the polynomial f_i and compute E_{ij} for $j = 1, 2, \dots k-1$ such that the participants in D get the required shares.
   (b) For $i = n$ generate $proof_n$ as described in the proof of Theorem 4.3. In this step B also chooses shares of $s_{n0} = \log_g H$ corresponding to $P_{l+1}, P_{l+2}, \dots, P_{k-1}(s_{n,l+1}, s_{n,l+2}, \dots, s_{n,k-1})$.
   (c) For $1 \leq i \leq l$ distribute s_{i0} as P_i would have done.
4. If one of the participants does not accept his share goto 1.

Whenever H is chosen at random, the E_{ij}'s are uniformly distributed in G_q such that the participants in D get the correct shoes, and $(\sigma_i)_{i=1,\dots,n}$ have the same distribution as after an execution of the key selection scheme. Thus this part generates $view_D$ with the required distribution (the random bits used by P_j for $j \in D$ are easy to generate).

This means that we are able to supply the input to A with the correct distribution and we are therefore able to simulate A as described below.

Since there are at least k honest participants, and since each P_i $(i \in D)$ has executed the key distribution protocol properly each of them has sent at least k correct shares of $s_{i0} = \log_g(\sigma_i)$. From these shares it is possibly to find a polynomial f_i of degree at most $k-1$ such that $f_i(j)$ is P_j's correct share of s_{i0} for $i \in D$ and $j = 1, 2, \dots, n$. Thus B knows the following correct shares:

$$\begin{array}{ll} s_{ij} & \text{for } 1 \leq i < n;\ 1 \leq j \leq n; \\ s_{nj} & \text{for } 1 \leq j \leq k-1. \end{array}$$

Hence B knows $s_{i0} = \log_g \sigma_i$ for $i = 1, 2, \ldots, n-1$. Due to the fact that for $m \in G_q$ and $y \in \mathbb{Z}_q$

$$E(m, y, H) = (C_1, C_2) \Longleftrightarrow E(m, y, h) = (C_1, C_2 \prod_{i=1}^{n-1} C_1^{s_{i0}})$$

the participants can map ciphertexts in the scheme with public key H to the corresponding ciphertext in the scheme with h as public key and vice versa. Whenever A requests (c_1, c_2) to be deciphered, B does the following:

1. Compute the corresponding ciphertext $C = (C_1, C_2)$ in the system with public key H

$$C_1 = c_1 \text{ and } C_2 = c_2(\prod_{1}^{n-1} c_1^{-s_{i0}})$$

2. Ask C to be deciphered. Let m be the plaintext corresponding to C and return m as the decryption of (c_1, c_2).

As the input to A is constructed with the correct distribution, it will output s_n with probability π. But s_n is given as

$$s_n = \sum_{i=1}^{n} s_{in}$$

and since s_{in} is known for $1 \leq i < n$, this means that B can find s_{nn} and therefore B can find the following k shares of $s_{n0} = \log_g \sigma_n = \log_g H$:

$$s_{n1}, s_{n2}, \ldots, s_{n,k-1}, s_{nn}$$

Now, it is easy to find $\log_g H$ from these k shares. ■

In the proof of Theorem 5.1 we only considered the case where all n participants choose the secret key. If h is selected and distributed by a trusted party, who does not cooperate with the dishonest participants, the proof is even simpler as we do not have to consider $\sigma_1, \sigma_2, \ldots, \sigma_n$.

The above theorem shows that the threshold scheme is as secure as the El Gamal public-key crypto system. However, the scheme is also vulnerable to the same presently known chosen ciphertext attacks. Such attacks are avoided in the original system by having sufficient redundancy in the plaintext, and therefore it is necessary to have a trusted machine, that collects all the C_i's, finds the plaintext and verifies that thy plaintext is valid. Only these plaintexts are subsequently revealed to the members.

If the ideas presented above are used with a public-key scheme which is secure against chosen ciphertext attacks it seems that the trusted party in the deciphering protocol can be avoided. However, in that case one should by careful to avoid problems paused by members stopping in the middle of an execution of PROTOCOL DECIPHER.

Chapter 6

Distributed Proofs

In an interactive proof system for a language, L, (see [GMR89]) an arbitrarily powerful prover convinces a polynomial time bounded verifier that a given string, x, is in L. However, in practical protocols the prover has to be polynomially bounded too, and in this case the advantage of the prover is not his computational power, but rather the knowledge of a secret (see [BC86] and [Cha86]). Therefore we will here consider a polynomially bounded prover, who has secret key, and using this secret key he can convince the verifier of an assertion regarding the common input $x \in \{0,1\}^*$ (for example that $x \in L$ for some language L). Whenever there is a public key corresponding to the secret key, it will be considered a part of the common input. Sometimes we shall also call the secret key a *witness* of the assertion.

It is essential in many applications that only someone who has a witness (the prover) can prove the assertion, but in other situations this property constitutes a practical problem (see for example Chapter 9).

A *distributed proof* makes up for this shortcoming by allowing the prover to distribute his witness to n *agents* in a (k,n)-secret sharing scheme, such that any k of these agents later can play the role of the prover.

The agents can (of course) do this by first recovering the secret and subsequently one of them can execute the usual protocol instead of the prover. However, this would be against the intentions of the prover, because he does not want anybody else to know the witness.

The goal of a distributed proof is to convince the verifier of the assertion, but the agents are not allowed to obtain any more information about the secret. A distributed proof with this property will be called *secure*.

In this section we first describe the model and define distributed proofs formally, and then a (theoretical) distributed proof for any language in *NP* is constructed. Finally, we apply distributed proofs to an identification scheme showing how a number of agents can represent a single person.

To avoid confusion we mention here that distributed proofs are different from multi-prover proof systems (defined in [BGKW88]), as we restrict the agents to polynomial time and allow that (dishonest) agents send (secret) messages to each other.

6.1 Definition of Distributed Proofs

The verifier and each agent will be modeled by an interacting probabilistic polynomial Turing machine. This is a polynomially bounded Turing machine, which in addition to the work tape has the following tapes

- a tape with randomly chosen bits (read-only; decides the random choices of the machine);
- a tape for common input (read-only; shared by all users);
- a tape for auxiliary input (read-only; can only be read by the owner);
- a tape for sending messages (can be read by the other machines);
- a tape for receiving messages (read only).

All agents share the same tape for sending messages to and receiving messages from the verifier. Thus this model can be shortly characterized by the following properties:

- Broadcasting is possible;
- Messages from the agents are not authenticated;
- Secret communication is not provided.

Figure 1 depicts this model in the case of two agents.

The machines can do computations simultaneously, but only one machine at a time can write a message on one of the broadcast tapes, and

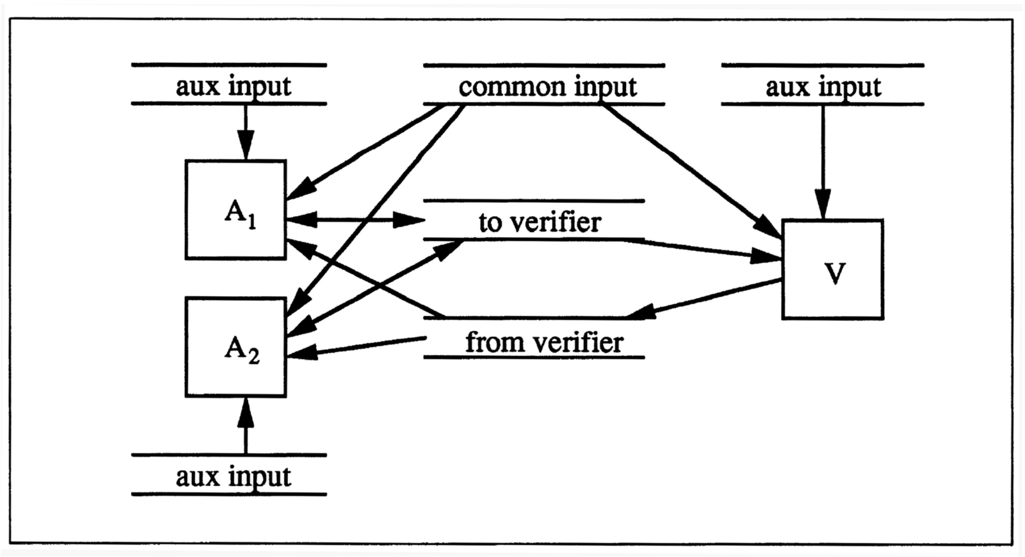

Figure 6.1: Model of two agents and one verifier. The work tapes and the random tapes are not shown.

when this machine has finished writing this message, all the other machines immediately copy it before continuing their computations.

As the model does not provide authentication of messages, we require that whenever an agent, P_i, sends a message, m, he actually writes (i, m) on the broadcast tape. This does not prove that P_i sent the message, but if all agents are honest, this must be sticient to convince the verifier. On the other hand, the lack of authentication shoed not enable dishonest agents to convince the verifier of a false claim.

An execution of a protocol between k agents $(A_1, A_2, \ldots, A_k)$ and a verifier, V, on common input x is denoted

$$[A_1(z_1), A(z_2), \ldots, A_k(z_k), V(y)](x)$$

where z_i is the auxiliary input of A_i and y is the auxiliary input of the verifier. If a participant has no auxiliary input, this argument is just omitted.

Definition 6.1
A participant (agent or verifier) is *honest* if it follows the protocol in all executions. A participant, who is not honest, is called *dishonest*.

Hence, an honest agent will never use his share of the witness to anything but executing the protocols properly. In particular, an honest agent will never help anybody finding the witness.

Dishonest agents and verifiers are modeled as described above except that any pair of dishonest participants may share a tape such that they can send secret and authenticated messages to each other.

Even though the origins prover distributes his secret (witness) among n agents, he does not want that anybody (not even an agent) ever finds it. Therefore he must trust that no set of at least k agents will ever cooperate in order to find the secret, Hence, we will always assume that at most $k-1$ of the agents are dishonest (this is a sticient, but not a necessary condition for the prover).

Furthermore, we will assume that the set of dishonest agents is selected when the protocol is initiated (a *static* model as opposed to a *dynamic* model in which the set of dishonest agents can be extended after each step in the protocol depending on the execution). Remember, however, that a dishonest agent can perfectly well follow the prescribed protocol most of the time and only deviate from it in certain situations.

Now consider a polynomial time predicate,$R(w,x)$, and let L be the language of strings $x \in \{0,1\}^*$, such that there exist a secret key, $w \in \{0,1\}^*$, satisfying $R(w,x)$. Thus

$$L = \{x \in \{0,1\}^* \mid \exists w \in \{0,1\}^* : R(w,x)\}.$$

As in Chapter 3 a secret sharing scheme is given by two functions, *distribute* and *combine*, such that if

$$\textit{distribute}(r,w) = (w_1, w_2, \dots, w_n)$$

then P_i gets the secret share w_i of w ($r \in \{0,1\}^*$ is a random string).

In the following definitions, "non-negligible" and "overwhelming" are always as functions of $|x|$.

Definition 6.2
Let R be a polynomial time computable predicate, and let a secret sharing scheme be given by (*distribute*, *combine*).
$(n+1)$ interacting probabilistic polynomial Turing machines $P_1, \dots, P_n, V$

are a (k, n)-distributed proof for membership in the language, L, with respect to (*distribute*, *combine*), if the following two requirements are satisfied

- *Completeness*: If $x \in L$ and w is a witness of this (i.e. $R(w, x)$), and if *distribute*$(r, w) = (w_1, w_2, \ldots, w_n)$ then for any k (honest) agents $P_{i_1}, \ldots, P_{i_k}$, the execution of

  $$[P_{i_1}(w_{i_1}), \ldots, P_{i_k}(w_{i_k}), V](x)$$

  results in V accepting with overwhelming probability.

- *Soundness*: If $x \notin L$ then for any k interacting Turing machines $A_1, \ldots, A_k$ and for any auxiliary inputs z_i to A_i the execution of

  $$[A_1(z_1), \ldots, A_k(z_k), V](x)$$

  results in V rejecting with overwhelming probability.

Both probabilities are over the random choices of the agents and the verifier. The protocol is called *unconditionally secure* for the verifier if the soundness condition is fulfilled for all powerful agents.

In the following a (k, n)-distributed proof is just called a distributed proof as the parameters k and n are clear from the context.

This definition states that a distributed proof is a proof system. It is also possible to define a distributed proof, which is a proof of knowledge. Intuitively this means, that the agents prove that together they possess sufficient information to find the secret key. By a slight modification of the definition of proofs of knowledge in [FFS88] (alternatively we could have modified the definition in [TW87]) we obtain

Definition 6.3
Let R be a polynomial time computable predicate, and let a secret sharing scheme be given by (*distribute*, *combine*).
$(n + 1)$ interacting probabilistic polynomial Turing machines $P_1, \ldots, P_n, V$ constitute a (k, n)-distributed proof of knowledge for R with respect to (*distribute*, *combine*) if the following two requirements are satisfied

- *Completeness*: If $R(w, x)$ and $\textit{distribute}(r, w) = (w_1, w_2, \ldots, w_n)$, then for any k agents $P_{i_1}, \ldots, P_{i_k}$, the execution of
$$[P_{i_1}(w_{i_1}), \ldots, P_{i_k}(w_{i_k}), V](x)$$
results in V accepting with overwhelming probability.
- *Soundness*: There is probabilistic Turing machine, M, with control over the agents such that if any k agents $A_1, A_2, \ldots, A_k$ can make V accept with non-negligible probability on common input x, then
$$M_{A_1,\ldots,A_k}(z_1, z_2, \ldots, z_k, x)$$
outputs a $w \in \{0,1\}^*$ satisfying $R(w, x)$ with overwhelming probability. Here z_i is the auxiliary input of A_i for $i = 1, 2, \ldots, k$.

The machines, M, above will be called a *knowledge extractor.*

The reader is referred to [FFS88] for a precise definition of what it means that $M_{A_1,A_2,\ldots,A_k}$ has control over $A_1, A_2, \ldots, A_k$.

When defining the security of distributed proofs we will require that the entire process consisting of the distribution of the witness followed by several executions of the distributed protocol be secure, as this is a natural requirement for the original prover. Furtherrnore, this approach makes it easier to exploit properties of the secret sharing scheme when proving that the distributed protocol is secure.

Each execution of the protocol may involve different sets of agents and different verifiers. However, due to the fact that 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 she may behave differently in each execution). This automatically handles the situation where different verifiers cooperate. For any verifier and any set of dishonest agents consider the following protocol:

1. The original prover distributes w among $P_1, P_2, \ldots, P_n$.
2. Repeat a polynomial number of times: The verifier and the dishonest agents select an input $x \in \{0,1\}^*$ to the distributed protocol and a set of agents with whom the protocol is executed.

It can be hard in practice for the dishonest participants to select inputs in L, but in that case x can be supplied by either an oracle or the original prover.

As a starting point we will call a distributed proof secure, if the above protocol is zero-knowledge. However, in order to obtain a definition, which is easier to use, we will require the existence of two simulators. The first simulator simulates the distribution of w and outputs some extra information, denoted *distinf* (for "distribution information") about the witness that was distributed. This extra output may be used by the second simulator, which simulates a subsequent execution of the proof in step 2 above.

Following the ideas of [CDG88], where the security of general multiparty protocols is defined, we therefore define

Definition 6.4
A distributed proof is *secure*, if for every set of dishonest agents, D ($D \subset \{1, 2, \ldots, n\}, \#D < k$), and every verifier, V^*, for every set of auxiliary inputs to the verifier and the dishonest agents, there exists two probabilistic polynomial time machines, M_D^{dist} and M_{D,V^*}^{proof} which satisfies:

1. $M_D^{dist}((w_i)_{i \in D})$ outputs a pair $(conv, distinf)$ such that $conv$ is indistinguishable from the messages which $(P_i)_{i \in D}$ receive when w is distributed subject to the constraint that P_i gets the share w_i for $i \in D$.

2. $M_{D,V^*}^{proof}(x, (z_i)_{i \in D}, aux, distinf)$ outputs a conversation which is indistinguishable from an execution of a proof in step 2 above, when the common input is x and the agents outside D are honest. Here z_i is the auxiliary input of P_i for $i \in D$ and aux is the auxiliary input of V^*.

The protocol is called *strongly secure*, if it is secure even wnen (some of) the executions in step 2 above are executed in parallel.

If P_i only receives a share of w it is very easy to construct M_D^{dist} as

$$M_D^{dist}((w_i)_{i \in D}) = (w_i)_{i \in D}.$$

However, as argued earlier it is reasonable that the agents require that w be distributed verifiably. In particular, we shall consider non-interactive verification, and in this case M_D^{dist} has to construct the broadcasted messages with the correct distribution.

Before proceeding Definition 6.4 deserves a few remarks.

- It allows that an execution of the distributed protocol reveals some information about the shares of the honest agents, as long as this information can be computed from the information, which the original prover broadcasted in the secret sharing scheme.
- It is implicitly assumed that all dishonest participants cooperate as they can be accessed by the simulator. This need not be the case in practice, but if the dishonest agents cannot obtain new information about the secret key when cooperating, they cannot learn anything either if they do not cooperate.
- It is not required that the agents identify themselves before executing the protocol. If someone manages to replace an agent without having a share of the secret key, this person is considered dishonest and clearly this person cannot reveal anything about the secret key.
- A dishonest agent who stops in the middle of the execution of the protocol may have an advantage in knowing that x is in L. But this is not a problem here as we are only considering the security when the common input is in L. Thus it is the responsibility of the honest agents that $x \in L$.

The above definitions do not say anything about how an honest agent can be certain that x is a legal input. This problem is not important for the definition of distributed proofs, but it must be solved in applications of distributed proofs.

6.2 A Theoretical Solution

Let R be a predicate such that the prover P, who has a secret key, w, can prove that the common input, x, is in L using the m-round protocol shown in figure 2. Such a zero-knowledge proof system exists for every language in NP, if probabilistic encryption exist (see [GMW86]).

The functions v_i and p_i in figure 2 can be computed in polynomial time illustrating that for $i = 1, \ldots, m$, the i'th message is computed in polynomial time from some random coins (r_P and r_V), the input (x), some auxiliary input (w and aux_V) and the messages received in the previous rounds.

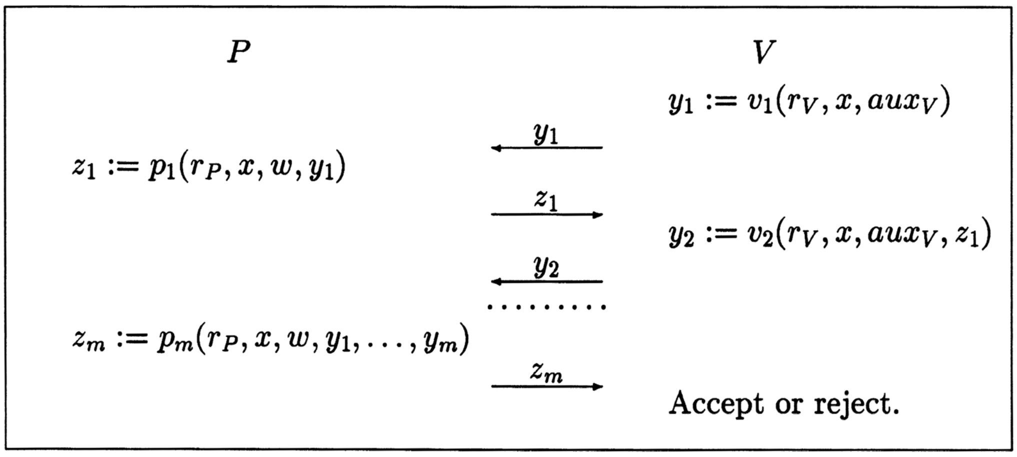
P
V
$y_1 := v_1(r_V, x, aux_V)$
y_1
$z_1 := p_1(r_P, x, w, y_1)$
z_1
$y_2 := v_2(r_V, x, aux_V, z_1)$
y_2
....
$z_m := p_m(r_P, x, w, y_1, \ldots, y_m)$
z_m
Accept or reject.

Figure 6.2: A general protocol.

Theorem 6.5
Under the Quadratic Residuosity Assumption[1] and assumption DLP the following holds.
If the protocol in figure 2 is a zero-knowledge proof of membership in L, it can be turned into a secure distributed proof of membership.
If the protocol in figure 2 is a proof of knowledge, it can be turned into a secure distributed proof of knowledge.

Proof sketch
The distributed proof works as follows:

1. The prover distributes w to $P_1, P_2, \ldots, P_n$ using the secure verifiable secret sharing scheme described in Section 3.4.
2. Whenever k agents, say $P_1, P_2, \ldots, P_k$, are eked to execute the protocol on common input x, they simulate the original proof system. In the i'th round ($i = 1, 2, \ldots, m$) they compute the prover's message as follows:
   (a) Each P_i supplies his share w_i of the witness and all agents verify that the share is correct using the predicate *verify*.

[1]This assumption says that for a Blum integer, n, and $x \in \mathbb{Z}_n^*$ with Jacobi-symbol 1 it is not feasible to decide whether x is a quadratic residue modulo n or not (see [Dam88] for a precise definition).

If P_i discovers that P_j has supplied an incorrect share, he stops the prototol.

(b) Compute $w' = combine[(1, w_1), (2, w_2), \ldots, (k, w_k)]$.

(c) Compute $z_i = p_i(\mathrm{r}, x, w', y_1, \ldots, y_i)$.

Each P_i also supplies a number of random bits. The XOR of the random bits from each participant is used as the input r.

These computations are done using the secure protocol for multiparty computations described in [CDG88] such that the intermediate results are kept secret.

The properties of the protocol from [CDG88] imply that this protocol is a proof system. We now prove that it is (auxiliary input) zero-knowledge. Due to the fast that the agents can simulate the distribution of w by choosing random elements (as stated in Theorem 3.6), this implies that the protocol is secure affording to Definition 6.4 ($distinf$ is the empty string in this case).

Consider an execution of the protocol between k agents and the verifier V^* and assume that the agents $P_1, P_2, \ldots, P_l$ are dishonest, whereas $P_{l+1}, \ldots, P_k$ are honest ($0 \leq l < k$). The simulator, M^{proof}, works as follows:

1. Simulate the original proof system.

   This results in a sequence of messages $(y_1, z_1, y_2, z_2, \ldots, y_m, z_m)$ plus the random bits used by V^*.

2. For $i = 1, \ldots, m$ simulate the multiparty computation which given $(x, aux_1, \ldots, aux_l, w_{l+1}, \ldots, w_k, y_1, y_2, \ldots, y_i)$ outputs z_i. Here aux_i is the auxiliary input of P_i for $1 \leq i \leq l$ (this is not necessarily w_i).

If the original protocol is statistical zero-knowledge then each z_i equals $p_i(r, x, w, y_1, \ldots, y_i)$ with overwhelming probability. In this case the out-put of the simulation in step 2 above cannot be distinguished from a real computation under the Quadratic Residuosity Assumption. Hence, the entire simulation cannot be distinguished from an execution of the protocol.

However, if the origins protocol is only computational zero-knowledge we are only assured that z_i produced in step 1 cannot be distinguished from $p_i(r, x, w, y_1, \ldots, y_i)$. However, an analysis of the proof in [CDG88] shows that a simulation of the multiparty computation which outputs z_i cannot

cannot be distinguished from a computation of $p_i(r, x, w, y_1, \ldots, y_i)$ unless z_i can be distinguished from $p_i(r, x, w, y_1, \ldots, y_i)$.

If the original protocol is a proof of knowledge, the distributed proof is also a proof of knowledge, because the knowledge extractor from the original protocol can be used in the distributed proof as well. ■

The multiparty protocol from [CDG88] depends on the Quadratic Residuosity Assumption, but if this protocol is replaced by the protocol from [GHY88], which can be implemented depending on assumption DLP, this extra assumption can be avoided. This will require a minor change in the proof as the definition of secure protocols in [GHY88] is slightly different from that in [CDG88].

In [CDG88] it is required that less than half the participants in the multiparty computation are honest. This requirement is only needed in order to handle situations where a participant stops in the middle of the computations The only advantages such a participant obtains in our protocol, is that he might know the result of the computation whereas the other agents do not. However, this is not an advantage in this application, because the original protocol is zero-knowledge.

In the proof of Theorem 6.5 we did not need an extra output from the simulator of the distribution $(distinf)$. However, in Chapter 9 this extra output will be very important.

6.3 Applications to Identification Schemes

Schnorr presented in [Sch90] an identification scheme in which a prover with a public key (p, q, g, h) and a secret key, $s = \log_g h$, identifies himself by proving that he knows $\log_g h$. In this section it is shown how to turn this protocol into a distributed proof of knowledge. This construction allows the owner of s to authorize n agents such that any k of these can represent him, whereas fewer than k cannot.

The Original Scheme

The basic identification protocol from [Sch90] with security parameters t and τ works as follows (P is the prover and V the verifier):

1. Repeat the following τ times:
   (a) P chooses $r \in \mathbb{Z}_q^*$ at random and computes $g_1 = g^r$.
   P sends g_1 to V.
   (b) V chooses $e \in \{0, 1, \dots, 2^t - 1\}$ at random and sends e to P.
   (c) P computes $y = r - se \bmod q$ and sends y to V.
   (d) V verifies that $g_1 = g^y h^e$.
2. V accepts if and only if all τ verifications in step (d) above are satisfied.

For $t = O(\log|q|)$ and $\tau = \Theta(|q|)$ this is a perfect zero-knowledge proof of knowledge (of $\log_g h$).

Schnorr proposes to use $t = 72$ and $\tau = 1$ for practical purposes. In this case the probability of cheating is 2^{-72} but it is not known if the protocol remains secure after many execution; (the "obvious" simulator has to make 2^{71} trials in order to construct a correct conversation). Therefore we will only consider the general scheme described above and turn that protocol into a distributed proof.

Distributed Identification

Due to the fact that the secret key is "known" as $\log_g h$, the prover can distribute s to $P_1, P_2, \dots, P_n$ with the secret sharing scheme presented in Section 4.1. Thus each agent, P_i, has a secret share, s_i of s and $h_i = g^{s_i}$ is publicly known. For the moment we will therefore assume that after the distribution of s, the public key of P is $(g, h, h_1, h_2, \dots, h_n)$ (omitting p and q).

Any k agents (say $P_1, P_2, \dots, P_n$) can represent P as follows (the public key is the common input and P_i has s_i as auxiliary input):

1. Each P_i sends i to the verifier.
2. The verifier computes $a_1, a_2, \dots, a_k$ such that $h = \prod_1^k h_i^{a_i}$. Hence
$$a_i = \prod_{j \neq i} \frac{j}{j - i}.$$
3. Repeat the following τ times:

(a) Each P_i chooses $r_i \in \mathbb{Z}_q^*$ at random and computes $g_{i1} = g^{r_i}$. P_i sends g_{i1} to V.

(b) V chooses $e \in \{0, 1, \ldots, 2^t - 1\}$ at random and broadcasts e to each P_i.

(c) P_i computes $y_i = r_i - s_i e \bmod q$ and sends y_i to V.

(d) V first computes $g_1 = \prod_1^k g_{i1}^{a_i}$ and $y = \sum_1^k a_i y_i$.
Then V verifies that $g_1 = g^y h^e$.

4. V accepts if and only if all τ verifications in step (d) above are satisfied.

Theorem 6.6
The above protocol is a secure distributed proof of knowledge of discrete logarithms with respect to the verifiable secret sharing scheme in Section 4.1.

Proof sketch
A simple modification of the proof in [Sch90] shows that the protocol is a distributed proof of knowledge. Furthermore, for any verifier V^* and any set of at most $k - 1$ dishonest agents, the honest agents can be simulated using the simulator of the original protocol for each honest agent. This and the fact that the distribution of s can be simulated perfectly (see Theorem 4.3) immediately implies that the protocol is secure. ■

This distributed identification scheme makes it possible for the verifier to decide which agents represent P. This can be avoided if the agents are allowed to broadcast messages to each other, which the verifier cannot see. This kind of broadcast should be used in step 1, (3a) and (3c) above, after which step 2 becomes unnecessary. Then each agent can compute $g_1 = \prod g_{i1}^{a_i}$ and send g_1 to the verifier in step (3a), and after receiving e and all the y_i's each agent sends $y = \sum_{i=1}^k y_i a_i$ to the verifier. The verifier should stop the execution if not all the g_1-values or all the y_1-values are equal. Using this variant the public key is still (g, h) after the distribution, and each P_i gets $(h_1, \ldots, h_n)$ as auxiliary input.

Each agent knows which agents participate in the identification protocol, but even if $k - 1$ agents cooperates, they cannot afterwards prove to anybody else who the k'th agent was.

Distributed Signatures

The original identification scheme can be turned into a signature scheme by setting τ equal to 1, choosing t as $\Theta(|q|)$ and computing e as a "pseudorandom hash value" of the message and g_1 (see [Sch90]). Thus the signature on the message $m \in \{0,1\}^*$ is a pair (g_1, y) satisfying

$$g_1 = g^y h^e$$

where $e = H(m, g_1)$ and H is a "suitable" hash-function (see for example [FS87] and [Sch90]).

By doing this with the distributed identification scheme, a signature scheme is obtained in which any k agents can sign messages on behalf of P. Let $A \subset \{1, 2, \ldots, n\}$ such that $\#A = k$. If the first distributed identification scheme is used, the agents $(P_i)_{i \in A}$ can construct a signature $(A, (g_{i1})_{i \in A}, (y_i)_{i \in A})$ on m which satisfies

$$\prod_{i \in A} g_{i1}^{a_i} = g^{\sum a_i y_i} h^e$$

where $e = H(m, (g_{i1})_{i \in A})$. This signature clearly shows which agents have signed the message for P.

If the second distributed identification scheme is used, the signature on m is a pair (g_1, y) where

$$g_1 = g^y h^e$$

and $e = H(m, g_1)$. This signature contains no information at all about the identity of the agents, but each agent sees a signature on the form

$$(A, (g_{i1})_{i \in A}, (y_i)_{i \in A}).$$

Hence, each agent in A knows and can prove who participated in the construction of the signature.

Since both signature schemes are obtained by parallelizing a zero-knowledge identification scheme, they seem to be secure against adaptive chosen message attacks (see [GMR88]).

Organizations

By combining the key selection scheme presented in Section 4.3 with the schemes presented above, one can construct a public key of an organization, such that nobody knows the corresponding secret key, but any k members can construct correct signatures with respect to this public key.

Chapter 7

Undeniable Signatures

Undeniable signatures were introduced in [CA90]. Briefly, an undeniable signature is a signature which cannot be verified without the help of the signer (see [CA90] 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.

In this section we first briefly present the undeniable signature scheme proposed by Chaum in [Cha91], and in Section 7.2 and 7.3 a few extensions (suggested in [CBDP91]) to this signature scheme are described.

7.1 An Undeniable Signature Scheme

The public key in the undeniable signature scheme is

$$KP = (p, q, g, h)$$

where $h \in G_q$, and the secret key, KS, is $x = \log_g h$. The signature on a message $m \in G_q$ is

$$z = m^x.$$

Assumption SDL, says that for a random message, m, there is a negligible probability that a polynomially bounded verifier can tell if a randomly chosen z is the signature on m or not. This assumption also implies that it is infeasible to forge a signature on a given message, m (except on a small number of messages). Namely, any method for obtaining a signature on m can be used to decide if a given z is the signature on m, by first finding the forged signature z' on m and then observing whether z equals z'.

However, an existential forgery (see [GMR88]) is possible by choosing $r \in \mathbb{Z}_q^*$ at random and computing

$$m = g^r \quad \text{and} \quad z = h^r.$$

Furthermore, if it is known that z_i is a signature on a message m_i for $i = 1, 2$ it can be concluded that $z_1 z_2$ is a signature on the message $m_1 m_2$.

The signature scheme should therefore not be applied to a message directly, but rather to the hash value of the message. The one-way hash function H, should at least have the properties that:

1. It should be impossible to find m and r such that $H(m)$ equals g^r (because of the existential forgery mentioned above).

2. H should spoil the homomorphism property.

The use of H also has the advantage that long messages can be signed efficiently, if $H(m)$ is easy to compute. For ease of notation the hash function is omitted in the following, where it is shown how the signer can prove whether a given message-signature pair is correct or not; the message m is implicitly assumed to be the reset of applying H to the original message.

Verifying Signatures

The signer, who knows x, can prove that z is indeed us signature on m by proving that $\log_g h$ equals $\log_m z$ as follows

PROTOCOL SDL

1. The verifier chooses $a, b \in \mathbb{Z}$ at random and computes the challenge $ch = m^a g^b$, which is sent to the signer.

2. The signer chooses $t \in \mathbb{Z}_q^*$ at random and computes $h_1 = ch^t$ and $h_2 = h_1^x$.
   The pair (h_1, h_2) is sent to the verifier.

3. The verifier sends the pair (a, b) to the signer.

4. If $ch = m^a g^b$ the signer sends t to the verifier, and otherwise he stops the execution.

5. The verifier accepts the proof if $t \neq 0 \bmod q$ and

$$h_1 = (m^a g^b)^t \text{ and } h_2 = (z^a h^b)^t.$$

Otherwise she rejects.

This protocol is slightly different from the one proposed in [Cha91]. As this protocol will be used in other contexts later we present a proof of the following theorem (the proof of the first three claims is from [BCDP91], and the last property is proven in [Ped91a]).

Theorem 7.1
PROTOCOL SDL satisfies

1. If z is a correct signature on m, the verifier accepts.

2. If z is not a correct signature on m, the verifier accepts with probability at most $1/q$ — even if the signer has unlimited computing power.

3. If z is a correct signature on m, the protocol is perfect zero-knowledge.

4. For any probabilistic polynomial time verifier V* there is a probabilistic machine M_{V^*} running in expected polynomial time on all inputs (p, q, g, h, m, z), such that if $\log_g h = \log_m z$, the output of M_{V^*} is statistically indistinguishable from a transcript of an execution of the protocol.

Proof
The first part is obvious. As for the second, observe that, if the signer is able to send correct answers corresponding to two different pairs (a, b) and (a', b') (where $a \neq a'$), then $\log_m z$ equals $\log_g h$. This follows from the fact that since $t \neq 0 \bmod q$ and $a - a' \neq 0 \bmod q$,

$$(m^a, g^b)^t = (m^{a'} g^{b'})^t \text{ and } (z^a h^b)^t = (z^{a'} h^{b'})^t$$

implies that

$$m = g^{(b'-b)(a-a')^{-1}} \quad \text{and} \quad z = h^{(b'-b)(a-a')^{-1}}.$$

In other words

$$\log_g m = \log_h z = \frac{b'-b}{a-a'} \neq 0$$

and thus

$$\log_g h = \log_m z.$$

Therefore, a cheating signer's probability of success is no better than the probability of finding a. But, for each value of $a \in \{0, \ldots, q-1\}$, there is exactly one value of b giving the same challenger Thus, ch contains no information about the value of a, and even a signer with unlimited computing power can therefore do no better than trying to guess a.

The third part of the theorem is proven by describing a simulator for the proof system. Let V^* be any probabilistic polynomial time verifier and consider an execution of the protocol between P and V^*. This execution will be simulated as follows:

1. Get a challenge, ch, from V^*.
2. Choose e and compute $h'_1 = g^e$ and $h'_2 = h^e$.
3. Get (a, b) from V^*.
   If $ch \neq m^a g^b$: stop;
4. Rewind V^* to after the challenge is sent.
   Choose t and compute $h_1 = (m^a g^b)^t$ and $h_2 = (z^a h^b)^t$.
5. Get (a', b') from V^*.
   If $ch = m^{a'} g^{b'}$: send t to the verifier and stop;
   If $ch \neq m^{a'} g^{b'}$: goto 4;

It is not hard to see that the simulator constructs a conversation having the same distribution as conversations of real executions, if it stops in step

3. Furthermore, if the machine stops in step 5, the constructed values $ch, h_1, h_2, (a', b')$ and t satisfy

$$\begin{aligned} ch &= m^{a'} g^{b'} \\ h_1 &= ch^t \\ h_2 &= h_1^x \end{aligned}$$

where a', b' and t are chosen with the same distribution as in real executions. As usual the simulator can easily be modified so that it also outputs the random bits used by V^* with the correct distribution.

Due to thy fact that thy simulator always produces pairs (h_1, h_2) with the same distribution as the real signers the simulator runs in expected polynomial time.

The above simulator is not sufficient to prove the last assertion because $h_2 \neq h_1^x$ in step 4, if $z \neq m^x$. Even though V^* can only discover this with very small probability, it may be sufficient to spoil the expected polynomial running time.

In order to solve this problem we insert a stopping procedure and replace step 4 and 5 in the above simulator by the following

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 $h_1 = ch^t$ and $h_2 = (z^a h^b)^t$.
   (c) Get (a', b') from the verifier.
   If $ch = m^{a'} g^{b'}$: send t to the verifier and stop;
   If $ch \neq m^{a'} g^{b'}$: goto (a);

   Procedure B:
   (a) $count := 0$.
   (b) Rewind V^* to after ch was sent.
   (c) Compute $h_1 = g^e$ and $h_2 = h^e$, where $e \in \mathbb{Z}_q$ is chosen at random.
   (d) Get (a', b') from the verifier.

(e) $ch = m^{a'} g^{b'}$: count := count + 1.

(f) If *count* $< |q|$: goto (b);
Otherwise: stop.

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

To show that M_{V^*} runs in expected polynomial time on all inputs 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 $\log_g h = \log_m z$. It follows from the proof of property 3 that if M_{V^*} stops in step 3, or because procedure A stops before procedure B, then the produced 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 $ch = m^{a'} g^{b'}$. 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 $2^{-|q|}$. ■

The fourth property in Theorem 7.1 is not important for the application to undeniable signatures, but it will be essential later,

Denying Signatures

Given a false message-signature pair, (m, z), the signer (knowing $x = \log_g h$) can prove that z is not a signature on m by proving that $\log_g h$ is not equal to $\log_m z$. A zero-knowledge protocol for this is presented in [Cha91]. For completeness this protocol is described below ("DDL" abbreviates different discrete logarithms). Let as usual $BC(m, r)$ denote a commitment to the message $m \in \{0,1\}^*$ using the random string $r \in \{0,1\}^*$, and let $k, t \in \mathbb{N}$ be security parameters:

PROTOCOL DDL satisfies

1. The signer first computes $(m^x/z)^i$ for $i = 1, \dots, k$ and stores the results in a table.

   Repeat the following t times:

   (a) The verifier chooses $s \in \{0, \dots, k\}$ and $a \in \mathbb{Z}_q$ at random and computes $ch_1 = m^s g^a$ and $ch_2 = z^s h^a$.
   The pair (ch_1, ch_2) is sent to the signer.

   (b) Using the table created above the signer finds s' such that $(m^x/z)^{s'} = ch_1^x/ch_2$.
   If no such s' is found, the signer chooses s' at random such that $0 \leq s' \leq k$.
   The sigher chooses $r \in \{0,1\}^*$ at random and sends $BC(s', r)$ to the verifier.

   (c) The verifier remembers the commitment and sends a to the signer.

(d) If $ch_1 = m^{s'}g^a$ and $ch_2 = z^{s'}h^a$, then the signer opens the commitment, and otherwise he stops the protocol.

(e) The verifier checks that the commitment received in step (b) contains s. If this is the case she accepts the proof, and otherwise she rejects.

2. The verifier accepts the proof if and only if she accepts in all t iterations.

In [CBDP91] it is proven that

Theorem 7.2
PROTOCOL DDL satisfies

1. If $\log_m z \neq \log_g h$, then V accepts with probability 1.
2. If $\log_m z = \log_g h$, then no matter what an unlimited signer does, V accepts with probability at most $(\rho + \frac{1}{k+1})^{-t}$, where ρ is the probability with which the signer can open a commitment to reveal more than one value of s.
3. The protocol is perfectly or statistically (computationally) zero-knowledge if the commitment scheme is unconditionally (computationally) secure for the signer.

This theorem is also true when the t iterations are executed in parallel.

7.2 Hidden Verifiers

The advantage of undeniable signatures over digital signatures is, that a person should not be able to verify the validity of a signature without the knowledge of the signer. In this section we first show how the verifier, by computing his messages in a special way, can convince a group of people of the outcome of the execution, and then it is demonstrated how the signer can prevent this fraud. A person who executes a protocol with the signer will be called a *known verifier*, whereas a person who tries to learn whether a signature is correct or not without the knowledge of the signer will be called a *hidden verifier*. Since a person who trusts the verifier, always believes what the verifier tells him, we will only consider hidden verifiers who do not trust the known verifier. As hidden verifiers constitute a problem for any (zero-knowledge) protocol where non-transitivity is of importance, the proposed countermeasure has many applications.

Now assume that there is a group of people of which only one (V) can execute the verification or denial protocol with the signer. All members of the group want to learn if a given signature is correct, but they do not trust V (and perhaps not each other). Assume further that V has agreed (gets paid) to help the other members to learn if the signature is correct.

It will be assumed that each hidden verifier can listen to the channel that the signer and V use for exchanging messages during the execution of the protocol (otherwise they have no guarantee that V does not simulate an execution of the protocol alone). However, the signer and V may share another channel so that they can send messages to each other without the knowledge of the hidden verifiers.

Consider PROTOCOL SDL and note that if V knows a and b before the signer has sent h_1 and h_2, then the hidden verifiers have no reason to trust the proof, as V could have sent a and b to the signer. Each hidden verifier should therefore require that the initial challenge be computed such that the corresponding pair (a, b) cannot be found before h_1 and h_2 has been received.

Due to this observation it may be assumed that the initial challenge is computed in a multi-party computation where the i'th verifier has a random string as input and gets (ch, s_i) as secret output. Later on, a and b such that $ch = m^a g^b$ can be found from all the s_i's. This way the hidden verifiers will be convinced of the conversation between the signer and V.

The signer can avoid this problem by requiring that a and b are computed as $f_k(s)$, where s is a string of random coins chosen by the verifier, and k is a key to a member of a family of functions, which the signer sends initially. The function f_k should satisfy:

- For $l \geq 2$ and for any function g, any multiparty computation of $f_k(g(s_1, \ldots s_l))$ by l parties where s_i is a secret input of one of the parties must take time at least $2T$ whereas a normal computation of $f_k(s)$ can be done within time T. The function g expresses that the verifiers can combine their secret inputs as they want to.

- All possible values for a and b have equal probability when the input is chosen at random.

If the signer requires that the challenge be sent within the time limit $\frac{3}{2}T$, he is assured that the challenge is not computed using a multiparty computation, and as a result only the person who has computed the challenge will be convinced of the proof.

A similar technique can be used in the denial protocol from [Cha91] as it also has a structure in which the verifier initially sends a random challenge.

7.3 Non-Interactive Undeniable Signatures with Preprocessing

One obvious argument against undeniable signatures in some applications (e.g. electronic mail) is that the verification and denial protocols are inherently interactive, and that they therefore cannot be used in environments where interaction is not readily available.

Here we present a solution to this problem, based on *preprocessing*: if the signer and receiver have in advance executed an interactive protocol to the receiver's satisfaction, then the signer can later send messages, which the receiver will accept as undeniably signed by the signer. The signer does not need to know in the initial phase which messages he will send later.

We assume that the signer, in addition to the keys required for the undeniable signatures, has a pair of public/secret keys to an ordinary signature system. For concreteness we assume this system is RSA and let $RSA(m)$ denote the signer's RSA signature on the message, m. We also assume for simplicity that the RSA system and the undeniable signature scheme have the same message space. This gives the following solution:

PREPROCESSING PHASE

1. The signer selects a random message x, and sends it to the receiver together with an undeniable signature $sign(x)$.

2. The signer and the receiver execute the verification protocol on input $(x, sign(x))$. Both parties store $(x, sign(x))$ for later use.

At a later time, the signer can sign a message, m, undeniably by:

MESSAGE SEND PHASE

1. The signer sends m and $s = RSA(m \oplus x)$ to the receiver.
2. The receiver gets (m, s) and checks that s is the correct RSA signature on $m \oplus x$ using his stored copy of x.

This signature is undeniable, because anyone can do the following: select s at random, put $x = m \oplus RSA^{-1}(s)$ and finally, by the properties of the original undeniable scheme, produce a number r that looks just like the real undeniable signature $sign(x)$. Hence, an entity which has not executed the verification protocol with the signer for the message/signature pair $(x, sign(x))$ has no reason to believe that the tuple (x, r, m, s) was really produced by the signer.

On the other hand, if the signer really has signed x undeniably, then an enemy who wants to forge a signature on the message, m, is faced with the problem of breaking RSA by computing $RSA(m \oplus x)$ from m and x. There is of course the possibility of an existential forgery by choosing s at random and putting $m = RSA^{-1}(s) \oplus x$. This may be solved by using a one-way hash function on m. The use of a hash function also has the advantage of making the choice of x independent of the length of m.

Thus the security of the non-interactive system follows from the security of RSA and the origins undeniable scheme.

In place of RSA, any signature scheme can be used for which existential forgery, but no stronger form of forgery, is possible (e.g. the El Gamal scheme) More concretely, we require that it is feasible to make random coherent pairs of messages and signatures, but inedible to produce a signature from a *given* message. Thus, curiously, schemes that are normally regarded as being more secure, such as the GMR-scheme (see [GMR88]), cannot be used.

Chapter 8

Convertible Undeniable Signatures

In addition to the properties of undeniable signatures described in the previous chapter, it could be usefd if there were some secret information, which the signer could release at some point after signing, which would turn the undeniable signatures into ordinary digital signatures. Then these signatures cord be verified without the aid of the signer, but of course they should still be difficult to forge. Such signatures will be called *convertible undeniable signatures.*

In practice this means that the signer should have a public key KP and two private keys KS_1 and KS_2. The first private key KS_1 should never be released; the signer uses it to produce signatures. The second private key KS_2 may be released to convert the undeniable signatures into ordinary digital signatures. Thus the important property of a convertible undeniable signature scheme compared to deniable signatures is, that it separates the ability to sign from the ability to verify. Anyone knowing KP and KS_2 can verify signatures, but producing new signatures requires KS_1.

In some applications of convertible undeniable signatures, one might prefer to convert only selected undeniable signatures into digits signatures (the signer may not wish everything he ever signed to be publicly verifiable). When a scheme allows this, we will say that it is a *selectively convertible undeniable signature scheme.*

In the first part of this chapter we define convertible undeniable signatures formally and give a "theoretical" construction of such signatures. In the second part a more practice scheme is presented.

Thy results in this chapter are also published in [BCDP91]. That paper also contains some examples of applications of selectively convertible signatures.

8.1 Theoretical Results

In this section, undeniable signatures are formally defined, and it is shown that (convertible) undeniable signatures exist if and only if digital signatures exist.

Formal Definitions

As we have seen, undeniable signatures differ from ordinary digital signatures in that, given an undeniable signature, the verifier should be unable to distinguish between valid and invalid signatures with any significant advantage. In order to formalize the notion of undeniability, we need a polynomial time simulator which — without access to the secret key — can produce fake signatures that cannot be distinguished from valid signatures. This will be formalized below.

We must also ensure that entering into the verification or denial protocols does not help an enemy to forge signatures, or to distingush valid signatures from invalid ones on messages that the signer is not willing to discuss. The simplest way to do this is to demand that the protocols are (auxiliary input) zero-knowledge. It is possible, but very complicated, to make the definitions work without this property, but in all examples where we can prove security of the protocols, they are in fact zero-knowledge. In this work the simpler approach has been chosen in order to improve the readability (see [BCDP91] for the more general definition).

In the following, we make use of a security parameter, k, that measures the security of the scheme and the lengths of various bit strings. Thus "polynomial", "negligible" and "overwhelming" are always as functions of k in the following.

An (undeniable) signature scheme consists of:

- A probabilistic polynomial time algorithm, A, which on input the security parameter k outputs a secret key, KS, and a matching public key, KP, of length $O(k)$ bits.

- A probabilistic polynomial time algorithm, Σ, which on input a message, m, and a secret key, KS, outputs a signature $\Sigma(m, KS)$. Messages may have arbitrary polynomial length. The set of possible signatures for a given message, m, with respect to a public key, KP, is denoted $KP(m)$. We demand that if $m \neq m'$, then $KP(m) \cap KP(m') = \emptyset$.
- A verification protocol, i.e. a zero-knowledge proof system with message, m, public key KP and signature z as common input to the prover and verifier. The protocol is a proof system for the language $KP(m)$.
- A denial protocol, i.e. a zero-knowledge proof system with message m, public key KP and invalid signature z as common input to prover and verifier. In this protocol the signer proves that $z \notin KP(m)$.

Definition 8.1
A *signature simulator* is a probabilistic polynomial time algorithm, which given a message, m, and a public key, KP, outputs a string $\mathit{Fake}(m, KP)$ (which is supposed to look like a real signature).

Definition 8.2
A *signature distinguisher* for a signature simulator is a probabilistic polynomial time algorithm, D, that does the following:

1. Receives a public key, KP, as input.
2. Repeats the following a polynomial number of times:
   Either output a message m and receive a valid signature on m;
   or output a pair (m, z) and receive $b \in \{0, 1\}$, such that b equals 1 if and only if $z \in KP(m)$.
3. Outputs a final message m_0 different from all messages output in step 2 and receives a string z_0.
4. Outputs 0 or 1.

The final output of D can be thought of as D's guess as to whether z_0 is a valid signature on m_0 or is an output from the signature simulator.

This definition expresses the intuition that even if an enemy can get signatures on messages of his choice, he still has no idea whether a given signature is valid, unless he talks to the signer. More formally:

Definition 8.3
A signature scheme as described above is *undeniable* if there exists a signature simulator SS, such that for any signature distinguisher D for SS:

$$|p_{SS}(k) - p_{\Sigma}(k)|$$

is negligible, where $p_{SS}(k)$ and $p_{\Sigma}(k)$ are the probabilities that D outputs 1 when the z_0 in step 3 above equals $\mathit{Fake}(m_0, KP)$, respectively $\Sigma(m_0, KS)$. These probabilities are taken over the choices of the public key, the coin flips of D, the choices of signatures (in step 2 of D) and the coin flips of SS, respectively Σ.

For the scheme in Section 7.1 the signature simulator, on input m and the public key KP, outputs a random $z \in G_q$. Assumption SDL is not sufficient to guarantee that the scheme is undeniable with this simulator, since a signature distinguisher could ask for the signature z on a message m in step 2, output $m_0 = m^2$ in step 3 and output 1 in step 4 if and only if $z_0 = z^2$. However, the hash function is assumed to prevent any signature distinguisher from being able to exploit any dependencies between different messages and their signatures.

We have said nothing so far about whether seeing executions of the verification and denial protocols could help an enemy. Note, however, that we have demanded that these protocols be zero-knowledge. Therefore, if there exists a successful distinguisher which participates in some of these protocols with the signer, there exists a successful distinguisher which satisfies the above definition: we simply take the distinguisher we are given and replace the interactions with the signer by simulations.

Finally, we need to express the property that an enemy cannot produce valid signatures by himself. For this, we use a minor modification of the concept of security against adaptive chosen message attacks (presented in [GMR88]).

Definition 8.4
A *signature enemy* is a probabilistic polynomial time algorithm, E, that does the following:

1. Receives a public key KP as input.
2. Repeats the following a polynomial number of times:
   Outputs a polynomial number of messages, $m_1, \ldots, m_r$, and receives valid signatures, $z_1, \ldots, z_r$, on these messages.
3. Outputs a final message m and a string z.

Intuitively, E tries to find a new message for which it can forge a signature. We demand that this only happens with very small probability. More formally:

Definition 8.5
A signature scheme as described above is *secure against forgery* if $p_E(k)$ is negligible for any signature enemy E, where $p_E(k)$ is the probability that for all i, $m \neq m_i$ and $z \in KP(m)$. The probability is over the choices of KP, the coin flips of E and the choices of signatures for each m_i.

As before, it is easy to argue that seeing executions of the verification and denial protocols does not help a signature enemy.

In the above definitions nothing has been said about convertible schemes, but these only require the following small modifications. A convertible undeniable signature scheme is an undeniable signature scheme in which the key generating algorithm, A, outputs a secret key KS on the form

$$KS = (KS_1, KS_2).$$

There must be a polynomial time algorithm, V, which on input

$$(KP, KS_2, m, z)$$

outputs 1, if $z \in KP(m)$ and outputs 0 otherwise.

In a selectively convertible scheme, the signature algorithm, Σ, produces on input (m, KS) a signature z and a key K_m. The set of possible keys corresponding to m and z is denoted $K(m, z)$. To verify a selectively converted signature we also need a polynomial time algorithm, which on input (KP, K_m, m, z) outputs 1 if and only if $z \in KP(m)$ and $K_m \in K(m, z)$.

The security of (selectively) convertible signatures (with respect to verification as well as forgery) can be defined almost as for undeniable signatures. We just have to remember that the signature enemy may know KS_2, and that the signature distinguisher as well as the signature enemy may receive $K_m \in K(m, z)$ for messages m of its choice and $z \in KP(m)$.

Existence of convertible undeniable signature schemes

In this section, we discuss which assumptions are sufficient for the construction of (selectively) convertible signature schemes. If one makes the assumption that a secure digital signature scheme exists, then we will see that it is quite easy to design a convertible undeniable signature scheme. Thus, by the result of [Rom90], it is sufficient to assume the existence of a one-way function.

By a "secure digital signature scheme", we mean one which is not existentially forgeable under an adaptive chosen plaintext attack, i.e., even if an enemy can get signatures on messages of his choice, he cannot sign any message that has not been signed by the signer (see [GMR88]).

Theorem 8.6
There exists a selectively convertible undeniable signature scheme which is secure against forgery if and only if there exists a secure digital signature scheme.

Proof
First, we remark that if a convertible undeniable signature scheme exists, then we trivially have an ordinary signature scheme by releasing KS_2 immediately.

Conversely, to set up a digital signature schemes there is a probabilistic polynomial time algorithm which on input a random string r, produces a pair (P, S), where P is a public key, and S is the matching secret key. It is intuitively obvious that the mapping from r to P must be a one-way function if the scheme is secure. A formal proof can be derived by using the signature scheme to build a secure identification protocol and using the result of [IL89].

Thus the existence of a secure digital signature scheme implies the existence of a one-way function, which by [GL89] and [ILL89] in turn implies the existence of pseudorandom bit generators, and hence by [GGM84] the existence of a pseudorandom function family. This is a parameterized family of functions $\{f_K\}$, where the parameter K can be thought of as a key. To a polynomially bounded enemy who does not know the key, images $f_K(x)$ appear to be totally random values with no correlation to x, even if the enemy gets to choose x.

In addition to pseudorandom functions we will need a commitment scheme. Let $BC(B, R)$ denote a commitment to $B \in \{0,1\}^*$ using the random string $R \in \{0,1\}^*$, and assume that the committer can open the commitment by revealing R.

Commitments of this form are only known to east assuming the existence of 1—1 oneuway functions. However, for the moment we will assume that we have such a commitment scheme to our disposal, and later we show how to get rid of the 1—1 assumption.

Let P and S be the public and secret keys for the signature scheme we are given, and let $S(M)$ denote the signature on M using the secret key, S. The public key in the undeniably system is

$$KP = (P, BC(K, R)),$$

where K is a key to the pseudorandum function family. The private key $KS = (KS_1, KS_2)$ is given by

$$KS_1 = S \quad \text{and} \quad KS_2 = R.$$

Let $\mathcal{M}$ be the message space for the given signature system, and assume that f_k maps messages in $\mathcal{M}$ to random bit-strings. Then the undeniable signature on $M \in \mathcal{M}$ has the following form:

$$sign(M) = Bc(S(M), f_K(M)).$$

The protocols for verifying and denying signatures can be constructed from a circuit that works as illustrated in figure 8.1. Using R, it will open the commitment to K, from which it computes $f_K(M)$. With this values it opens the commitment to $S(M)$, and finally it checks this signature on M using the public key P. The circuit gives three bits b_1, b_2 and b_3 as output.

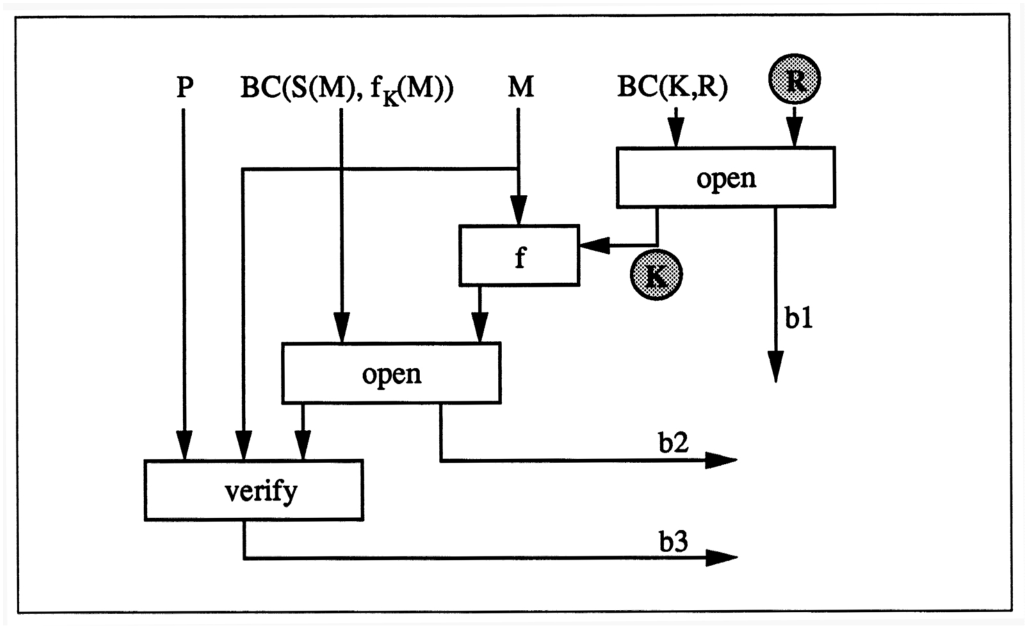

Figure 8.1: Circuit for verifying or denying signatures.

They are defined to be 1, if the opening of the two commitments and the signature check, respectively, was successful.

By the general protocols of [BCC88] or [IY88], the signer can now convince anyone of the value of any boolean function of $b1$, $b2$, $b3$ in zero-knowledge — in particular without revealing R and K. If he wants to verify a signature, he convinces the verifier that $b1 \wedge b2 \wedge b3 = 1$; if he wants to deny a signature, he convinces the verifier that $b1 \wedge ((\neg b2) \vee (\neg b3)) = 1$. The scheme is convertible, since the release of R enables the computation of K, and therefore all commitments to signatures can be opened. It is also selectively convertibles since for a signed message M, one can release $f_K(M)$. This allows computation of $S(M)$ from $sign(M)$, but by the properties of pseudorandom functions it does not help in computing any other function values.

The scheme is secure against forgery, because, since the signer chooses R and K independently of S, any forgery of the undeniable signatures could be used to forge signatures in the origins signature scheme.

To show that the scheme is undeniable we construct a signature simulator which simply makes a bit commitment to a random string of the correct length.

We now argue that this signature simulator works. The reader should have no troubles filling in the details using the definitions of pseudorandom function families arid bit commitment schemes. The signature distinguisher, D, does the following (see Definition 8.2):

1. Receive the public key, $KP = (P, BC(K,R))$;
2. Repeat the following a polynomial number of times:
   Output a message $m \in \mathcal{M}$ and get a signature $BC(S(m), f_K(m))$; or output a pair (m, σ) and get $b \in \{0,1\}$ such that $b = 1$ if and only if σ is a correct signature on m.
3. Output a message m_0 and get σ_0 where σ_0 is either on the form $BC(S(m_0), f_K(m_0))$ or $BC(r_0, r_1)$ where $r_0, r_1 \in \{0,1\}^*$, $|r_0| = |S(m_0)|$ and $|r_1| = |f_k(m_0)|$.
4. Output 0 or 1.

Let P_Σ be the probability that D outputs 1 if σ_0 is a signature on m_0 and let P_{SS} be the probability that D outputs 1 if σ_0 is a random commitment to a random string. We want to show that $|P_\Sigma - P_{SS}|$ is negligible. Consider the following two experiments:

1. Replace $BC(K,R)$ by $BC(R_0, R_1)$ in the public key, where R_0 and R_1 are randomly chosen in $\{0,1\}^*$ of the appropriate length, and run the distinguisher as described above.
2. Replace furthermore $f_K(m)$ by a random string, in all signatures (in step 2 and 3).

In the following we make use of the fact that the key K, the commitment scheme BC and the signature functions (S, P) are chosen independently of each other.

Let $P'_\Sigma(P'_{SS})$ and $P''_\Sigma(P''_{SS})$ be the probabilities with which D outputs 1 in the four ales corresponding to the two experiments and the two choices of σ_0 in step 3 of D.

Now $|P_\Sigma - P'_\Sigma|$ is negligible, because D and the signer have otherwise found a messages K, such that they can distinguish a random commitment to K from random commitments. Similarly is $|P_{SS} - P'_{SS}|$ negligible.

If $|P'_\Sigma - P''_\Sigma|$ is non-negligible then D and the signer can distinguish random values from values of the pseudorandom function. This is a contradiction — even if the signer has selectively converted some of the signatures. For the same reason is $|P'_{SS} - P''_{SS}|$ negligible.

Finally is $|P''_\Sigma - P''_{SS}|$ negligible, because D and the signer have otherwise found a message, $S(m)$, such that they can distinguish random commitments to this message from random commitments to random strings of the same length. Thus

$$\begin{aligned}|P_\Sigma - P_{SS}| &= |P_\Sigma - (P'_\Sigma - P'_\Sigma) - (P''_\Sigma - P''_\Sigma) - \\ &\quad (P''_{SS} - P''_{SS}) - (P'_{SS} - P'_{SS}) - P_{SS}| \\ &\leq |P_\Sigma - P'_\Sigma| + |P'_\Sigma - P''_\Sigma| + \\ &\quad |P''_\Sigma - P''_{SS}| + |P''_{SS} - P'_{SS}| + |P'_{SS} - P_{SS}|\end{aligned}$$

and the last expression is negligible.

Finally we address the problem of basing the scheme on *any* one-way function. By the result of Naor, [Nao90], one can build bit commitments from any one-way function, but this requires interaction: the verifier sends a random string R_V and the signer/prover responds with the commitment. The randomness of R_V is necessary to ensure that the prover is actually committed, but it does not affect the secrecy of the bits committed to. We will use this scheme for the commitment to K; for this commitment, R_V can be supplied by a trusted key-center, which will usually have to exist to guarantee the authenticity of the public keys. But any mutually trusted source of random bits (such as a multiparty coinflipping protocol) would suffice here. Once K is committed to, we do not need Naor's scheme any more. For the commitments to signatures, we can use the "hard-core" bits of the one-way function (see [GL89]), because the random input is now determined by K.

■

The following corollary shows that the conditions needed for the construction of an undeniable signature scheme are no we er than the necessary conditions for the construction of a convertible undeniable signature scheme.

Corollary 8.7
There exists a secure undeniable signature system if and only if there exists a one-way function.

Proof sketch
If undeniable signatures exist, one can prove the existence of a one-way function in the same way as in the proof of Theorem 8.6.

The converse follows from the previous theorem and the result of [Rom90] that secure digital signatures exist if one-way functions exist. ■

Independently, this corollary was proven earlier by Micali ([Mic90]).

8.2 A Practical Solution

In Section 8.1 it was shown how to construct a selectively convertible undeniable signature scheme, based on any one-way function. This solution is very far from being practical, however. In this section we show how the El Gamal signature scheme [EG85] can be changed into a practical convertible undeniable signature scheme.

It does not appear to be possible to get a convertible scheme by a more straightforward generalization of the scheme from Section 7.1. For examples the most obvious generalization is to use a composite modulus but then it becomes difficult to tell whether or not a given message represents an element of the subgroup we are using.

El Gamal Signatures

The El Gamal signature scheme may be applied in any group where discrete log is a hard problem. This includes the group G_q introduced in Section 2. In this group, the El Gamal scheme can be described as follows:

The private key is a number x between 1 and q and the public key is $h = g^x$. The signature on a message $m \in \mathbb{Z}_q^*$ is a pair (r, s) satisfying

$$g^m = h^r r^s.$$

This signature is constructed by choosing $k \in \mathbb{Z}_q^*$ at random and computing r and s by

$$r = g^k$$

and

$$s = k^{-1}(m - xr) \bmod q.$$

Here r is considered an element of $\mathbb{Z}_q$.

The Convertible Scheme

Using the El Gamal signature scheme, we can construct an undeniable signature scheme as follows, The private keys are

$$KS_1 = x \text{ and } KS_2 = y, \quad 1 < x, y < q$$

and the public key is

$$KP = (p, q, g, h_1, h_2), \text{ where } h_1 = g^x \text{ and } h_2 = g^y.$$

When the signer makes public $KP = (p, q, g, h_1, h_2)$, the receiver, which could be a key center or a user, should verify that g, h_1 and h_2 are in G_q.

The signature on the message $m \in \mathbb{Z}_q^*$ is

$$sign(m) = (g^t, r, s),$$

where $t \in \mathbb{Z}_q^*$ is chosen at random, and (r, s) is the El Gamal signature on $g^t tym \bmod q$ (in this product the binary representation of g^t is considered to be a representation of an element in $\mathbb{Z}_q$).

In the El Gamal scheme, it is possible for a forger to construct a signature (r', s') on a message m'. It is very unlikely that m' is meaningful but this attack implies that the El Gamal scheme must be used together with a hash function. The new scheme also requires a hash function, but for simplicity of notation, we will not mention it when describing the scheme.

The triple (T, r, s) is a legal signature on a message m, if and only if

$$(T^{Tm})^y = h_1^r r^s$$

(whenever T is in the exponent, it is considered to be an element of $\mathbb{Z}_q$). Throughout this section, we will use v to denote $h_1^r r^s$ and w to denote T^{Tm}. Thus verifying a signature (T, r, s) on m is equivalent to deciding if $\log_w v$ equals $\log_g h_2$.

Therefore the signer can prove that (T, r, s) is a legal signature on the message m as follows: (S is the signer and V the verifier)

PROTOCOL VERIFY SIGNATURE

1. S and V compute $w = T^{Tm}$ and $v = h_1^r r^s$.
2. S proves that $\log_w v$ equals $\log_g h_2$ using PROTOCOL SDL.
3. V accepts the signature if and only if she accepts the proof.

Proposition 8.8
If (T, r, s) is a legal signature on m, the following hold

1. The verifier always accepts the verification.
2. PROTOCOL VERIFY SIGNATURE is zero-knowledge.

If (T, r, s) is a false signature, the verifier accepts with negligible probability (in $|q|$).

Proof
Follows from Theorem 7.1. ■

A signature is denied as follows:

PROTOCOL DENY SIGNATURE

1. S and V both compute $w = T^{Tm}$ and $v = h_1^r r^s$.
2. S proves that $\log_g h_2 \neq \log_w v$ using a zero-knowledge proof system.
3. V accepts the denial if and only if she accepts the proof.

Proposition 8.9
If (T, r, s) is a false signature on m, the following hold:

1. The verifier always accepts the denial.
2. PROTOCOL DENY SIGNATURE is zero-knowledge.

If (T, r, s) is a legal signature on m, the verifier will accept the denial with negligible probability (in $|q|$) no matter what an unlimited signer does.

Proof
By definition of the protocol. ■

Conversion of all signatures

An undeniable signature is converted to an ordinary signature by releaseing KS_2. Knowing $KS_2 = y$ and KP, everybody can verify a signature (T, r, s) on the message m by computing $(T^{Tm})^y$ and verifying that it equals $h_1^r r^s$. Thus when y is released all previous and future signatures are El Gamal signatures on messages on a special form.

Selective conversion

Knowing t such that $T = g^t$, anyone can check that (T, r, s) is a signature on m by verifying that

$$T = g^t \text{ and } (h_2^{Tm})^t = h_1^r r^s.$$

Therefore, a single signature can be converted to an ordinary digital signature by releasing t. This method of converting signatures requires that the signer remembers the t used to construct the signature on m. This is most conveniently done by choosing a key, K, to a pseudorandom function f_K (see [GGM84]) and then computing t as $f_K(m)$ (see also the proof of Theorem 8.6). The properties of families of pseudorandom functions guarantee that, given polynomially many pairs $(m_i, f_K(m_i))$, it is infeasible to find $f_K(m)$ for a message $m \neq m_i$. Conversion of any polynomial number of signatures can therefore not affect the undeniability of other signatures.

Security

Although we have not been able to reduce security of this scheme to any standard intractability assumption, there is strong intuitive evidence that the scheme does in fact have the desired properties.

We will begin by discussing the possibility of creating false signatures. Since the verification and denial protocols are zero-knowledge, the strongest attack in this context is the chosen message attack, in which the enemy can ask for signatures on messages of his choice, and later tries to "sign" a new message. For our scheme, this means that the enemy can get triples of the form (T, r, s), such that

$$T^{Tym} = h_1^r r^s,$$

where m is chosen by the enemy, and T is chosen by the signer. In the followings we make the worst case assumption that the enemy knows t, the discrete logarithm of T, and y. This means that the enemy gets El Gamal signatures on numbers of the form $Ttmy$.

The only known consequence for the El Gamal scheme under this attack is that the enemy care construct new "signed" messages from the ones he is given (see [EG85]), but these new messages result from applying a hard-to-invert transformation to the known messages, and therefore they cannot be controlled easily. It is also possible to construct "signed" — but still hard to control — messages from scratch. They have the form $m = g^B h_1^C BC^{-1} \bmod q$, where B and C can be arbitrary integers. The construction that uses already signed messages is similar, but more complicated. In the El Garnal scheme, as in ours, this weakness is solved by applying a one-way hash function to the messages before signing.

Now suppose that the enemy is somehow able to create a signature (T, r, s) on a message m. There are two cases:

1. If he also knows t, such that $T = g^t$, he has treated an El Gamal signature on $Ttmy$. So either he has found a completely new way to break the El Gamal system, or he has found a way to write the result of the hard-to-invert transformation mentioned above in the form $Ttmy$. We conjecture that this is a hard problem: m is in fact shorthand for a one-way hashed image of the actual message, the mapping $f(t) = g^t t$ can reasonably be assumed to be one-way, and y is a constants Thus none of the factors in $(Tt)my$ can be easily controlled by the enemy.

2. Thy only remaining possibility is that the enemy can find values satisfying $(T^T)^{my} = h_1^r r^s$ *without* knowing the discrete logarithm of T. This problem could only be easier than case 1, if T is computed in some clever way from signatures the enemy got earlier from the signer. As discussed above, these signatures are equivalent to El Gamal signatures on messages of a special form. But these messages cannot be controlled

by the enemy, since they all involve a factor of the form tg^t where t is chosen independently by the signer. Furthermore, it does not seem to be any advantage to the enemy that the signed messages contain a product tg^t, because he can construct signatures on such messages by himself. Hence it is reasonable to assume that these signed messages will be of no more use to the enemy than the ones he can construct himself from scratch. These observations suggest that this case is not easier than case 1.

We now turn to the problem of verifying signatures without the aid of the signer. We have to come up with a signature simulator and claim that

$$Fake(m, KP) = (T, r, s),$$

where T, r, and s are chosen at random, will do. Remember that if $KP = (p, q, g, h_1, h_2)$, then the signer also chooses T and r at random, and then computes s from T, r, $\log_g(T)$, $\log_g(r)$ and the secret key.

Any signature distinguisher is at some point going to output a message m and receive a string (T, r, s), which is either a correct signature or produced by the signature simulator. The distinguisher then has to determine if $T^{Tmy} = h_1^r r^s$. The natural way to verify an equation like this is to compute each side and compare. But if the distinguisher can compute the left side, it can also compute $g^{ty} = (T^{Tmy})^{(Tm)^{-1}}$. This computation of g^{ty} from $h_2 = g^y$ and $T = g^t$ contradicts assumption DH.

The definition of a signature distinguisher glows, however, that the distinguisher receives signatures on messages of its choice. Thus, the distinguisher may know a number of void signatures (T_i, r_i, s_i) on messages m_i. In the worst case the distinguisher also knows $t_i = \log_g T_i$. Hence we have a situation where $(g^{t_i y}, t_i, g^y)$ is known for a number of independently chosen t_i's, and the signature distinguisher is now trying to guess the value of g^{ty} for a t which is independent of all the t_i's. For each i, $(g^{t_i y}, t_i, g^y)$ could easily be generated with the same distribution by the distinguisher itself. The only additional information, it has, is that each $g^{t_i y}$ can be expressed in a special form, namely as $(h_1^{r_i} r_i^{s_i})^{(T_i m_i)^{-1}}$. Since this expression involves the independently chosen r_i's, we conjecture that this extra information does not help the signature distinguisher.

But if the distinguisher is unable to compute g^{ty}, then he is left with the problem that is also the basis of the scheme from Section 7.1: trying to decide whether $\log_w v$ equals $\log_g h_2$, where it is reasonable to assume that v and w are random elements. According to assumption SDL this is a hard problem.

Generalizations

This method of constructing undeniable signatures can be used in any group. However, if the group in question is not of prime order, one has to apply the more general protocol presented in [CEG87] for proving equality between logarithms. The reason for this is that the proof of Theorem 7.1 does not work, if w (which the signer chooses) generates a small subgroup.

Chapter 9

Agents in Undeniable Signature Schemes

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 requires that he can always be reached. Since an udeniable 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 (by giving him KS_2), 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 lest k of these to cooperate. This chapter shows how distributed proofs can be used to achieve this goal for the selectively convertible scheme presented in Section 8.2. The results presented in this chapter are also described in [Ped91a].

9.1 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 $(P_1, P_2, \ldots, P_n)$ by distributing t. As $T = g^t$ is part of the signature and therefore not secret, the secret sharing scheme from Section 4.1 can be used (the common input is T and the public key):

PROTOCOL DISTRIBUTE SINGLE SIGNATURE

1. S distributes t using PROTOCOL DISTRIBUTE in Section 4.1. Thus P_i gets the share $t_i = f(i)$, where f is a polynomial over $\mathbb{Z}_q$ of degree $k-1$ such that $f(0) = t$.
2. Each agent P_i verifies his share as described in Section 4.1.
3. 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 $\tau_i = g^{t_i}$ 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 $P_1, P_2, \ldots, P_k$) 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 4.1, each a_i can be computed from the identities of the k agents.

PROTOCOL DECIDE

1. V and each P_i verify that $T' = \prod_1^k \tau_i^{a_i}$.
   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. $P_1, P_2, \ldots, P_k$ can verify a signature by executing (now $T' = T$)

PROTOCOL DISTRIBUTED VERIFICATION

1. P_i and V compute $w = h_2^{Tm'}$ and $v = h_1^{r'} r'^{s'}$.

2. $P_1, P_2, \ldots, P_k$ prove that $\log_w(v) = \log_g(T)$ using PROTOCOL DISTRIBUTED SDL below.

3. V accepts the signature if and only if she accepts the proof.

It is sufficient to construct a secure distributed proof for the language SDL. By a slight modification of PROTOCOL SDL this can by done as follows:

PROTOCOL DISTRIBUTED SDL

1. The verifier chooses $a, b \in \mathbb{Z}_q$ at random, computes $ch = w^a g^b$ and broadcasts ch to the k agents.

2. Upon receiving $ch \in G_q$, each agent, P_i, chooses $r_i \in \mathbb{Z}$ at random and computes $h_{i1} = ch^{r_i}$ and $h_{i2} = h_{i1}^{t_i}$.
   P_i broadcasts the pair (h_{i1}, h_{i2}).

3. When the verifier has received k pairs of elements in G_q she broadcasts (a, b).

4. If $ch = w^a g^b$, P_i broadcasts r_i.
   Otherwise P_i broadcasts the message "stop" and stops the execution.

5. The verifier accepts the proof if

$$\prod_{i=1}^{k} h_{i1} = ch^{\sum_1^k r_i} \quad \text{and} \quad \prod_{i=1}^{k} h_{i2}^{a_i/r_i} = v^a T^a.$$

   Otherwise the proof is rejected.

In the following it will be shown, that this is a secure distributed 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.

Theorem 9.1
PROTOCOL DISTRIBUTED SDL 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 $M_{V^*,D}$ on input
   - 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

$$\begin{aligned}\prod_{i=1}^{k} h_{i2}^{a_i/r_i} &= \prod_{i=1}^{k} (ch^{r_i t_i})^{a_i/r_i} \\ &= \prod_{i=1}^{k} ch^{t_i a_i} \\ &= ch^t \\ &= v^a T^b.\end{aligned}$$

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 PROTOCOL SDL, which makes the verifier accept with probability greater than q^{-1}. This contradicts Theorem 7.1.

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 $h_{i2} = \tau_i^{e_i}$ 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 $h_{i2} = ((w^{t_i})^a \tau_i^b)^{r_i}$.
   The dishonest provers compute (h_{i1}, h_{i2}) as usual.
6. Get (a', b') from the verifier.
   If $ch = w^{a'} g^{b'}$: give r_i to the verifier for $i \in H$, and for $i \in D$ compute r_i as usual. Then stop.
   If $ch \neq w^{a'} g^{b'}$: goto (4).

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. ■

We now want to show that PROTOCOL DISTRIBUTED SDL is a secure distributed proof of membership according to Definition 6.4. Thus we have to provide two simulators, one for the distribution of t and one for the actual executions of the proof.

The second simulator is almost given from the proof of Theorem 9.1, but we have to provide the necessary input to this simulator (in particular w^{t_i}). In order to do this we shall use the fact that the first simulator is allowed to produce an auxiliary output, *distinf*.

Theorem 9.2
PROTOCOL DISTRIBUTED SDL is a secure distributed proof with respect to the secret sharing scheme in Section 4.1.

Proof
Let V^* be a probabilistic polynomial time verifier, and let $D \subset \{1, \ldots, n\}$ be a set of dishonest agents and assume that $\#D < k$.

Let $H = \{1, 2, \ldots, n\} \setminus D$ be the set of honest agents. The simulator of the distribution phase (M^{dist}) is based on the same principles as the proof of Theorem 4.3 and works as follows on input $(t_i)_{i \in D}$:

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 $\tau_i = g^{t_i}$.

2. As in the proof of Theorem 4.3 compute g^{f_i} for $j = 0, \ldots, k-1$, where the polynomial

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

   satisfies

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

   Compute for $i \in H \setminus H'$

$$\tau_i = g^{f(i)} = \prod_{j=0}^{k-1} (g^{f_i})^{i^j}$$

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

3. Let *proof* $= (g^{f_1}, g^{f_2}, \ldots, g^{f_{k-1}})$.
   Output *conv* $=$ (*proof*, $(t_i)_{i \in D}$) and *distin f* $= (t_i)_{i \in H' \cup D}$.

proof constructed above has the same distribution as the message broadcasted by the dealer.

The simulator of an execution of the proof (M^{Proof}) has as input:

- 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
- *distin f*

However, from *distin f* and $v = w^t$ it is easy to construct w^{t_i} for all $i \in \{1, 2, \ldots, n\}$ (again as in the proof of Theorem 4.3) and when these values are known, the simulator from the proof of Theorem 9.1 can be used. ■

As the simulator in the proof of Theorem 9.1 also works if many proofs are executed in parallel we get the following

Corollary 9.3
PROTOCOL DISTRIBUTED SDL is strongly secure.

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 — this is true even if the verifier executes the verification protocol many times simultaneously (with different honest agents).

9.2 Generalizations

The signer can, by distributing y, authorize agents, that are able to verify all signatures using PROTOCOL DISTRIBUTED SDL above.

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 one must be careful to prevent that a dishonest agent convinces the verifier of the result of this computation.

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

The techniques suggested in this section can also be applied with the undeniable signature scheme from Section 7.1, but in this case, agents who can verify signatures can also sign new messages.

9.3 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 suggests how a number of agents can prove that an alleged signature is false.

Using the techniques described below, agents sharing y can deny any (false) signature, but as discussed in Section 9.2 this requires that they are able to decide whether an arbitrary signature is correct or not.

Now suppose (T, r, s) is a legal signature on m which has been distributed to $P_1, \ldots, P_n$ as in PROTOCOL DISTRIBUTED SINGLE SIGNATURE. Thus $t = \log_g T$ has been distributed to the 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 (otherwise the signatures may not be undeniable), 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 9.1). $P_1, P_2, \ldots, P_k$ can therefore prove that (T, r, s) is not a signature on m as follows:

PROTOCOL DISTRIBUTED DENIAL

1. P_i and V compute $w = h_2^{T_m}$ and $v = y^r r^s$.
2. $P_1, P_2, \ldots, P_k$ proves that $\log_w(v) \neq \log_g(T)$ as shown in PROTOCOL

DISTRIBUTED DDL below.

3. V accepts that the signature is false if and only if she accepts the proof.

It has not been possible to change PROTOCOL DDL into an efficient and secure distributed proof. The problem is that this protocol requires that the prover determine s (see Section 7.1), but in a distributed proof this means that a (dishonest) agent, who stops after this has been determined, can obtain extra knowledge. Instead we propose the following protocol, which intuitively seems to be secure, although we cannot prove it formally.

Let as usual $BC(B, R)$ denote a commitment to $B \in \{0,1\}^*$ using the random string $R \in \{0,1\}^*$. The keys to the commitment scheme should be supplied by a trusted (key authentication) center. Any k agents can prove inequality of discrete logarithms as follows (all participants get $p, q, g, T, (\tau_1, \ldots, \tau_k)$ and (v, w) as common input, and the i'th prover gets $t_i = \log_g \tau_i$ as auxiliary input):

PROTOCOL DISTRIBUTED DDL

1. Each P_i chooses $e_i \in \mathbb{Z}_q$ and $r_i \in \{0,1\}^*$ and computes $w_{1i} = w^{e_i}$ and $\beta_i = BC(w_{1i}, r_i)$.
   The commitment β_i is broadcasted.

2. When all k agents have broadcasted a commitment, P_i opens his commitment by broadcasting r_i.

3. When all k agents have opened their commitments, P_i and the verifier find all the w_{1i}'s and compute $w_1 = \prod_1^k w_{1i}$.
   If $w_1 = 1$ the protocol is stopped.
   Compute $v_{1i} = v^{e_i}$ and broadcast v_{1i}.

4. For $i = 1, 2, \ldots, k$: P_i proves that $\log_v v_{1i} = \log_w w_{1i}$ by executing PROTOCOL SDL with the verifier.

5. If the verifier rejects one of these proofs, the protocol is stopped.
   Otherwise all participants compute $v_1 = \prod_1^k v_{1i}$.

6. Each P_i computes $w_{2i} = w_1^{t_i}$ and broadcasts w_{2i}.

7. For $i = 1, 2, \ldots, k$: P_i proves that $\log_g \tau_i = \log_{w_1} w_{2i}$ by executing PROTOCOL SDL with the verifier.

8. If the verifier rejects one of these proofs, the protocol is stopped.
9. The verifier computes $w_2 = \prod_1^k w_{1i}^{a_i}$, and accepts the proof if and only if $w_2 \neq v_1$.

We first show that the protocol is a proof system.

Theorem 9.4
PROTOCOL DISTRIBUTED DDL 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 do, V accepts with probability at most $\frac{1}{q}$.

Proof
For $e \neq 0 \bmod q$

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

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 PROTOCOL SDL will succeed with probability at most $\frac{1}{q}$. ■

PROTOCOL DISTRIBUTED DDL 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 machines 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

$$v = y^r r^s \quad \text{and} \quad w = h_2^{TH(m)}$$

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 El Gamal 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.

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 executions it seems hard to obtain any information by comparing different executions of the protocol.

Consider the following attempt to simulate an honest prover in a single execution of PROTOCOL DISTRIBUTED DDL

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

where e_i 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

$$v = \tau_i^b \quad \text{and} \quad w = g^a$$

because if $\log_v v_1 = \log_w w_1 = e$ then w_{2i} must satisfy (remember $\tau_i = g^{t_i}$)

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

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

We now argue that, if the verifier and the dishonest agents cannot choose v and w better than at random, then the protocol can be simulated, and by the above remarks it can therefore be considered a secure distributed proof.

Consider an execution of the protocol between $P_1, P_2, \ldots, P_k$ and a verifier V^*, and assume that $D \subset \{P_1, P_2, \ldots, P_k\}$ 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

$$(T, \tau_1, \ldots, \tau_k, v, w).$$

where

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

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 randome
5. Simulate the "proof" that $\log_{w_1} (w_{2i}) = \log_g (\tau_i)$.

In step 5 the simulator is going to simulate a "proof" of a false claim as it is very unlikely that w_{2i} equals $w_1^{t_i}$. Theorem 7.1 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, and thus the above simulator runs in expected polynomial time.

$M_{V^*,D}$ generates (v_{1i}, w_{1i}) with the same distribution as executions of the protocole 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 simulation of the "proof" that $\log_{w_1} w_{2i} = \log_g \tau_i$ cannot be distinguished from an execution of a correct proof.

However, since the signature scheme is undeniable, it is 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 simulation of step 5 can not be distinguished from real executions.

Thus PROTOCOL DISTRIBUTED DDL is “zero-knowledge” in the (weak) sense that if the inputs v and w are chosen at random, there is a polynomial time machine whose output cannot be distinguished from real executions of the protocol (under assumption SDL). But as remarked above the protocol is not auxiliary input zero-knowledge.

Chapter 10

Conclusion

We have presented two noninteractive verifiable secret sharing schemes. The information-theoretic secure scheme from Chapter 3 is quite efficient, but it is primarily interesting because it, unlike previous suggestions, reveals no Shannon information about the key. One cannot say in general whether it is best to protect the secrecy of the key unconditionally or to guarantee the correctness of the shares unconditionally, but as we have seen one cannot obtain both properties in a non-interactive verifiable secret sharing scheme. Thus the choice between the scheme presented here and that of [Fel87] must depend on the particular application.

It is possible to use this secret sharing scheme in protocols for (computationally) secure multi-party computations, and in particular it is very easy to compute linear combinations of shared secrets. The shares of the product of two shared secrets can be constructed using the ideas in [BGW88], but would be interesting to have a more efficient method for multiplying shared secrets.

In Section 3.6 we have applied the ability to add distributed secrets to obtain a protocol, that allows the members of an organization to choose an anonymous secret and distribute it verifiably among themselves. The scheme for sharing a secret $s = \log_g h$ in Chapter 4 also offers this possibility, and as an immediate consequence of this we saw that it is possible to construct a threshold crypto system in which the members choose the secret and public keys. This facility seems to be quite useful in practice, as not even a trusted party has to know the secret keys but the construction would be more interesting if the members could also decipher without relying on a trusted party who verifies that the recovered plaintext is correct.

The development of distributed proofs was inspired by the application to undeniable signatures, but as shown in Chapter 6 this concept also has an interesting application to identification and signature schemes. However, the distributed signature scheme suggested here leaves open the problem of constructing signatures such that none of the agents can decide, who constructed the signature, whereas the original signer can do this if necessary. Such a signature scheme can be constructed using the concept of group signatures proposed in [CH91].

The definition of distributed proofs requires that at most $k-1$ agents are dishonest. In a (k,n)-secret sharing scheme all n shareholders can in principle be dishonest as long as no more than $k-1$ of them are ever willing to cooperate in order to misuse the secret. It word be tempting to make this more general assumption for distributed proofs as well but the restriction that only $k-1$ agents are dishonest implies that at least one honest agent always participates in a distributed proof and in particular this prevents dishonest agents from ever executing a distributed proof for L on an input x not in L.

However this leaves open the problem, how an honest agent decides whether $x \in L$. In the application to verification of undeniable signatures it has been solved by letting the signer generate a list of hash values of pairs (message, signature), which the agents are allowed to verify. This is an appropriate solution in this particular application as the signer probably wants to control which signatures the agents verify. However, in general the agents would have to decide whether $x \in L$ using a multiparty computation. It would be interesting to have efficient such protocols for the problems mentioned in this work. Alternatively, one could require that the distributed protocols are unrestricted input zero-knowledge (see [FFS88]) such that they can be simulated for all inputs.

Regarding the security of distributed proofs we have required that the distribution of the secret key can be simulated and that each subsequent run of the distributed proof is zero-knowledge. As it is hard for the honest agents to prevent that a verifier and the dishonest agents execute the distributed proof many times simultaneously with different honest agents, one should aim at constructing distributed proofs that are strongly secure.

In this work we have concentrated on protocols that are secure against a static adversary. This seems to be sufficient for most applications of distributed proofs, but it would be very interesting to have distributed proofs which are also (provably) secure against dynamic adversaries.

The notion of convertible undeniable signatures and in particular selectively convertible signatures is a useful extension of undeniable signatures as it gives the signer more choices for his signatures. The scheme presented in Chapter 8 is quite attractive, as it is relative efficient and is based on a well known signature scheme, but it would be interesting to have an efficient (selectively) convertible undeniable signature scheme for which the problems of forging/verifying signatures are provably as hard as solving other well known problems.

The distributed protocol for verifying the signatures in this scheme (PROTOCOL DISTRIBUTED VERIFICATION) is also quite applicable in practice, as it is efficient and the verifier's part of the protocol is lost as in PROTOCOL SDL. Furthermore, this protocol achieves the right level of security as it is strongly secure. However, this work leaves open the problem of constructing an efficient, provably secure distributed proof in which the agents prove inequality of discrete logarithms.

List of Protocols

Bibliography

[BC86] G. Brassard and C. Crépeau. Nontransitive transfer of confidence: a perfect zero-knowledge interactive protocol for *sat* and beyond. In *Proceedings of the 27th IEEE Symposium on the Foundations of Computes Science*, pages 188–195, 1986.

[BCC88] G. Brassard, D. Chaum, and C Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computes end System Sciences*, 37:156–189, 1988.

[BCDP91] J. Byoar, D. Chaum, I. Damgård, and T. Pedersen. Convertible undeniable signatures. In *Advances in Cryptology - proceedings of CRYPTO 90*, Lecture Notes in Computer Science, pages 189–205. Springer-Verlag, 1991.

[BCP] J. Bos, D. Chaum, and G. Purdy. A voting scheme. Preliminary draft.

[BD90] E. F. Brickell and D. M. Davenport. On the clasification of ideal secret sharing schemes. In *Advances in Cryptology - proceedings of CRYPTO 89*, pages 278–285, 1990.

[BDL91] J. Brandt, I. Damgård, and P. Landrock. Speeding up prime number generation, 1991. Presented at AsiaCrypt'91.

[Bet88] T. Beth. A Fiat-Shamir-like authentication protocol for the El-Gamal scheme. In *Advances in Cryptology - proceedings of EUROCRYPT 88*, Lecture Notes in Computer Science, pages 77–86. Springer-Verlag, 1988.

[Bla79] G. R. Blakley. Safeguarding cryptographic keys In *Proceedings AFIPS 1979 Nat. Computer Conf.*, pages 313–319, 1979.

[BM84] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal of Computation*, 13:850–864, 1984.

[BM91] E. F. Bricked and K. S. McCurley. An interactive identification scheme based on discrete logarithms and factoring. In *Advances in Cryptology - proceedings of EUROCRYPT 90*, Lecture Notes in Computer Science, pages 63–71. Springer-Verlag, 1991.

[BGKW88] 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.

[BGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 1–10, 1988.

[BS88] E. F. Brickell and D. R. Stinson. The detection of cheaters in threshold schemes, August 1988.

[CBDP91] D. Chaum, J. Boyar, I. Damgård, and T. Pedersen. Undeniable signatures: Applications and theory, 1991. A survey over undeniable signatures.

[CCD88] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 11–19,1988.

[CDG88] 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. SpringerVerlag, 1988.

[CEG87] D. Chaum, J.-H. Evertse, and J. van de Graaf. An improved protocol for demonstrating possession of a discrete logarithm and some generalizations. In *Advances in Cryptology - proceedings of EUROCRYPT 87*, Lecture Notes in Computer Science, pages 127–141, 1987.

[CGMA85] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *Proceedings of the 26th IEEE Symposium on the Foundations of Computes Science*, pages 383–395, 1985.

[Cha86] D. Chaum. Demonstrating that a public predicate can be satisfied without revealing any information about how. In *Advances in Cryptology - proceedings of CRYPTO 86*, Lecture Notes in Computer Science, pages 195–199, 1986.

[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.

[CA90] D. Chaum and H. van Antwerpen. Undeniable signatures. In *Advances in Cryptology - proceedings of CRYPTO 89*, Lecture Notes in Computer Science, pages 212–216. Springer Verlag, 1990.

[CH91] D. Chaum and E. van Heyst. Group signatures. In *Advances in Cryptology - proceedings of EUROCRYPT 91*, Lecture Notes in Computer Science, pages 257–266. Springer-Verlag, 1991.

[CHP91] D. Chaum, E. van Heyst, and B. Pfitzmann. Cryptographically strong undeniable signatures, unconditionally secure for the signer, 1991. To appear in the proceedings of CRYPTO 91.

[Dam88] I. Damgård. The application of flaw free functions in cryptography. Technical Report DAIMI PB – 269, Aarhus University, May 1988. Ph.D.-thesis.

[dB90] B. den Boer. Diffie-Hellman is as strong as discrete log for certain primes. In *Advances in Cryptology - proceedings of CRYPTO 88*, Lecture Notes in Computer Science, pages 530–539. Springer-Verlag, 1990.

[Des88] Y. Desmedt. Society and group oriented cryptography: A new concept. In *Advances in Cryptology - proceedings of CRYPTO 87*, Lecture Notes in Computer Science, pages 120–127, 1988.

[DF90] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *Advances in Cryptology - proceedings of CRYPTO 89*, Lecture Notes in Computer Science, pages 307–315, 1990.

[DH76] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Inform. Theory*, IT-22(6):644–654, November 1976.

[DS82] D. Dolev and H.R. Strong. Polynomial algorithmes for multiple processor agreement. In *Proceedings of the 14th Annual ACM Sympoium on the Theory of Computing*, pages 401–407, 1982.

[EG85] T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology - proceedings of CRYPTO 84*, Lecture Notes in Computer Science, pages 10–18. SpringerVerlag, 1985.

[Fel87] 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.

[FFS88] U. Feige, A. Fiat, and A. Shamir. Zero-knowledge proofs of identity. *Journal of Cryptology*, 1(2):77–94, 1988.

[Fra90] Y. Frankel. A practical protocol for large group oriented networks. In *Advances in Cryptology - proceedings of EUROCRYPT 89*, Lecture Notes in Computer Science, pages 56–61. SpringerVerlag, 1990.

[FS87] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology - proceedings of EUROCRYPT 86*, Lecture Notes in Computer Science, pages 186–194. Springer-Verlag, 1987.

[GGM84] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. In *Proceedings of the 25th IEEE Symposium on the Foundations of Computer Science*, 1984.

[GHY88] 2. Galil, S. Haber, and M. Yung. Cryptographic computation: Secure fault-tolerant protocols and the public-key model. In *Advances in Cryptology - proceedings of CRYPTO 87*, Lecture Notes in Computer Science, pages 135–155. Springer-Verlag, 1988.

[GL89] O. Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the 21st Annual ACM Symposium on the Theory of Computing*, 1989.

[GMR88] S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen message attack. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. S*IAM Journal of Computation*, 18(1):186–208, 1989.

[GMW86] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In *Proceedings of the 27th IEEE Symposium on the Foundations of Computer Science*, pages 174–187, 1986.

[Gor84] J. Gordon. Strong primes are easy to find. *Electronic Letters*, June 1984.

[Hwa91] T. Hwang. Cryptosystem for group oriented cryptography. In *Advances in Cryptology - proceedings of EUROCRYPT 90*, Lecture Notes in Computer Science, pages 352–360. Springer-Verlag, 1991.

[IL89] R. Impagliazzo and M. Luby. One-way functions are essential for complexity based cryptography. In *Proceedings of the 30th IEEE Sympoium on the Foundations of Computes Science*, 1989.

[ILL89] R. Impagliazzo, L. A. Levin, and M. Luby. Pseudo-random generation from one-way functions. In *Proceedings of the 21st Annual ACI Symposium on the Theory of Computing*, 1989.

[IS91] I. Ingemarsson and G. J. Simmons. A protocol to set up shared secret schemes without the assistance of a mutually trusted party. In *Advances in Cryptology - proceedings of EUROCRYPT 90*, Lecture Notes in Computer Science, pages 266–282. Springer-Verlag, 1991.

[IY88] R. Impagliazzo and M. Yung. Direct minimum-knowledge computations. In *Advances in Cryptology - proceedings of CRYPTO 87*, Lecture Notes in Computer Science, pages 40–51. SpringerVerlag, 1988.

[Kot85] S. C. Kothari. Generalized linear threshold scheme. In *Advances in Cryptology - proceedings of CRYPTO 84*, Lecture Notes in Computer Science, pages 231–241. SpringerVerlag, 1985.

[LSP82] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Trans. on Programming Languages and Systems*, 4(3):382–401, July 1982.

[Mic90] S. Micali, August 1990. Personal communication through J. Boyar.

[MR91] S. Micali and T. Rabin. Collective coin tossing without assumptions nor broadcasting. In *Advances in Cryptology - proceedings of CRYPTO 90*, Lecture Notes in Computer Science, pages 253–266. Springer-Verlag, 1991.

[MS81] R. J. McEliece and D. V. Sarwate. On sharing secrets and reed-solomon codes. *Communications of the ACM*, 24:583–583, 1981.

[Nao90] M. Naor. Bit commitment using randomness. In *Advances in Cryptology - proceedings of CRYPTO 89*, Lecture Notes in Computer Science, pages 128–136, 1990.

[Ped91a] T. P. Pedersen. Distributed provers with applications to undeniable signatures. In *Advances in Cryptology - proceedings of EUROCRYPT 91*, Lecture Notes in Computer Science, pages 221–242. Springer-Verlag, 1991.

[Ped91b] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing, 1991. To appear in the proceedings of Crypto'91.

[Ped91c] T. P. Pedersen. A threshold cryptosystem without a trusted party. In *Advances in Cryptology - proceedings of EUROCRYPT 91*, Lecture Notes in Computer Science, pages 522–526. Springer-Verlag, 1991.

[PH78] S. Pohlig and M. E. Hellman. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. *IEEE Transactions on Information Theory*, IT-24:106–110, 1978.

[Rab80] M. O. Rabin. Probabilistic algorithm for primality testing. *Journal of number theory*, 12:128–138, 1980.

[RB89] T. Rabin and M. Ben-Or, Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the 21st Annual ACM Symposium on the Theory of Computing*, pages 73–85, 1989.

[Rom90] J. Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proceedings of the 22nd Annual ACM Symposium on the Theory of Computing*, 1990.

[Sch90] C. P. Schnorr. Efficient identification and signatures for smart cards. In *Advances in Cryptology - proceedings of CRYPTO 89*, Lecture Notes in Computer Science, pages 239–252. Springer-Verlag, 1990.

[Sha48] C. E. Shannon. A mathematical theory of communication. *Bell Syst. Tech. J.*, 27:379–423, July 1948.

[Sha79] A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.

[Sim90] G. J. Simmons. How to (really) share a secret. In *Advances in Cryptology - proceedings of CRYPTO 88*, Lecture Notes in Computer Science, pages 390–448. Springer-Verlag, 1990.

[SS77] R. Solovay and V. Strassen. Fast monte-carlo tests for primality. *Siam journal of computing*, pages 84–85, 1977.

[TW86] M. Tompa and H. Woll. How to share a secret with cheaters. In *Advances in Cryptology - proceedings of CRYPTO 86*, Lecture Notes in Computer Science, pages 261–265. Springer-Verlag, 1986.

[TW87] M. Tompa and H. Wall. Random set-reducibility and zero knowledge interactive proofs of possession of information. In *Proceedings of the 28th IEEE Sgmposium on the Foundations of Computes Science*, pages 472–482, 1987.

[Wag79] S. S. Wagstaff Jr. Greatest of the least primes in arithmetic progression having a given modulus. *Mathematics of Computation*, 33(147):1073–1080, July 1979.