Diss. ETH No. 12520

Group Signature Schemes and Payment Systems Based on the Discrete Logarithm Problem

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY
ZÜRICH

for the degree of
Doctor of Technical Sciences

presented by

JAN LEONHARD CAMENISCH
Dipl. El.-Ing. ETH

born 25. April, 1968
citizen of Chur

accepted on the recommendation of

Prof. Dr. Ueli Maurer, referee
Prof. Dr. Ivan Bjerre Damgård, co-referee

1998

Leer - Vide - Empty

to Isabelle

Leer - Vide - Empty

Abstract

The security of many cryptographic systems relies on the difficulty of computing discrete logarithms in certain finite groups.

This dissertation studies existing cryptographic protocols which are based on this problem. These protocols are unified and extended to a framework for designing cryptographic systems. Using this framework, new and efficient realizations of digital group signature schemes and digital payment systems are developed.

Group signature schemes allow a member of a group to sign messages anonymously on the group's behalf. In the case of later dispute, a designated group manager can reveal the signer's identity. An efficient realization of this concept is proposed. Furthermore, the concept of generalized group signatures is developed and realized. This type of scheme allows the definition of sets of group members which can jointly sign on the group's behalf.

Anonymous digital payment systems allow a customer to pay digitally and anonymously. Unfortunately, anonymity also opens the path to criminal misuse, for instance to launder money. As a compromise between the protection of privacy and the possibility of surveillance for crime inspection, the concept of revocable anonymity has been proposed. It introduces a trustworthy third-party which can reveal the identity of a payer in cases of misuse. From an operational point of view, it can be an important requirement that this third-party is not involved in ordinary transactions, but only in anonymity revocation. In this work we present an efficient anonymous digital payment systems satisfying this requirement.

Leer - Vide - Empty

Zusammenfassung

Die Sicherheit vieler kryptographischer Systeme beruht auf der Schwierigkeit, diskrete Logarithmen in endlichen Gruppen zu berechnen.

Diese Dissertation untersucht existierende kryptographische Protokolle, welche auf diesem Problem aufbauen. Diese Protokolle werden vereinheitlicht und zu einem Baukasten für kryptographische Systeme erweitert. In einem zweiten Teil werden damit neue und effizientere Realisierungen digitaler Gruppenunterschriften und anonymer digitaler Zahlungssysteme entwickelt.

Gruppenunterschriften erlauben es Mitgliedern einer Gruppe Dokumente anonym im Namen der Gruppe zu unterschreiben. Im Falle eines Missbrauchs kann jedoch ein Gruppenmanager die Identität des Mitglieds eruieren, welches die Unterschrift geleistet hat. Es wird eine effiziente Realisierung dieses Konzepts entwickelt. Weiter wird das Konzept der verallgemeinerten Gruppenunterschrift entworfen und realisiert. Dieses erlaubt Mengen von Gruppenmitgliedern zu definieren, so dass nur Gruppenmitglieder, die eine solche Menge bilden, gemeinsam im Namen der Gruppe unterschreiben können.

Anonyme digitale Zahlungssysteme ermöglichen es Kunden, digital und anonym zu bezahlen. Da die Anonymität aber auch missbraucht werden kann, um zum Beispiel Geld zu waschen, wurde die aufhebbare Anonymität als Kompromiss zwischen Privatsphärenschutz und Verbrechensbekämpfung vorgeschlagen. In einem solchen System gibt es eine vertrauenswürdige Instanz, welche in Fällen des Missbrauchs gezielt die Anonymität eines Kunden aufheben kann. Von einem operationellen Standpunkt aus gesehen kann es wichtig sein, dass diese vertrauenswürdige Instanz nur für das Aufdecken der Anonymität ak-

tiv sein muss. In dieser Arbeit wird ein solches System präsentiert.

Acknowledgements

First of all, I am profoundly grateful to Ueli Maurer, who gave me the opportunity to work in cryptography and always supported me. I thank Ivan Damgård for his interest in this work and for his insightful comments.

Jean-Marc Piveteau and Markus Stadler guided me into the fascinating field of cryptography and the basis for this thesis lies in the joint work with them. Many of the results reported herein are the fruits of lively discussions with Markus Stadler.

It has been a pleasure to share an office with Christian Cachin and discuss UNIX, cryptography, and such things as 42. He also provided the practical basis for this thesis: a set of LaTeX-macros. I also had the opportunity to share an office with Stefan Wolf and discussing fatzkes, nölding, and tools.

I'm indebted to Ronald Cramer for his valuable critics and for "being an asshole for some minutes". Markus Michels suffered for me and read an early version of Chapter 3. Chris Green did a marvelous job proof-reading this writing. Sandra Baumer volunteered to proof-read Chapter 4.

I thank the not-yet-mentioned former and current members of the "Information security and cryptography group", Masayuki Abe, Dani Bleichenbacher, Matthias Fitzi, Martin Hirt, Reto Kohlas, and Thomas Kühne. We always had a great and relaxing atmosphere. Keep going!

Funds for this work were provided by the Swiss Commission for Technology and Innovation (KTI) by grants No. 2724.1 and 3179.1, and by the Union Bank of Switzerland.

Last, but certainly not least, I am grateful to my parents and grandparents for their support and sending me on the right way.

Jan Camenisch, February 1998

Contents

Chapter 1

Introduction

In their seminal 1976 paper "New Directions in Cryptography" [DH76], Diffie and Hellman devised the concept of public key cryptography and showed that secret communication is possible without a prior exchange of a secret key, as was necessary previously. Their ingenious idea was to use two different keys, a public key for encryption and a private key for decryption. Based on this asymmetry, they further devised the concept of digital signatures. Here, the private key is used to sign a message and the public key is used to verify a signature. However, Diffie and Hellman did not provide realizations of the new concepts, but they proposed a protocol that allows two entities to derive a common secret key only by exchanging information in public. This protocol is based on the difficulty of computing discrete logarithms in a certain finite group.

The concept of public key cryptography inspired many researchers, and it soon became a fast-growing and fascinating research discipline. In the following years, many realizations of digital signature schemes and public key encryption schemes were proposed, most notably the ones by Rivest, Shamir, and Adleman [RSA78] and by ElGamal [ElG85a]. Based on these primitives, more complex systems such as digital payment schemes or voting schemes were devised.

This dissertation is concerned with cryptographic protocols based on the difficulty of computing discrete logarithms in finite groups. Parts of this thesis have already appeared in [CMS96, Cam97, CMS97, CS97a, CS97b].

Chapter 2 provides the mathematical and cryptographical background. This includes problems from number theory, such as computing discrete logarithms or factoring large integers, that are underlying our protocols. An introduction to public key cryptography and its concepts is given, and basic realizations thereof are described. The concept of zero-knowledge proofs of knowledge, an important tool for characterizing properties of cryptographic protocols, is summarized. The chapter ends by introducing hash functions and secret sharing schemes.

In Chapter 3 a framework for designing cryptosystems based on the hardness of computing discrete logarithms is developed. We summarize and unify known building blocks such as protocols for proving the knowledge of the discrete logarithm of a given public key. A new method for proving that the discrete logarithms of public keys satisfy a given set of modular relations completes this chapter.

The remaining chapters describe realizations of two cryptographic concepts: group signature schemes and digital payment schemes. A group signature scheme allows a member of a group of entities to digitally sign messages anonymously on behalf of the group. In case of a later dispute a designated group manager can revoke the anonymity and identify the originator of a signature.

Chapter 4 states our model of group signature schemes, which is a generalization of the models found in the literature in that it allows the definition of coalitions of group members, called authorized coalitions, that are able to sign on behalf of the group. More precisely, only group members who form an authorized coalition are jointly able to sign on the group's behalf. The chapter presents the first realization of a generalized group signature scheme together with a realization of an ordinary group signature scheme that is more efficient than previously proposed schemes.

The group signature schemes described in Chapter 4, as well as the schemes found in the literature, have the property that the size of the group's public key and/or the length of signatures are linear in the number of group members. Hence, these scheme are impractical for large groups. Chapter 5 proposes a new approach to realize group signature schemes that overcome this problem, i.e., in this approach the size of the group's public key as well as the length of signatures are independent of the group's size. The idea underlying this new way of realizing group signature schemes is that group members are given a certificate by the group manager stating that they belong to the group. To sign a message on the group's behalf, a group member must prove knowledge of a certificate, but does not have to present the certificate itself.

Chapter 6 considers anonymous payment schemes. Such schemes allow digital payments in a way that protects the payer's privacy, i.e., the bank does not know which payer transferred money to which payee. However, anonymity can be misused by criminals, for instance for money laundering. As a remedy, the concept of conditional anonymity has been proposed [BGK95, SPC95]. It introduces a trustee as a trusted third party that is capable, in cooperation with the bank, to revoke the anonymity of a payment and to identify the payer. This chapter presents digital payment systems with revocable anonymity. Unlike other such systems that have been previously proposed, the trustee is only involved in the act of anonymity-revocation but not in other transactions such as payments.

What group signature schemes and payment systems with revocable anonymity have in common is that there is a trusted third party that is able to revoke the anonymity of other parties. This party is trusted not to reveal identities at will. To reduce the risk of fraudulent anonymity-revocation, the revocation capability can be distributed among several entities such that only designated subset of them can jointly reveal identities. Chapter 7 discusses several methods to achieve this.

Leer - Vide - Empty

Chapter 2

Foundations and Basic Protocols

This chapter provides an introduction to the topics of algebra, number theory, and cryptography that will be used in the subsequent chapters. Readers familiar with these topic may skip this chapter. References for these topics include the books by Koblitz [Kob94], Kranakis [Kra86], Menezes et al. [MvOV97], Schneier [Sch96], and Stinson [Sti95]. Readers interested in the theory of cryptography will find the manuscript entitled "Foundations of Cryptography" by Goldreich [Gol95] helpful. A reference for complexity theory is the book of Papadimitriou [Pap94].

2.1 Preliminaries

2.1.1 Complexity Theory

An algorithm is a computational procedure that takes a (variable) input and halts uttering an output. Often, the model of a Turing machine is used to make the notion of an algorithm precise. In complexity theory, problems are often classified by the most efficient known algorithm for solving them. The efficiency of an algorithm is measured with respect to the resources required to solve the problem (i.e., primitive steps and memory or the number of gates).

To compare running times of algorithms, the standard asymptotic notation is used. The expression $f(n) = O(g(n))$ denotes an asymptotic upper-bound on $f(n)$ imposed by $g(n)$ and means that there exists *some* positive constant c and a positive integer n_0 such that $0 \leq f(n) \leq c\,g(n)$ for all $n \geq n_0$. Intuitively, this means that f grows no faster asymptotically than g. Furthermore, if $g(n) = O(f(n))$ holds, then we write $f(n) = \Omega(g(n))$, and if both $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$ hold, we write $f(n) = \Theta(g(n))$. An upper-bound that is not asymptotically tight, denoted with $f(n) = o(g(n))$, is the following: For *any* positive constant c, there exists an integer n_0 such that $0 \leq f(n) < c\,g(n)$ for all $n \geq n_0$. The expression $o(1)$ is often used to denote a term $f(n)$ with $\lim_{n\to\infty} f(n) = 0$.

A polynomial-time algorithm is an algorithm that has a worst-case running time of $O(n^k)$ for some constant k, where n is the size of the input. Exponential-time algorithms have a worst-case running time that can be upper-bounded by $O(c^n)$ for some $c > 1$. Sub-exponential time algorithms are those whose running time is upper-bounded by $O\big(\exp(c + o(1)n^{\alpha}(\ln n)^{1-\alpha})\big)$, where c is a positive constant, and α is a constant satisfying $0 < \alpha < 1$. Observe that for $\alpha = 0$ the running time is polynomial, while for $\alpha = 1$ it is fully exponential.

Computational problems are often modeled as decision problems: decide whether a given $x \in \{0,1\}^*$ belongs to a language $L \subseteq \{0,1\}^*$. **P** is the class of languages for which this can be decided in polynomial time. **NP** is the class of problems for which the decision whether x belongs to L can be verified in polynomial time when provided a certificate (or witness) of this fact. Clearly $\mathbf{P} \subseteq \mathbf{NP}$.

Let $R \subseteq \{0,1\}^* \times \{0,1\}^*$ be a boolean relation. We say that R is *polynomially bounded* if there exists a polynomial $p(\cdot)$ such that $|w| \leq p(|x|)$ holds for all $(x,w) \in R$. Furthermore, R is an **NP**-relation if it is polynomially bounded and if there exists a polynomial-time algorithm for deciding membership of pairs (x,w) in R. Finally,

$$L_R = \{x \mid \exists\, w \text{ such that } (x,w) \in R\}$$

is the language defined by R.

Definition 2.1. *A language L is in* **NP** *if there exists an* **NP***-relation $R_L \subseteq \{0,1\}^* \times \{0,1\}^*$ such that $x \in L$ if and only if there exists a w such that $(x,w) \in R_L$. Such a w is called a* witness *of the membership of x in L. The set of all witnesses of x is denoted as $R_L(x)$.*

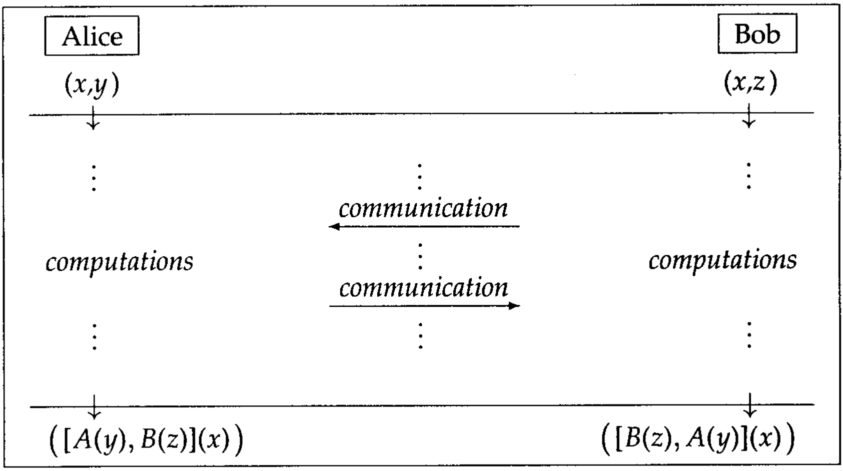

Figure 2.1: An example for the notation of protocols.

2.1.2 Interactive Protocols

An interactive protocol can be seen as a game between two players, say Alice and Bob. The parties send messages back and forth and perform some computation as prescribed by the specification of the protocol. Eventually the protocol finishes and each player obtains a (possibly different) output.

Definition 2.2. *An* interactive protocol *is a pair of algorithms* (A, B) *for two communicating players Alice and Bob. The players' outputs are denoted* $[A(y), B(z)](x)$ *and* $[B(z), A(y)](x)$*, respectively, where* x *denotes their common input and* y *and* z *for their respective private inputs. Alice's view of a protocol with Bob consists of the entire list of parameters Alice "sees" during the execution of the protocol and is denoted* $\langle A(y), B(z)\rangle(x)$ *. This includes all communicated values, Alice's inputs and outputs, as well as all computations and random choices made by her.*

Often, the communicating parties are assumed to be two deterministic Turing machines that have an input tape, an output tape, a random tape, a write-only communication tape, and a read-only communication tape. The read-only communication tape of one machine is the write-only communication tape of the other machine.

We now introduce some notation that will later be used when analyzing properties of a protocol. Let (A, B) be an interactive protocol. Then $\widetilde{B}$ denotes an interactive algorithm that a (dishonest) player Bill could run instead of B. The only thing that the algorithms B and $\widetilde{B}$ have in common is that they properly interact with A. To analyze the properties of Alice that manifest themselves though the interaction with Alice running A, one often uses a third party. This party, called Master, interacts with Alice, but, in contrast to Bob and Bill, has the ability to reset and restart Alice at will. This type of interaction is called *oracle access* to Alice, if an interaction with Alice is counted as a single step for Master, and is called *black-box access* to Alice, when the running time of Alice counts also as running time of Master. With $M^{A(x)}(y)$ we denote Master's output when running M on input y and is interacting with Alice running A on input x.

Next, we explain our notation for illustrating protocols (see Figure 2.1). The players' names are indicated in boxes on the first line and their lists of inputs are shown in brackets on the next line. The players' computations and their communication is shown between the two horizontal lines. Their lists of outputs in an honest execution of the protocol are shown in brackets on the bottom line. Dishonest players are not restricted to store only their specified output, but are assumed to store their entire view. Whenever the protocol specifies that a player must verify a condition it is assumed that, if the verification fails, the protocol is stopped and all parties are informed.

2.1.3 Miscellaneous Notations

Let $\mathcal{H}(\cdot)$ denote a hash function that maps binary strings of arbitrary length to binary strings of a fixed length, i.e., $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\ell$. For a definition of a cryptographic hash function we refer to Section 2.10. Throughout this thesis we assume a standard binary representation of integers, elements of algebraic groups, and text strings. For instance, if a is a group element, then $b := \mathcal{H}(a)$ means that $\mathcal{H}$ is applied to the binary representation of a. If b is used later in some algebraic expression, we assume that it is first recovered from the binary string $\mathcal{H}(a)$. Furthermore, $a \| b$ denotes the concatenation of the binary strings representing a and b.

By $c[i]$ we denote the i-th bit of a string c counting from the right-hand end. The term $(c_i)_{i \in S}$ denotes the ordered list of c_i's for which i is in some set S, i.e., if $S = \{5, 1, 6, 2\}$ then $(c_i)_{i \in S} = (c_1, c_2, c_5, c_6)$. The term $\mathbf{pr}_i(\cdot)$ denotes the first projection of a tuple, i.e., $\mathbf{pr}_i((x_1, \ldots, x_n)) = x_i$.

The expression $\xi \in_R X$ means that ξ is randomly chosen from the (finite) set X according to the uniform distribution.

The logarithms of x to the bases e and 10 are denoted by $\ln(x)$ and $\lg(x)$, respectively. Moreover, when not stated otherwise, $\log(x)$ denotes the logarithm of x to the base 2.

2.2 Algebra and Number Theory

While for a long time number theory was considered a pure theoretical science with no applications, today it plays an important role in cryptography. Most public-key cryptosystems are based on problems from number theory. In this chapter we describe some facts from algebra and problems found in number theory that are instrumental to (public-key) cryptography.

2.2.1 Groups

Let S be a nonempty set and $*$ be a *binary operation* that maps $S \times S$ to S, thus $a * b$ denotes the result of $*$ applied to the elements $a, b \in S$. The operation $*$ is called *commutative* if $a * b = b * a$ holds for all $a, b \in S$, and *associative* if we have $(a * b) * c = a * (b * c)$ for all $a, b, c \in S$. An element $e \in S$ is called an *identity element* if for all $a \in S$, $e * a = a * e = a$ holds. An *inverse* of an element $a \in S$ is an element $b \in S$ such that $a * b = b * a = e$ holds, provided that e exists.

Definition 2.3. *A* group *is a set G together with an associative binary operation $*$ on elements of G such that G contains an identity element for $*$ and every element has an inverse under $*$. If $*$ is commutative, the group is called* abelian *or commutative. Often, a group is denoted by $\langle G, * \rangle$ or simply by G. A group G is called* finite *if $|G|$ is finite. The number of elements of a finite group is called its* order.

It can easily be seen that in a group the identity element is unique, as is the inverse of any element. If the operation is called addition, the identity element is denoted as 0 and the inverse element of a as $-a$. If the operation is called multiplication, the identity element is denoted as 1 and the inverse of an element a as $1/a$ or a^{-1}. Subsequently, the multiplicative notation is used when dealing with arbitrary groups. So a^m means that a is multiplied m-times by itself, and a^{-m} denotes $(1/a)^m$.

A group G is called *cyclic* if there exists an element $a \in G$ such that every element $b \in G$ can be written in the form a^x for some $x \in \mathbb{Z}$. Such an element a of G is called a *generator* of G, and one writes $\langle a \rangle = G$ to indicate that a generates G. The *order* of an element b, denoted $\text{ord}(b)$, is the smallest positive integer n such that $b^n = 1$. The order of any element of a finite group divides the order of the group. Furthermore, if a is a generator of the cyclic group of order m, then the element $b = a^i$ has order $m/\gcd(m, i)$. In particular, b is a generator of G if and only if $\gcd(m, i) = 1$. Hence if m is prime, every element different from 1 is a generator of G.

A subset $H \subseteq G$ is called a subgroup of a group G if it is a group in its own right under the operation of G. In particular, H contains 1, and if $a, b \in H$ then $ab, a^{-1} \in H$. Furthermore, $|H|$ divides $|G|$. An important class of subgroups of G are the groups generated by an element a of G, denoted $\langle a \rangle$. The order of $\langle a \rangle$ equals the order of a. Hence, the order of any group element divides the order of the group.

2.2.2 The Groups $\mathbb{Z}_m$ and $\mathbb{Z}_m^*$

The set of the integers modulo m, denoted as $\mathbb{Z}_m$, together with addition modulo m constitutes an abelian group of order m. Another important group is $\mathbb{Z}_m^*$, formed by the positive integers smaller than m and relatively prime to m together with the multiplication modulo m. The order of $\mathbb{Z}_m^*$ is given by the Euler totient function $\varphi(m)$.

Definition 2.4. *Let n be a positive integer. The Euler φ-function is defined as the number of nonnegative integers k less than n which are relatively prime to n:*

$$\varphi(n) = |\{k \mid 1 \leq k < n \textit{ and } \gcd(k, n) = 1\}|.$$

For an integer $n = \prod_{i=1}^{k} p_i^{\alpha_i}$, where the p_i's are distinct primes, we have

$$\varphi(n) = n \prod_{i=1}^{k} (1 - p_i^{-1}).$$

The group $\mathbb{Z}_m$ is cyclic for all m, whereas $\mathbb{Z}_m^*$ is cyclic if and only if m is 2, 4, or a power of an odd prime.

2.2.3 Efficiency of Group Operations in $\mathbb{Z}_m$

To measure the efficiency of cryptosystems built on an algebraic group, it is necessary to know how many resources an operation on group elements takes. In this subsection we will consider only operations that can be carried out fast, i.e., in time polynomial in the size of group-elements. These are addition, multiplication, exponentiation, and the computation of inverses. More information is found in the books of Knuth [Knu81], Cohen [Coh93], and Cormen et al. [CLR92]

Let m be an integer. Addition modulo m of two elements of $\mathbb{Z}_m$ takes $O(\log(m))$ time (bit-operations). The additive inverse in $\mathbb{Z}_m$ can also be computed in linear time.

The modular multiplication of two elements of $\mathbb{Z}_m^*$ takes $O(\log^2(m))$ time. The inverse of an element a can also be computed in $O(\log^2(m))$ time using the extended Euclidean algorithm. The algorithm calculates two integers u, v such that $ua + vm = \gcd(a, m)$. By definition of $\mathbb{Z}_m^*$ we have $ua + vm = 1$ and thus $u \equiv a^{-1} \pmod{m}$.

Exponentiation of a group element a with an integer $n < |G|$ means to apply the group operation n times on a with itself, i.e., to compute $b = a^n$. The naive method requires $n - 1$ group operations. Using the so-called *square and multiply* method, exponentiation can be performed with a maximum of only $2\log(n)$ group operations in any group. The idea is as follows: represent n in binary form

$$n = \sum_{i=0}^{k} \alpha_i 2^i$$

with $\alpha_i \in \{0, 1\}$ and $k = \lfloor \log(n) \rfloor$. Then apply the following algorithm:

```
$b := 1$
for $i = k$ downto $0$ do
  $b := b \cdot b$
  if $\alpha_i = 1$ then $b := b \cdot a$ fi
od
```

Hence, an exponentiation in $\mathbb{Z}_m^*$ can be achieved in $O(\log^3(m))$ time.

2.3 Number-Theoretic Problems

The security of many cryptosystems relies on the intractability of solving some (number theoretic) problems such as factoring a large integer. Basically, a problem is intractable if there is no algorithm that solves the problem using a reasonable amount of resources (time and/or memory). Most often, it cannot be proved that no such algorithm exists, but one rather has to assume that no such algorithm exists because nobody has found one. We still have to specify what a reasonable amount of time means. In theory, this means that the resources needed by the best algorithm are at least exponential in the number of bits needed to describe the problem. In practice, this means that the fastest computer should not be able to find a solution for a specific instance of the problem within years (centuries, lifetime of the universe etc.).

In the following, we present only those problems that are used later on in this thesis. For further information and more details we refer to [MvOV97, Coh93].

2.3.1 The Discrete Logarithm Problem

The difficulty of computing discrete logarithms is the basic problem underlying the results of this thesis. We will formally state this problem and give a short overview of existing algorithms for solving it. For a more extensive survey on the discrete logarithm problem and the state of the art in solving it, as well as for references for the various algorithms, see [McC90a, Odl94].

Definition 2.5. *Let G be a finite cyclic group and $g \in G$ be a generator of G. The* discrete logarithm *of some element $a \in G$, denoted* $\log_g(a)$*, is the unique* *integer $x, 0 \leq x < |G|$, such that $a = g^x$.*

Often, the discrete logarithm is also called index of an element a. If g is not a generator, the notion of the discrete logarithm of a to the base g is extended to be the smallest integer x, such that $a = g^x$, if it exists.

The following facts, known from ordinary logarithms, also hold for discrete logarithms. Let $G = \langle g \rangle$ be a cyclic group of order n and a, b, and c elements of G. Then we have $\log_g(ab) \equiv \log_g(a) + \log_g(b) \pmod{n}$ and $\log_g(a^x) \equiv x \log_g(a) \pmod{n}$ for any integer x. Furthermore, if h is also a generator of G, then $\log_g(a) \equiv \log_h(a) \log_h(g)^{-1} \pmod{n}$. The last equation includes the interesting special case $\log_g(h) \equiv \log_h(g)^{-1} \pmod{n}$.

Definition 2.6. *The* discrete logarithm problem *(DLP) is the following: given a finite cyclic group G, a generator g of G, and an element a, find the integer $x, 0 \leq x \leq |G| - 1$, such that $a = g^x$ holds.*

In the following we give a brief overview of algorithms to solve the DLP in a cyclic group $G = \langle g \rangle$. They can be categorized by the kind of representations of the group elements they work with:

I. Generic algorithms that work in arbitrary groups. These include Pollard's rho algorithm [Pol78] and the Baby-Step/Giant-Step algorithm.

II. Algorithms which work in arbitrary groups but are especially efficient if the group's order is *smooth,* i.e., has only small prime factors. An example is the Pohling-Hellman algorithm [PH78].

III. Special algorithms that exploit the representation of the group elements and thus work only in the group they were designed for. The index calculus algorithm for $\mathbb{Z}_p^*$ is an example (e.g. [COS86]).

Let us first discuss generic algorithms. The naive algorithm is of course exhaustive search, i.e., computing the successive powers $g^0, g^1, g^2, \ldots$ until the element a is obtained. This requires $O(n)$ group operations. A more efficient algorithm is the so called Baby-Step/Giant-Step algorithm attributed to Shanks [Knu81]. It computes discrete logarithms in $O(\sqrt{n} \log(n))$ group operations and needs storage for $\sqrt{n}$ group elements. Pollard's rho algorithm has the same time complexity but needs only negligible storage.

If the group order $n = p_1^{e_1} \cdot \ldots \cdot p_l^{e_l}$ has only small prime factors, it is favorable to use the Polling-Hellman algorithm. It requires only $O(\sum_{i=1}^{r} e_i(\lg n + \sqrt{p_i}))$ group operations if the factorization of n is given (see also Section 2.3.4). If n is prime then the Polling-Hellman algorithm degenerates to the Baby-Step/Giant-Step.

The algorithms considered so far are all exponential. However, for the groups $\mathbb{Z}_p^*$ and $\mathbb{Z}_{2^m}^*$, where p is a prime, there exist sub-exponential algorithms. The index calculus algorithms have a running time that is upper-bounded by $O(\exp((c + o(1))\sqrt{\ln q \ln\ln q}))$ group operations, where q is p or 2^m and $c > 0$ is some constant. For $\mathbb{Z}_p^*$, one of the most efficient algorithms is the number field sieve. It needs $O(\exp((1.92 + o(1))(\ln p)^{1/3}(\ln\ln p)^{2/3}))$ group operations to compute a discrete logarithm.

We have seen that the hardness of the DLP depends strongly on the representation of the elements of the group considered. This is not surprising, since in some groups the DLP is rather easy to solve: For instance, in $\langle \mathbb{Z}_m, + \rangle$ computing discrete logarithm of an element a to the base g means solving the equation $gx \equiv a \pmod{m}$ for x. This basically involves the computation of $\gcd(g, m)$, which takes $O(\log^2(m))$ group operations using the extended Euclidean algorithm. Hence, computing discrete logarithms in an arbitrary cyclic group G is in essence finding an isomorphism between G and $\langle \mathbb{Z}_{|G|}, + \rangle$ that can be evaluated efficiently.

Shoup [Sho97] showed that there exists a lower bound on the computational complexity of the DLP for generic algorithms that do not exploit any special properties of the encoding of the group elements (type I and II). More precisely, Shoup showed that any algorithm that solves the DLP must perform at least $\Omega(\sqrt{p})$ group operations, where p is the largest prime dividing the order of the group. If the group order is prime, this lower bound is for instance met by Pollard's rho algorithm.

Together with the rho algorithm, Pollard [Pol78] proposed also a so-called lambda algorithm that computes discrete logarithms with a parameter-dependent success probability in $O(\sqrt{w})$ group operations when the solution lies within a restricted interval of width w.

2.3.2 The Representation Problem

The representation problem is a generalization of the discrete logarithm problem with respect to the number of bases.

Definition 2.7. *Let G be a finite cyclic group of order n and let the elements $g_1, \dots, g_m \in G$ be distinct generators of G. A* representation *of some element $a \in G$ is an m-tuple $(x_1, \dots, x_m)$, $0 \leq x_i \leq n-1$ for all $1 \leq i \leq m$, such that*

$$a = \prod_{i=1}^{m} g_i^{x_i}.$$

A representation is also called an *index tuple.* For any element $a \in G$ there exist exactly n^{m-1} m-tuples representing a with respect to $g_1, \dots, g_m$.

Definition 2.8. *The* representation problem *(RP) is the following: given a finite cyclic group G, a generator-tuple $g_1, \dots, g_m$, and an element a, find integers $x_1, \dots, x_m$, $0 \leq x_i \leq n-1$, such that $a = \prod_{i=1}^{m} g_i^{x_i}$.*

The representation problem is a generalization of the DLP. Moreover, if the generators $g_1, \dots, g_m$ are chosen randomly, it is as hard as the DLP to find two different representations of an element. For a thorough discussion of the representation problem we refer to [Bra93].

2.3.3 The Diffie-Hellman Problem

The Diffie-Hellman Problem is closely related to the discrete logarithm problem.

Definition 2.9. *The* Diffie-Hellman problem *(DHP) is the following: given a finite cyclic group G, a generator g of G, and the two elements g^u and g^v, find the element g^{uv}.*

It is obvious that the DHP can be solved in polynomial time when the DLP can be solved in polynomial time by first computing $u = \log_g(g^u)$ and then $(g^v)^u$. For some groups, the DLP and the DHP were shown to be computationally equivalent [dB90, Mau94, MW96].

Definition 2.10. *The* Decision Diffie-Hellman problem *(DDHP) is the following: given a finite cyclic group G, a generator g of G, and the three elements g^u, g^v, and g^w, decide whether the elements g^w and g^{uv} are equal.*

The Decision Diffie-Hellman problem was first mentioned in [Bra93], although there are earlier cryptographic systems that implicitly rely on the hardness of this problem (e.g. [Cv90, Cha91]).

Clearly, an efficient algorithm to solve the DHP implies one for the DDHP. Generally, the DDHP is assumed to be intractable, but not much is known about it.

Shoup [Sho97] proved that the lower bound on the complexity of generic algorithms to solve these two problems is $\Omega(\sqrt{p})$, where p stands for the largest prime divisor of the group order in the case of the DHP, and for the smallest prime divisor of the group order in the case of the DDHP. However, it can be shown that in this model the DLP and the DHP are not computationally equivalent [MW].

2.3.4 Factoring Large Integers

The integer factorization problem is the problem underlying the probably widest known public-key cryptosystem.

Definition 2.11. *The* integer factorization problem *(FACTORING) is the following: given a positive integer n, find its prime factorization, i.e., find pairwise distinct primes p_i and positive integers e_i such that $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$.*

The algorithms for factoring an integer n can be divided into two types.

I. General purpose algorithms. Their running time depends only on the size of n. Examples are the quadratic sieve and the general number field sieve.

II. Special purpose algorithms. Their running time depends on a special property of n, such as the size of the largest prime factor. Trial division, Pollard's rho algorithm, Pollard's $p-1$, and the elliptic curve algorithm fall in this category.

The most obvious algorithm for factoring is trial division. In the worst case, all primes smaller than $\sqrt{n}$ must be tried for completely factoring n. A more efficient, but still exponential algorithm is Pollard's rho [Pol75]. It has running time $O(\sqrt{n})$. If n has a small prime factor, say p, the elliptic curve method finds it in $O(\exp((1+o(1))\sqrt{2\ln p \ln\ln p}))$ time. Other sub-exponential algorithms are the quadratic sieve algorithm introduced in [Pom85] which has a running time of $O(\exp((1+o(1))\sqrt{\ln n \ln\ln n}))$ and the number field sieve algorithm [LLJ93] having a time-complexity of $O(\exp((1.92+o(1))(\ln n)^{1/3}(\ln\ln n)^{2/3}))$. The latter was used to factor RSA-130 [CDEH+96]. When the smallest prime of n is about the size of $\sqrt{n}$, the elliptic curve and quadratic sieve have the same asymptotic running time. In practice, however, the quadratic sieve is faster in this case.

In contrast to the situation with the discrete logarithm, it is not clear which kind of algorithm is best when only given the integer n. Of course, one wants to apply one of the special purpose algorithms whenever possible. In [MvOV97] it is proposed to apply all the different types of algorithms in the following order:

1. Trial division by small primes up to some bound b_1.
2. Pollard's rho, hoping to find any small factors smaller than some bound $b_2 > b_1$.
3. An elliptic curve factoring algorithm, hoping to find any small factors smaller than some bound $b_3 > b_2$.
4. Finally, applying one of the general-purpose algorithms.

2.3.5 Computing Square-Roots and *e*-th Roots

Computing e-th roots over the integers modulo a composite is the problem the RSA cryptosystem [RSA78] is based on.

Definition 2.12. *Let G be a group and $e < |G|$ be an integer. An element $b \in G$ is called an e-th root of an element $a \in G$ if we have*

$$b^e = a\ .$$

If $\gcd(e, |G|) = 1$ holds, then an e-th root always exists and is unique. Computing e-th roots is feasible if $|G|$ is known: it can be solved by computing e^{-1} in $\mathbb{Z}^*_{|G|}$ and thus obtaining $b = a^{e^{-1}}$.

Definition 2.13. *The* e-th root problem *(ERP) is the following: given a group G of unknown order, a positive integer $e < |G|$ and an element $a \in G$, find an element $b \in G$ such that $b^e = a$.*

If $G = \mathbb{Z}_n^*$, with n being the product two primes p and q, and the condition that $b \in G$ is replaced by $b \in \mathbb{Z}_n$, we get the *RSA problem* (RSAP). In this case the order of the group can be found by factoring n. Thus, if FACTORING is easy then so is the RSAP. Whether the converse is also true is not known.

Definition 2.14. *Let n be the product of two primes p and q. Then an element $a \in \mathbb{Z}_n$ is a* quadratic residue *modulo n (or a square) if there exists an integer w such that*

$$w^2 \equiv a \pmod{n}.$$

If there exist no such $w \in \mathbb{Z}_n$, a is called a quadratic non-residue. *The set of all quadratic residues and the set of all non-residues in $\mathbb{Z}_n^*$ are denoted by QR_n and QNR_n, respectively.*

If the factorization of the modulus is known, then there exists an efficient algorithm to decide whether an element is in QR_n. For a modulus with unknown factorization this is believed to be as hard as factoring, but this has not been proved.

Definition 2.15. *The* quadratic residuosity problem *(QRP) is the following: Given integers n and a, $0 \le a < n$, decide whether there exists an integer w, $0 \le w < n$, such that*

$$w^2 \equiv a \pmod{n}.$$

The Legendre symbol is useful for keeping track of whether or not an integer a is a quadratic residue modulo a prime p.

Definition 2.16. *Let p be an odd prime and a an integer. The* Legendre symbol $\left(\frac{a}{p}\right)$ *is defined to be*

$$\left(\frac{a}{p}\right) = \begin{cases} 0, & \text{if } p|a \\ 1, & \text{if } a \in QR_n \\ -1, & \text{if } a \in QNR_n \end{cases}$$

The Jacobi symbol is a generalization of the Legendre symbol to integers n which are not necessarily odd primes.

Definition 2.17. *Let $n \geq 3$ be an integer with prime factorization $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$ and let a be an integer. The* Jacobi symbol $\left(\frac{a}{n}\right)$ *is defined to be*

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{e_1} \left(\frac{a}{p_2}\right)^{e_2} \cdots \left(\frac{a}{p_k}\right)^{e_k} .$$

2.4 Public Key Cryptography

Encryption schemes can be divided into two categories: *private key* or symmetric schemes and *public key* or asymmetric schemes. In a private key scheme, the same key is used for encryption and decryption. Hence, when two parties want to securely communicate with a symmetric encryption scheme, they need to exchange a private key in advance. In a public key scheme, a different key is used for encryption and decryption. The key used for encryption, the *public key*, can be published, while the *secret key* used for decryption must be kept secret. This is an advantage: while the key for a symmetric encryption scheme must be exchanged securely, the public keys need to be exchanged only authentically – a much weaker requirement.

While symmetric encryption schemes were already known to Caesar, public key cryptography was only invented in 1977 by Diffie and Hellman with their paper entitled "New Directions in Cryptography" [DH76]. They put forth the notion of *trap-door one-way functions*. These are functions that are easy to compute but hard to invert – unless a trap-door is known. More precisely, the inverse of the function is hard to find given only the function. Given such a function $f : \mathcal{A} \rightarrow \mathcal{B}$, it can be published as public key, while the trap-door (the secret key) remains secret. The encryption of a message $m \in \mathcal{A}$ is then simply $f(m)$, and decryption is $f^{-1}(f(m)) = m$.

Apart from encryption, trap-door one-way functions also make it possible to realize digital signature schemes. For instance, the signature of a message $m \in \mathcal{B}$ can be $s = f^{-1}(m)$. It can then be verified by everybody by deciding whether $m = f(s)$ holds using the public key.

2.5 Public Key Encryption

Definition 2.18. *A* public key encryption *scheme is a triple of algorithms* $(\mathtt{gen}, \mathtt{enc}, \mathtt{dec})$. *The first algorithm is probabilistic, the second one is often probabilistic, and the third one is deterministic. The algorithm* gen *generates a secret key* x_a *and a corresponding public key* y_a *for an entity* A *when input the system parameter. The algorithm* enc *takes* y_a *and the message* m *as input and outputs a ciphertext* c. *On input of a ciphertext* c *and the secret key* x_a *of party* A, dec *outputs the encrypted message* m. *The following must be satisfied for all messages* m *and all key pairs* (x_a, y_a) *output by* gen.

$$\mathtt{dec}(c, x_a) \begin{cases} = m & \textit{if } Prob\big(c = \mathtt{enc}(m, y_a)\big) > 0 \\ \neq m & \textit{otherwise.} \end{cases}$$

If the algorithm enc is probabilistic we call the encryption scheme probabilistic.

The length of messages that can be encrypted is often limited. Furthermore, public key encryption schemes are often slower than symmetric ciphers. To encrypt large messages, public key encryption is therefore used only to encrypt a session key, which is in turn used as a key for a fast symmetric cipher such as DES or IDEA [LM91, Lai92].

To enable a recipient to recognize a decrypted string as a valid message, messages should contain some redundancy. Typically, this is achieved by applying a hash function to the message and appending the result to the message before encryption.

2.5.1 Diffie-Hellman Key Exchange

Apart from founding public-key cryptography, Diffie-Hellman [DH76] also proposed a concrete scheme for obtaining a common private key using an authentic but not secret channel. This private key can then for instance be used for encryption with a symmetric encryption algorithm.

The scheme works as follows: Let G be a finite cyclic group of order q and let $g \in G$ be a generator of G such that computing discrete logarithms in G is infeasible. Furthermore, let $y_A := g^{x_A}$ and $y_B := g^{x_B}$ be the public keys of two parties Alice and Bob and let x_A and x_B be their respective secret keys. To derive a common secret key k, Alice and

Bob exchange their public keys over the authentic channel and raise the partner's public key to the power of their own secret key and thereby get

$$k := y_A^{x_B} = y_B^{x_A} = g^{x_A x_B} ,$$

the common private key.

The security of the scheme is based on the DHP; it is this scheme that gave the problem its name. This protocol can also be used to encrypt messages the discrete logarithm of which is known to the sender. Let $m = g^{\hat{m}}$ be the message that B wants to send privately to A. To do so, B computes $c = y_A^{\hat{m}}$, which A can decrypt upon reception by computing $c^{1/x_A} = m$.

2.5.2 The RSA Encryption Scheme

Rivest, Shamir, and Adleman [RSA78] were the first to propose a concrete realization of a trap-door one-way function as introduced by Diffie and Hellman. It is based on the difficulty of computing e-th roots modulo a composite n, i.e., the RSA problem.

This encryption scheme works as follows. Let p and q be two large primes such that $p-1$ and $q-1$ are not smooth, $n = pq$, and let e be an integer satisfying $\gcd(e, \varphi(n)) = 1$. The public key of a recipient Bob is the pair (n, e) and his secret key is the triple (p, q, d), where d satisfies $de \equiv 1 \pmod{\varphi(n)}$.

To encrypt a message $m \in [0, \ldots, n-1]$ for Bob, a sender Alice computes

$$c := m^e \pmod{n}$$

and sends c to him. Bob can recover m using the secret value d as follows:

$$m := c^d \pmod{n}.$$

The correctness of this encryption method is seen as follows:

$$c^d \equiv (m^e)^d \equiv m \pmod{n}$$

holds since $ed \equiv 1 \pmod{\varphi(n)}$.

The security of the scheme is based on the RSAP. However, there are some pitfalls that can make the system insecure. For instance, when e is chosen small for reasons of efficiency, a number of attacks are possible. Håstad showed that when encrypting the same message for multiple recipients having the same public exponent e but a different modulus, the message can be computed from the cipher-texts without knowing any of the corresponding secret keys [Hås88]. Furthermore, if polynomial relations among the encrypted messages are known, messages can also be recovered [CFPR96]. Such attacks can be prevented by *salting*, i.e., appending a random bit-string to the message before encryption.

2.5.3 The ElGamal Encryption Scheme

The following encryption scheme was proposed by ElGamal [ElG85a, ElG85b]. It can be seen as a special way of using the Diffie-Hellman key exchange protocol. Let G be a finite cyclic group of order q and let $g \in G$ be a generator of G such that computing discrete logarithms in G is infeasible. In the original proposal, G was chosen to be $\mathbb{Z}_p^*$ (thus we have $\mathrm{ord}(g) = p - 1$) where p is a large prime. In order to encrypt a message $m \in G$ for Bob having public key $y = g^x$, Alice first chooses an a randomly in $\mathbb{Z}_q$ and computes the pair $(A, B) = (g^a, y^a m)$ being the encryption of m. Bob, knowing the secret key x, can decrypt the message m by calculating

$$\frac{B}{A^x} = \frac{y^a m}{g^{ax}} = \frac{g^{xa} m}{g^{ax}} = m \; .$$

Alternatively, the role of the public key and the base can be interchanged and we get the following variant which we will also employ later. Now a message can be encrypted by randomly selecting an a in $\mathbb{Z}_q$ and computing the pair $(A, B) = (y^a, g^a m)$. Decryption is performed by calculating

$$\frac{B}{A^{x^{-1}}} = \frac{g^a m}{y^{ax^{-1}}} = \frac{g^a m}{g^{xax^{-1}}} = m \; .$$

In both schemes, the security is based on the assumed intractability of the DHP. Note that these are probabilistic encryption schemes. Furthermore, if a different a was used for every encryption, it is equivalent to the DDHP to decide whether two pairs (A, B) and (A', B') both encrypt the same message m.

2.6 Identification Protocols

An identification protocol allows a prover Peggy to convince a verifier Vic of her identity. Vic is given a public key which he knows belongs to Peggy. Thus, if some entity can prove to him knowledge of the secret key corresponding to Peggy's public key, he can conclude that this entity must be Peggy.

Informally, an identification protocol consists of a probabilistic algorithm $\mathtt{gen}$ and an interactive protocol $(\mathtt{prv}, \mathtt{ver})$ for a *prover* Peggy and a *verifier* Vic. The algorithm $\mathtt{gen}$ generates a secret key x_p and a corresponding public key y_p for Peggy when input the system parameters. The protocol $(\mathtt{prv}, \mathtt{ver})$ must basically satisfy three properties. First, it must be secure for Peggy, i.e., Vic should not be able to learn anything about her secret key. Second, Peggy should always be able to convince Vic of her identity if he is honest. And third, an entity Paula not knowing the secret key should not be able to convince Vic that she is Peggy. These properties can be formalized as zero-knowledge and as a complete and sound proof of knowledge. Since these are important concepts, we will treat them separately in Section 2.9.

2.6.1 The Schnorr Identification Protocol

Schnorr's identification protocol [Sch91] is a so-called three move protocol, i.e., there are three communication steps between Peggy and Vic. This is a property shared with many other protocols we will encounter. The exchanged messages t, r, and s are called commitment, challenge, and response (see Figure 2.2 for an example).

The security of the protocol is based on the assumed intractability of the discrete logarithm problem. The protocol is derived from an initial idea of Chaum et al. [CEvdG88].

The Schnorr protocol allows Peggy to prove that she knows the discrete logarithm of her public key. More generally, this protocol can also be used as a sub-protocol in more complex protocols to prove one's knowledge of the discrete logarithm of an arbitrary group element with respect to an arbitrary base. Such extensions will be treated in detail in Chapter 3. Let us therefore describe the protocol in a more general algebraic setting than in [Sch91].

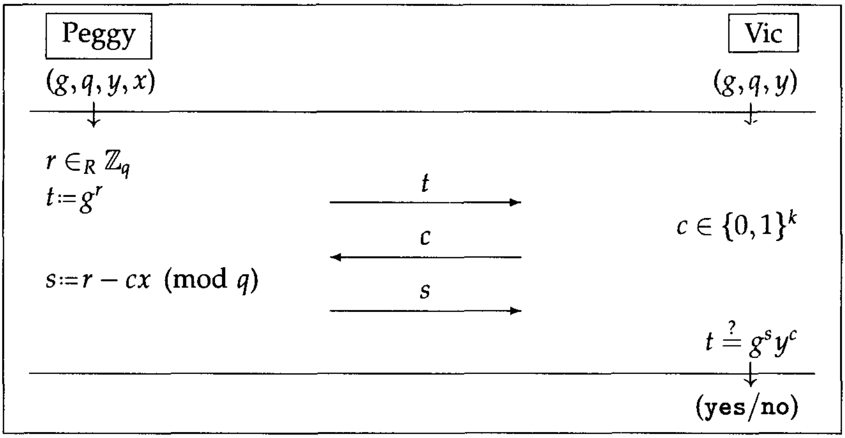

Figure 2.2: The Schnorr identification protocol. The secret key is the discrete logarithm of the public key.

Let G be a finite cyclic group of order q and let $g \in G$ be a generator of G such that computing discrete logarithms in G is infeasible. Let $y = g^x$ be the authentically published public key of Peggy and let x be her secret key. Then, using the protocol depicted in Figure 2.2, Peggy can convince Vic of her identity by proving to him her knowledge of the discrete logarithm of y.

It can easily be seen that Peggy can always convince an honest verifier Vic, i.e., that Peggy can reply to all challenges. In contrast, a cheating prover can only prepare for one of the 2^k possible challenges, and if this fails, try to find a correct s. However, if the cheater can find such an s, he can also compute the discrete logarithm of y to the base g. A protocol with this property is called a *proof of knowledge* (see Section 2.9). Furthermore, it can be argued that the protocol is secure for Peggy, since all the information Vic obtains seems completely random to him. This property is called *honest-verifier zero-knowledge* (see Section 2.9.2). A detailed security analysis of the protocol is found in Section 2.9.4.

2.7 Digital Signature Schemes

This section considers the digital analogue of handwritten signatures. Informally, a digital signature is a binary string that relates a message to the signer's public key. On one hand, everyone must be able to verify a signature when given the public key of the signer and the message. On the other hand, only the signer, i.e., the party that knows the secret key, must be able to compute a signature.

As mentioned earlier, the concept of a digital signature was put forth by Diffie and Hellman in their seminal paper [DH76].

Definition 2.19. *A* digital signature scheme *is a triple of algorithms* $(\mathtt{gen}, \mathtt{sig}, \mathtt{ver})$. *The first is probabilistic, the second one is often probabilistic, while the third one is deterministic. The algorithm* **gen** *generates a secret key x_s and a corresponding public key y_s of a signer S on input of the system parameters. The algorithm* `sig` *takes x_s and a message m as input and outputs a signature σ of m. On input of a message m, a signature σ, and the public key y_s of a signer, the the algorithm* `ver` *outputs* `true` *or* `false`. *The following must be satisfied.*

$$\mathtt{ver}(m, \sigma, y_s) = \begin{cases} \mathtt{true} & \textit{if } Prob\big(\sigma = \mathtt{sig}(m, x_s)\big) > 0 \\ \mathtt{false} & \textit{otherwise} \end{cases}$$

Furthermore, a signature scheme must be unforgeable. This means that is must be infeasible to compute a signature of a message with respect to a public key without knowing the corresponding secret key.

Attacks on signature schemes are often characterized by the kind of forgery they achieve.

total break: An adversary is able to compute the secret key of the signer, or can efficiently compute a signature of any message by some other algorithm than `sign`.

selective forgery: An adversary is able to compute a signature of a particular given message or a special class of messages.

existential forgery: An adversary is able to compute a signature of at least one arbitrary message. The adversary might have no or little control over the message that is signed.

In the last two kinds of attacks, the adversary is allowed to involve the signer. The attack is successful if the adversary manages to compute a signature of a message that the signer did not sign. Goldwasser, Micali and Rivest [GMR88] were the first to propose a signature scheme that is provably secure against existential forgery, the strongest notion of security, assuming only the existence of claw-free trap-door permutations. However, most proposed signature schemes cannot be shown to be secure against all these attacks.

A way to construct a digital signature scheme is to take an identification protocol and replace the challenge by the result of a one-way (hash) function applied to the commitment and the message to be signed. For such constructions, it is often argued that a signature scheme is secure if the underlying identification protocol is a zero-knowledge proof of knowledge and a secure one-way function is used. To give a theoretical foundation to this kind of reasoning the *random oracle model* [BR93] was invented. In this model the hash-function is replaced by an oracle that outputs a random bit-string when queried. When the oracle gets a query that it has gotten before, it answers with the same string as before. In their paper Bellare and Rogaway also provide a signature scheme that is secure in this model.

2.7.1 The RSA Signature Scheme

The RSA signature scheme [RSA78] is directly obtained from the RSA encryption scheme. It is thus based on the assumed intractability of computing e-th roots modulo a composite.

Let p and q be two large primes such that $p-1$ and $q-1$ are not smooth, and e be an integer with $\gcd(e, \varphi(n)) = 1$. A signer's public key is the pair (n, e) and his secret key is the triple (p, q, d), whereby $de \equiv 1 \pmod{\varphi(n)}$ must hold. Finally, let $\mathcal{H}$ be a collision-resistant hash function that maps $\{0,1\}^*$ to $\mathbb{Z}_n$.

Definition 2.20. *An* RSA signature *of a message $m \in \{0,1\}^*$ for the public key (n, e) is a value $s \in \mathbb{Z}_n$ satisfying*

$$s^e \equiv \mathcal{H}(m) \pmod{n}.$$

An RSA signature can be computed by the signer (who knows d) as follows:

$$s := \mathcal{H}(m)^d \pmod{n}.$$

The correctness of this signing procedure follows from the

$$s^e \equiv \mathcal{H}(m)^{de} \equiv \mathcal{H}(m) \pmod{n},$$

since $de \equiv 1 \pmod{\varphi(n)}$.

If the hash function $\mathcal{H}(\cdot)$ were not applied to the message, the product of signatures would result in the signature of the product of the corresponding messages. Moreover, signatures could in this case be existentially forged by first choosing s and then setting $m := s^e \pmod{n}$. The ISO/IEC 9796 standard states a signing process for RSA that covers such problems.

2.7.2 The ElGamal Signature Scheme

The following signature scheme was proposed by ElGamal [ElG85a, ElG85b] together with the public key encryption scheme described in Section 2.5.3.

Let G be a finite cyclic group of order q, and let $g \in G$ be a generator of G such that computing discrete logarithms in G is infeasible. Let $y = g^x$ be the public key of the signer and x the corresponding secret key. Finally, let $\mathcal{H}$ be a collision-resistant hash function that maps $\{0,1\}^*$ to $\mathbb{Z}_q$.

Definition 2.21. *An* ElGamal signature *of a message $m \in \{0,1\}^*$ for the public key $y = g^x$ is a pair $(u,s) \in G \times \mathbb{Z}_q$ satisfying*

$$g^{\mathcal{H}(m)} = y^u u^s .$$

In the above definition, y^u means that the group element u is first represented as an integer $u' \in \mathbb{Z}_{|G|}$, u'' is set to $u' \pmod{q}$, and only then $y^u := y^{u''}$ is computed. In the following, as already mentioned in Section 2.1, we will implicitly assume that such conversions between group elements, integers and strings are done.

Such a signature (u,s) of a message m can be computed by the signer (who knows x) as follows:

1. choose r at random from $\mathbb{Z}_q^*$,
2. compute $u := g^r$, and
3. compute $s := r^{-1}(\mathcal{H}(m) - xu) \pmod{q}$.

The correctness of this signature scheme follows from the congruence

$$y^u u^s = g^{xu} g^{rs} = g^{xu+rs} = g^{\mathcal{H}(m)}.$$

We will briefly discuss the security of this signature scheme. Clearly, the signatures can be forged when computing discrete logarithms in G is feasible. Furthermore, it is important that the random value r is chosen differently each time a message is signed, since otherwise the secret key could be computed from two such signatures. Finally, if no hash function was used, signatures could be forged existentially. For a more detailed analysis of the security we refer to [AV96, Ble96, ElG85b, LL97, MvOV97, PS96]. The attacks of [LL97] apply also to other schemes based on the discrete logarithms that employ a prime order subgroup. However, they can easily avoided by choosing the actual system parameters carefully, e.g., choosing p and q such that all prime factors of $(p-1)/2q$ are larger than q.

A variant of the ElGamal signature scheme was proposed as a standard under the name Digital Signature Algorithm (DSA) by the U.S. National Institute of Standards and Technology (NIST) [Nat94]. An overview of all different variants of signature schemes of the ElGamal-style is given in [Pet96].

2.7.3 The Schnorr Signature Scheme

The Schnorr signature scheme [Sch91] is another variant of the ElGamal signature scheme. It is also an example of the construction of a signature scheme from an identification protocol. Compared to the ElGamal signature scheme, Schnorr's scheme achieves shorter signatures.

Let G be a finite cyclic group of order q and let $g \in G$ be a generator of G such that computing discrete logarithms in G is infeasible. Let $y = g^x$ be the public key of the signer and x be his secret key. Finally, let $\mathcal{H}$ be a collision-resistant hash function that maps $\{0,1\}^*$ to $\mathbb{Z}_q$.

Definition 2.22. *A* Schnorr signature *of a message $m \in \{0,1\}^*$ is a pair (c,s) with $c, s \in \mathbb{Z}_q$ satisfying the verification equation*

$$c = \mathcal{H}(m \| g^s y^c).$$

Remark 2.1. The size of the domain of the hash function (and thus of c) could be smaller than q, but should be at least 2^{160}.

A Schnorr signature (c,s) of a message m can be generated by the signer as follows:

1. choose r at random from $\mathbb{Z}_q$,
2. compute $c{:=}\mathcal{H}(m\|g^r)$ and
3. $s{:=}r - cx \pmod{q}$.

The correctness of this signing procedure follows from

$$g^s y^c = g^{s+xc} = g^{r-xc+xc} = g^r$$

and $\mathcal{H}(m\|g^s y^c) = \mathcal{H}(m\|g^r) = c$.

2.8 Blind Digital Signature Schemes

The concept of blind signatures was introduced by Chaum [Cha83] to be able to protect the privacy of users in applications such as electronic payment systems. In contrast to regular signature schemes, a blind signature scheme is an interactive two-party protocol between a *recipient* and a *signer*. It allows the recipient to obtain a signature of a message in a way that the signer learns neither the message nor the resulting signature.

Definition 2.23. *A* blind signature scheme *consists of a probabilistic algorithms* **gen**, *an interactive protocol* (R, S) *between the signer* S *and the recipient* R, *and a verification predicate* **ver**. *The algorithm* **gen** *generates a secret key* x_s *and and a corresponding public key* y_s *of a signer* S *when input the system parameters. The following must be satisfied if both players are honest:*

$$\mathtt{ver}(m, \sigma, y_s) = \begin{cases} \mathtt{true} & \textit{if } Prob\big(\sigma = [R, S(x_s)](y_s, m)\big) > 0 \\ \mathtt{false} & \textit{otherwise} \end{cases}$$

Furthermore, a blind signature scheme must be unforgeable. This means that it must be infeasible to compute a signature of a message with respect to a public key without engaging in the signing protocol with the signer. Finally, when given a set of views of protocol runs and the (unordered) set of message-signature pairs that resulted from these protocol runs, it must be infeasible for the signer to find the correspondence between views and message-signature pairs. This property is called blindness.

There are two kinds of blindness that disable a signer from linking views and message-signature pairs.

Definition 2.24. *A blind signature scheme* (R, S) *is called* statistically blind *if the tuples* $(m, [R, S](y_s, m))$ *and* $\langle S, R\rangle(\cdot)$ *are statistically independent, and* computationally blind *if linking views and message-signature pair requires the signer to solve some computationally infeasible problem (e.g., computing a discrete logarithm).*

The first realization of a blind signature scheme was presented by Chaum [Cha84]. It is based on the RSA signature scheme. Later, Chaum and Pedersen [CP93] and, independently, Okamoto [Oka93] presented schemes based on the discrete logarithm problem. Okamoto's schemes are based on the Schnorr signature scheme. Camenisch, Piveteau, and Stadler [CPS94a] proposed further discrete-logarithm-based blind signature schemes for a modification of DSA and for the Nyberg-Rueppel signature scheme [NR93]. Their approach was generalized by Horster, Michels and Petersen [HMP95, Pet96] for most of the ElGamal-style signature schemes. All these schemes are statistically blind.

A variation of blind signatures are the so-called partially blind signatures [AF96, AC97]. In such schemes a part of the message is signed blindly while the other part remains unblinded and is thus known to the signer when signing. Such schemes are useful to assure the signer that certain information is contained in a message that is otherwise blindly signed.

2.8.1 The Blind RSA Signature Scheme

In Figure 2.3 the blind RSA signature generation protocol as invented by Chaum [Cha84] is displayed. It allows a recipient to blindly obtain a signature of a message $m \in \mathbb{Z}_n^*$ from the signer who has the public key (e, n) of an ordinary RSA signature scheme. The signer's secret key is $d = e^{-1} \pmod{\varphi(n)}$. The protocol is correct since s is a valid signature of m:

$$s^e \equiv \tilde{s}^e r^{-e} \equiv (\tilde{m}^d)^e r^{-e} \equiv \tilde{m} r^{-e} \equiv m r^e r^{-e} \equiv m \pmod{n}.$$

To prove the protocol's blindness property, it must be shown that any given valid message-signature pair (m, s) could have originated in any given instance of the signing protocol. Let the signer's view of such an instance be $(\tilde{m}, \tilde{s})$. If (m, s) was generated during the instance of the

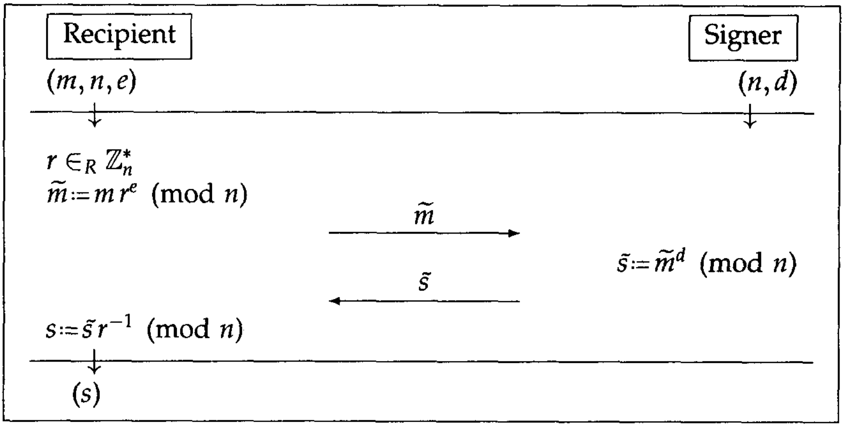
Recipient
Signer
(m,n,e)
(n,d)
$r \in_R \mathbb{Z}_n^*$
$\tilde{m} := m r^e \pmod{n}$
$\tilde{m}$
$\tilde{s} := \tilde{m}^d \pmod{n}$
$\tilde{s}$
$s := \tilde{s} r^{-1} \pmod{n}$
(s)

Figure 2.3: A protocol for obtaining a blind RSA signature of a message $m \in \mathbb{Z}_n^*$.

protocol leading to the view $(\tilde{m}, \tilde{s})$, then the blinding factor chosen by the recipient must have been

$$r' := (\tilde{m} m^{-1})^d \pmod{n}.$$

Now, we have to show that for this r' the congruence $\tilde{s} \equiv sr' \pmod{n}$ holds:

$$sr' \equiv s(\tilde{m} m^{-1})^d \equiv (s^e \tilde{m} m^{-1})^d \equiv (m \tilde{m} m^{-1})^d \equiv \tilde{m}^d \equiv \tilde{s} \pmod{n},$$

whereby we used the fact that s is a valid signature of m.

2.8.2 The Blind Schnorr Signature Scheme

The protocol shown in Figure 2.4 is a protocol for blindly issuing Schnorr signatures. It was first proposed in a slightly different version in [Oka93]. The setup of the system is the same as for the ordinary Schnorr signature scheme. The protocol allows a recipient to obtain a signature of a message $m \in \{0,1\}^*$ without the signer receiving any information about m or the resulting signature (c,s). If both players follow the protocol then the pair (c,s) is a valid Schnorr signature of m: since we have

$$g^s y^c = g^{\tilde{s}+\gamma} y^{\tilde{c}+\delta} = g^{\tilde{r}-\tilde{c}x+\gamma+\tilde{c}x} y^{\delta} = \tilde{t} g^{\gamma} y^{\delta} = t\,,$$

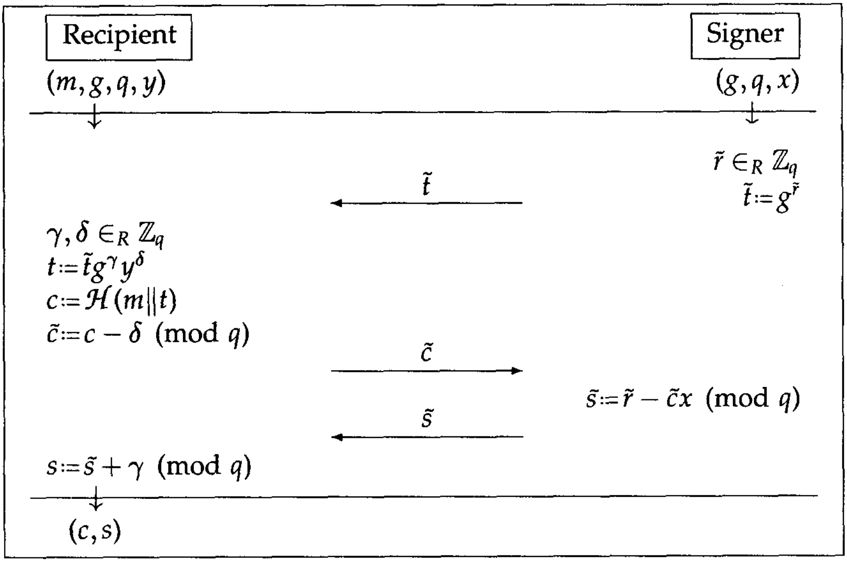
Recipient
Signer
(m, g, q, y)
(g, q, x)
$\tilde{r} \in_R \mathbb{Z}_q$
$\tilde{t} := g^{\tilde{r}}$
$\tilde{t}$
$\gamma, \delta \in_R \mathbb{Z}_q$
$t := \tilde{t} g^{\gamma} y^{\delta}$
$c := \mathcal{H}(m \| t)$
$\tilde{c} := c - \delta \pmod q$
$\tilde{c}$
$\tilde{s} := \tilde{r} - \tilde{c}x \pmod q$
$\tilde{s}$
$s := \tilde{s} + \gamma \pmod q$
(c, s)

Figure 2.4: A protocol for obtaining a blind Schnorr signature of a message $m \in \{0,1\}^*$.

the verification condition $c = \mathcal{H}(m \| g^s y^c)$ holds and the signature is valid. To prove that the protocol is blind, i.e., that the signer's view is statistically independent of both the message and the signature (c, s), one has to show that for every possible view and every possible signature there exists exactly one pair (γ, δ) of blinding factors which would result in that particular signature and view. Given any view consisting of $\tilde{r}$, $\tilde{t}$, $\tilde{c}$, $\tilde{s}$, and any signature (c, s) of a message m, let

$$\begin{array}{lcll} \gamma & := & s - \tilde{s} & \pmod q, \\ \delta & := & c - \tilde{c} & \pmod q, \quad \text{and} \\ t^* & := & \tilde{t} g^{\gamma} y^{\delta}. & \end{array}$$

To prove the blindness property, it remains to show that $t^* = t = g^s y^c$ is satisfied:

$$t^* = \tilde{t} g^{\gamma} y^{\delta} = g^{\tilde{r} + \gamma + x\delta} = g^{\tilde{r} + s - \tilde{s} + x(c - \tilde{c})} = g^{s + xc} g^{\tilde{r} - \tilde{s} - x\tilde{c}} = g^s y^c = t.$$

The last two equalities hold because $\tilde{s} \equiv \tilde{r} - x\tilde{c} \pmod q$ and since (c, s) is a valid signature.

2.9 Zero-Knowledge Proofs of Knowledge

Traditionally, a proof of a (target-)statement is a sequence of statements. When interpreted, this sequence eventually leads to the validity of the target-statement. Anyone who is able to interpret the sequence can verify the proof. Such a proof is a fixed object that, once obtained, can be passed on to other people to convince them of the validity of the statement, too.

In contrast, an *interactive proof* of a statement is an interactive protocol between two entities, a prover and a verifier. After the execution of the protocol, the verifier is convinced of the validity of the statement. Interactive proofs have two remarkable advantages compared to conventional proofs. First, there exist protocols for proving a statement such that the verifier does not receive any information apart from the validity of the statement. Hence, the verifier is not able to prove the statement to other people. Such protocols are called *zero-knowledge.* Second, there are statements that can be proved interactively but not with a conventional proof [Sha90].

Interactive proof systems were invented by Babai et al. [Bab85, BM88] and by Goldwasser et al. [GMR85] who also introduced the notion of zero-knowledge. Goldreich et al. showed that all statements in **NP** have a zero-knowledge proof-system [GMW87b]. In the model used in these papers, the prover is computationally unbounded, while the verifier is a probabilistic polynomial-time machine.

Building on this model, several papers (e.g., [BCC88, BC86, Cha87]) modified the original model such that the prover is a (probabilistic) polynomial-time machine. Whereas before, the prover was powerful enough to compute a witness of the statement, now he is not. Thus, either the statement is in **P**, or the prover must 'know' a witness, i.e., store the witness or compute it in polynomial time from other stored information. To distinguish between the models, protocols in the latter model are usually called *arguments, computationally convincing proofs,* or *proofs of knowledge,* while the term *proof systems* refers to the Goldwasser, Micali, and Rackoff model. For a discussion of the different models we refer to [BC89].

In the following we will consider only protocols between players that are computationally bounded, i.e., probabilistic polynomial-time Turing machines.

2.9.1 Interactive Proofs of Knowledge

The concept of a proof of knowledge was first mentioned as a remark in [GMR85], formalized by Feige, Fiat, and Shamir [FFS88], and then refined by many others (e.g. [TW87, FS90, BG92]). Below, we give a definition of a proof of knowledge similar to that presented in [BG92].

Definition 2.25. *Let $R \subseteq \{0,1\}^* \times \{0,1\}^*$ be a polynomially bounded binary relation and let L_R be the language defined by R. An* interactive proof of knowledge *is a protocol (P, V) that has the following two properties:*

Completeness: If $(x, w) \in R$ then $[V, P(w)](x) = \texttt{accept}$.

Validity: There exists a probabilistic expected polynomial-time machine K (knowledge extractor) *such that for every $\tilde{P}$, for all polynomials $p(\cdot)$ and all sufficiently large $x \in L_R$,*

$$Prob\big((x, K^{\tilde{P}(x)}) \in R\big) \geq Prob\big([V, \tilde{P}](x) = \texttt{accept}\big) - \frac{1}{p(|x|)} .$$

The probabilities are taken over all random choices of V, P, $\tilde{P}$, and K, respectively.

In this definition $\tilde{P}$ denotes any prover who does not necessarily know the secret. Moreover, $K^{\tilde{P}(x)}$ means that the knowledge extractor is given oracle access to $\tilde{P}$, i.e., can reset and rerun $\tilde{P}$ on input x.

The validity property captures the meaning of an interactive proof of knowledge, namely that P demonstrates her knowledge via communication with V and that one can extract this knowledge via communication with $\tilde{P}$ only. It also captures what the knowledge of a machine means: everything that it can compute in expected polynomial time (e.g., by using the knowledge extractor).

The above definition gives no requirements for the case $x \notin L_R$, i.e., a proof of knowledge for R does not necessarily give an interactive proof for language membership in L_R. If this is required, or if the protocol is also run on inputs $x \notin L_R$ then the following additional property should also be satisfied.

Soundness: For every $\widetilde{P}, \forall x \notin L_R$,

$$\text{Prob}\big([V, \widetilde{P}](x) = \texttt{accept}\big) < \frac{1}{2}$$

holds. The probabilities are taken over all random choices of V and $\widetilde{P}$.

If a protocol satisfies this soundness criterium, the probability of acceptance can be made arbitrarily small by repeating it sequentially sufficiently many times. For further details and explanation on proofs of knowledge we refer to the paper by Bellare and Goldreich [BG92].

2.9.2 Zero Knowledge Protocols

The notion of zero-knowledge makes it possible to prove that a protocol is secure for the prover, i.e., that the verifier is not able to gain any information from the interaction with the prover that he could not have computed himself, no matter how he deviates from the protocol.

To define zero-knowledge, we need some notions of indistinguishability of random variables as put forth in [GM84, GMR85].

Definition 2.26. *Let $L \in \{0,1\}^*$ be a language and let $A = \{A(x)\}_{x \in L}$ and $B = \{B(x)\}_{x \in L}$ be two ensembles of random variables indexed by strings $x \in L$. We say that the ensembles A and B are*

- perfectly indistinguishable *if for all $x \in L$ the random variables $A(x)$ and $B(x)$ are identically distributed.*

- statistically indistinguishable *if their statistical difference is negligible, or more technically, if for every polynomial $p(\cdot)$ and for all sufficiently long $x \in L$ it holds that*

$$\sum_{\alpha \in \{0,1\}^*} \big|Prob\big(A(x) = \alpha\big) - Prob\big(B(x) = \alpha\big)\big| \; < \; \frac{1}{p(|x|)} \, .$$

- computationally indistinguishable *if no efficient algorithm exists that can distinguish them, i.e., for every probabilistic polynomial-time* *algorithm D, for every polynomial $p(\cdot)$ and for all sufficiently long $x \in L$ it holds that*

$$\left|Prob\big(D(x, A(x)) = 1\big) - Prob\big(D(x, A(x)) = 1\big)\right| \; < \; \frac{1}{p(|x|)} \, .$$

According to these three kinds of indistinguishability there are three different degrees of zero-knowledgeness of an interactive protocol.

Definition 2.27. *[GMR85] An interactive protocol (P, V) is said to be* perfect/statistical/computational zero-knowledge, *if for every probabilistic polynomial-time verifier $\widetilde{V}$ there exists a probabilistic expected polynomial-time simulator $S_{\widetilde{V}}$ so that the two ensembles*

$$\{[\widetilde{V}, P](x)\}_{x \in L} \;\; and \;\; \{S_{\widetilde{V}}(x)\}_{x \in L}$$

are perfectly/statistically/computationally indistinguishable.

When a protocol is simply said to be "zero-knowledge", most authors mean that it is computational zero-knowledge. We follow this convention. An alternative, but equivalent definition is to require the simulator $S(\cdot)$ to output $\widetilde{V}$'s view $\langle \widetilde{V}, P \rangle(\cdot)$ rather that $\widetilde{V}$'s output.

There exist several different flavors of definitions for zero-knowledge. For instance, some definitions require the simulator to run in strict polynomial time but allow the simulator to fail with a small probability (e.g., [Gol95]).

An interesting question is whether the composition of zero-knowledge protocols remains zero-knowledge. While a parallel composition of zero-knowledge protocols is in general no longer zero-knowledge, sequential compositions can be shown to be zero-knowledge, when the definition is slightly modified [GO94], i.e., the verifier and the simulator are allowed an extra input z the size of which is polynomially bounded in the size of x. This definition is called *auxiliary input zero knowledge.*

To prove that a protocol is zero-knowledge according to Definition 2.27, one would have to construct a simulator for every possible verifier. In practice, this is often done by constructing a single simulator that works for all verifiers. To match this situation, a third definition of zero-knowledge was proposed, *black-box zero-knowledge.* Here, it is required that there exists a single simulator that works for all verifiers. This single simulator is allowed to use a verifier as a black-box, i.e., the simulator can choose the input and the coin tosses of the verifier. It has been shown [GO94] that all protocols that are black-box zero-knowledge are a subset of all protocols that are auxiliary-input zero-knowledge, which in turn are a proper subset of the protocols that are zero-knowledge according to Definition 2.27.

A slightly weaker requirement than zero-knowledge is honest-verifier zero-knowledge. It formalizes the property of a interactive protocol that a verifier who follows the protocol specification (runs V) cannot extract information from the prover.

Definition 2.28. *An interactive protocol* (P, V) *is said to be* perfect (statistical/computational) honest-verifier zero-knowledge, *if there exists a probabilistic expected polynomial-time simulator* S_V *so that the two ensembles* $\{[V, P](x)\}_{x \in L}$ *and* $\{S_V(x)\}_{x \in L}$ *are perfectly (statistically/ computationally) indistinguishable.*

2.9.3 Witness Hiding Protocols

Feige and Shamir [FS90] proposed an alternative criterium of security: witness indistinguishability and witness hiding. The latter seems to be a natural security requirement that can replace zero-knowledge in many cryptographic protocols. Moreover, it has the advantage over zero-knowledge that it is preserved under arbitrary composition of protocols.

Informally, a proof of knowledge is witness indistinguishable if the verifier cannot tell which witness the prover is using (even if he knew all witnesses).

Definition 2.29. *Let* (P, V) *be a proof of knowledge for a binary relation* R. *It is called* witness indistinguishable *if for every polynomial-time verifier* $\tilde{V}$, *all sufficiently long* $x \in L_R$, *any two sequences* $W = \{w_x\}_{x \in L_R}$ $W' = \{w'_x\}_{x \in L_R}$ *such that* $w_x, w'_x \in R(x)$, *and all auxiliary inputs* $z \in \{0,1\}^*$ *the two ensembles random variables*

$$\{x, \langle P(w), \tilde{V}(z)\rangle(x)\}_{x \in L_R} \text{ and } \{x, \langle P(w'), \tilde{V}(z)\rangle(x)\}_{x \in L_R}$$

are computationally indistinguishable.

A variation of witness indistinguishability due to Goldreich [Gol95] is *witness independence*. It requires the random variables $\langle P(w_1), \widetilde{V}(z)\rangle(x)$ and $\langle P(w_2), \widetilde{V}(z)\rangle(x)$ to be identically distributed.

If an instance has only a single witness, then the related protocol is trivially witness indistinguishable. In their paper Feige and Shamir proved that witness indistinguishability is preserved under polynomial composition of protocols. Furthermore, every zero-knowledge protocol is witness indistinguishable.

Before we can define what a witness hiding proof is, we need the notion of an invulnerable generator for a relation.

Definition 2.30. *G_R is a generator for relation R if on input 1^n it produces instances $(x, w) \in R$ of length n. G_R is called an* invulnerable generator *if for any polynomial-time* cracking *algorithm C, for all polynomials $p(\cdot)$, and for all sufficiently large n we have*

$$Prob\big((x, C(x)) \in R\big) < \frac{1}{p(n)},$$

where $x = \mathrm{pr}_1(G_R(1^n))$. The probabilities are taken over the random choices of G_R and C.

Recall that $\mathrm{pr}_1(\cdot)$ denotes the first entry of a tuple given in the argument.

Intuitively, a proof of knowledge is witness hiding if participating in the protocol does not help a (dishonest) verifier to compute a new witness to the input which he did not know at the beginning of the protocol, i.e., if the verifier is able to compute a witness after participating in the protocol, then he could have done so before.

Definition 2.31. *[FS90] Let (P, V) be a proof of knowledge for relation R, and let G_R be an invulnerable generator for R. The protocol (P, V) is said to be* witness hiding *on (R, G_R) if there exists an expected polynomial-time* witness extractor *W, such that for any polynomial-time $\widetilde{V}$, for all polynomials $p(\cdot)$, and for all sufficiently large n we have*

$$Prob\big((x, [\widetilde{V}, P(w)](x)) \in R\big) < Prob\big((x, W^{\widetilde{V}, G_R}(x)) \in R\big) + \frac{1}{p(n)},$$

where w is arbitrarily distributed over $R(x)$, and $x = \mathrm{pr}_1(G_R(1^n))$. The probabilities are taken over the distributions of the inputs and witnesses, as well as over the random choices of P and W.

In contrast to zero-knowledge, the distribution of the input enters the definition. In particular, there may be infinitely many witnesses that are easily extracted, but these are not output by the generator.

The definition of witness hiding guarantees only that witnesses are not disclosed completely. In contrast to zero-knowledge, partial information may leak. For instance, a digital signature cannot be zero-knowledge, but they can be witness hiding (e.g., fail-stop signatures [vHP93, WP90]).

Witness indistinguishable proofs are not necessarily witness hiding; for instance, if for each instance there exists only a single witness, then a protocol that yields this single witness is (trivially) witness indistinguishable. However, if each input has at least two "computationally independent" witnesses, then a witness indistinguishable proof of knowledge is also witness hiding, as is shown in [FS90]. Furthermore, parallel compositions of the protocol are also witness hiding, since witness indistinguishability is maintained under composition of protocols.

2.9.4 An example: Schnorr's Identification Scheme

In [Sch91] it is shown that Schnorr's identification scheme [Sch91] (see also Figure 2.2) is a proof of knowledge, but that it is not zero-knowledge. Chaum et al. [CEvdG88] have shown that the variant with $k = 1$ and $q = p - 1$ is a zero-knowledge proof of knowledge when sequentially repeated $\log(p)$ times. In this subsection we will analyze the Schnorr identification protocol with respect to the definitions presented in this section. Let $\mathcal{G}$ be a family of groups with prime order such that computing discrete logarithms in them is infeasible. The binary relation R_s underlying the protocol is the set $\{((G, g, y), x) \mid y = g^x \text{ with } 0 \leq x < \mathrm{ord}(G)\,;\, y, g \neq 1 \in G\,;\, G \in \mathcal{G}\}$ and let L_s be the language defined by R_s.

To show that the Schnorr identification protocol is a proof of knowledge for the relation R_s we have to prove the protocol's completeness and validity. As mentioned earlier, it is easy to see that P can always convince V and hence the protocol is complete. To prove the protocol's validity we construct a knowledge extractor. Let ε denote $\mathrm{Prob}\big([V, \tilde{P}](G, g, y) = \texttt{accept}\big)$ and let ℓ denote the length of the input $\big(\approx 3 \log g\big)$. Consider the following program for a knowledge extractor

K that has oracle access to the "prover" $\widetilde{P}$:

1. Run $\widetilde{P}$ using a randomly chosen $c \in \{0,1\}^k$. Proceed if the obtained triple (t, c, s) is accepting, otherwise output $\perp$ and halt.
2. Reset and run $\widetilde{P}$ again with a randomly chosen $\tilde{c} \in \{0,1\}^k$ until an accepting triple $(t, \tilde{c}, \tilde{s})$ is found. If $\tilde{c} \neq c$ proceed to Step 3, otherwise output $\perp$ and halt.
3. Output $x := \dfrac{\tilde{s} - s}{c - \tilde{c}} \pmod{q}$

Let p_t denote the probability that $\widetilde{P}$ outputs a commitment t and let ε_t denote the probability that then, on input of a random $c \in_R \{0,1\}^k$, $\widetilde{P}$ outputs an s such that (t, c, s) is an accepting triple. Then we have $\varepsilon = \sum_t p_t \varepsilon_t$.

Let us first discuss the expected running-time of K. Recall that verifying whether a triple is accepting requires $O(\ell^3)$ steps. Consider a particular t. Then, the probability that K halts in the first step is $1 - \varepsilon_t$ and the running time is $O(\ell^3)$ (a call to the oracle $\widetilde{P}$ counts as one step). In the other two cases we have an expected running time of $1/\varepsilon_t O(\ell^3)$. However, since Step 2 is only entered with probability ε_t, the total expected running-time given a particular t is

$$\left(1 - \varepsilon_t + \varepsilon_t \frac{1}{\varepsilon_t}\right) O(\ell^3) = (2 - \varepsilon_t) O(\ell^3).$$

Since a particular t gets chosen with probability p_t, the expected running time of the knowledge extractor is

$$\sum_t p_t (2 - \varepsilon_t) O(\ell^3) = (2 - \varepsilon) O(\ell^3) \leq 2O(\ell^3),$$

which is polynomial in the length of the input as required.

What remains to discuss is the probability that the extractor K will output a witness and not the special symbol $\perp$. The probability that in the second step a triple with $\tilde{c} \neq c$ is found is

$$\frac{\varepsilon_t 2^k - 1}{\varepsilon_t 2^k} = 1 - \frac{1}{\varepsilon_t 2^k},$$

since we have $\varepsilon_t 2^k > 1$ accepting triples in Step 2 and one of which we cannot use. Again, the probability that Step 2 is reached is ε_t and thus the total success probability is

$$\text{Prob}\big(((G,g,y), K^{\widetilde{P}((G,g,y))}) \in R_s\big) = \sum_t p_t \varepsilon_t (1 - \frac{1}{\varepsilon_t 2^k}) = \varepsilon - 2^{-k}.$$

For all polynomials $p(\cdot)$ and all sufficiently large ℓ, this probability is at least $\varepsilon - p(\ell)^{-1}$ if $k = \Theta(\text{poly}(\ell))$ holds. Hereby we have constructed a knowledge extractor.

Lemma 2.1. *The Schnorr identification protocol is a proof of knowledge for* $k = \Theta(\text{poly}(\ell))$.

Let us discuss next under which conditions the Schnorr identification protocol is zero-knowledge. As stated in [Sch91] it is not zero-knowledge when k is selected as required in Lemma 2.1. This is because k is too large and thus the success-probability of the simulator is negligible. However, when a smaller k is chosen, namely $k = O(\log(\ell))$, it is zero-knowledge. The following algorithm constitutes a simulator for the output of any verifier $\widetilde{V}$.

1. choose $\tilde{c} \in_R \{0,1\}^k$.
2. choose $r \in_R \mathbb{Z}_q$.
3. compute $t{:=}g^r y^{\tilde{c}}$.
4. run $\widetilde{V}$, send it the computed t and receive a c.
5. if c equals $\tilde{c}$ send $s{:=}r$ to $\widetilde{V}$ and output $\widetilde{V}$'s output and halt, otherwise continue with Step 1

By construction, the output of the simulator is identically distributed to the output of the verifier. To conclude that the protocol is zero-knowledge, it remains to discuss the running time of the simulator. The probability that in Step 5 the variable c will equal $\tilde{c}$ (and thus that simulator will halt) is 2^{-k}, since for the verifier $\widetilde{V}$ all possible choices of $\tilde{c}$ are equally likely. Hence the expected runtime of the simulator is $O(2^k)$. Therefore, we have to choose k as $O(\log(\ell))$ so that the simulator's expected running time is polynomial in ℓ. However, for such a choice the

Schnorr identification scheme is no longer a proof of knowledge since this would require that $k = \Theta(\text{poly}(\ell))$ holds.

To get both properties at the same time, the protocol must be repeated in a sequential manner. Let j be the number of repetitions (rounds). To simulate the output of any verifier $\widetilde{V}$ in the obtained protocol, we need to use the fact that the simulator is allowed to reset and rerun the verifier. Before the first round, the simulator resets the verifier. Then, the simulator tries to find values which pass the first round (similar to the steps as described above) and stores these values. Using these values, the simulator always passes the first round, and can try to find values for the second round, and so on. The expected running time of this simulator is $O(\text{poly}(j2^k))$ and thus j must be chosen as $O(\text{poly}(\ell))$.

Let us now consider how j must be chosen so that we get a proof of knowledge. The knowledge extractor is similar to that for the ordinary protocol, with the difference that in Steps 1 and 2, not only a single protocol run is considered but all j iterations. In this case, the expected running time remains polynomial in ℓ. The probability that the extractor will output a witness becomes $\varepsilon - 2^{-kj}$. As the zero-knowledge property requires k to be $O(\log(\ell))$, the parameter j must be satisfy $\Omega(\text{poly}(\ell))$. Thus we have the following lemma.

Lemma 2.2. *The Schnorr identification protocol is a proof of knowledge and is perfect zero-knowledge for $k = O(\log(\ell))$ when sequentially repeated $\Theta(\text{poly}(\ell))$ times.*

The Schnorr identification protocol is not known to be witness hiding but is trivially witness indistinguishable since for each instance there exists only one witness. However, Okamoto presented a variant that is witness hiding [Oka93]. This variant uses two different bases and thus two secret exponents. So, there exist q different witnesses per instance. This variant is witness indistinguishable and thus also witness hiding.

2.10 Hash Functions

In signature schemes, hash functions are applied to reduce messages of arbitrary length to bit-strings that can be handled by the signing algorithm. Hash functions can also be used as a substitution for the verifier in proofs of knowledge and thus turn them into signature schemes.

Definition 2.32. *A* hash function *is a function mapping binary strings of arbitrary finite length to binary strings of a fixed length ℓ:*

$$\mathcal{H} : \{0,1\}^* \to \{0,1\}^\ell.$$

Naturally, we require a hash function to be efficiently computable. For cryptographic purposes a hash function must also be hard to invert, i.e., it must have at least one of the following properties:

- *weak collision resistant:* For a given x, it is hard to find an $x' \neq x$ such that $\mathcal{H}(x) = \mathcal{H}(x')$ [NY89].
- *strong collision resistant:* It is hard to find a pair (x, x') with $x \neq x'$ such that $\mathcal{H}(x) = \mathcal{H}(x')$ if $\mathcal{H}$ is chosen at random from a family of hash-functions [Dam88].
- *one-way:* For a given c, it is hard to find an x such that $\mathcal{H}(x) = c$.

It can easily be seen that a strong collision resistant hash function is also one-way and of course weak collision resistant. In the context of this thesis, a weak collision resistant hash function meets all requirements. However, for some applications more specific properties such as correlation freeness are needed [Oka93, And95]. For a recent discussion on the subject of strong and weak collision resistance see [BR97].

The notion of hardness depends on the actual security requirements. Assuming that the best algorithm to invert the hash function is brute-force search, the hardness depends mostly on the number of output-bits. To estimate the number of necessary output-bits the so-called birthday-attack has to be considered: to find a collision with probability $1/2$ only about $2^{\ell/2}$ random hashes must be made. Today, an output size of 160 bits seems to yield a reasonable security.

Numerous cryptographic hash functions have been proposed so far in the literature. They can be divided into two kinds by their construction:

I. Hash functions that use block ciphers as building blocks.

II. Specially designed hash functions.

An example of the first type is the Tandem DM based on IDEA [LM93]. Examples of the second type are the secure hashing algorithm (SHA) [Nat93], Ron Rivest's MD5 [Riv92], or the RIPEMD-160 by Dobbertin, Bosselaers and Preneel [DBP96, BDP97]. Recent attacks on MD5 by Dobbertin [Dob96] make the use of SHA or RIPEMD-160 advisable. Preneel's thesis [Pre93] is a comprehensive treatment on hash functions.

2.11 Secret Sharing

Secret sharing schemes are used often as a building block of cryptographic systems. For instance, they are instrumental to multi-party computations and distributed protocols (e.g., [GMW87a, Fra93, Can95]).

Informally, a secret sharing scheme is a procedure that allows a group of people to share a secret in such a way that no one of them gets any information about the secret. However, all members of any designated subset can pool their obtained shares and reconstruct the secret again. Such schemes were independently invented by Blakley [Bla79] and Shamir [Sha79]. For a survey on different schemes we refer to Simmons [Sim91] and Stinson [Sti92].

Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be a set of participants. A set $\{P_i | i \in \mathcal{S}\}$ of participants that can jointly reconstruct the secret σ is called a *qualified subset* of $\mathcal{P}$. In this case we also refer to $\mathcal{S}$ as a qualified set. The set of all qualified sets $\Gamma \subseteq 2^{\{1,\ldots,n\}}$ is called the *access structure* and is usually required to be monotone, which means that the following holds.

$$\text{If } \mathcal{S}_1 \in \Gamma \text{ and } \mathcal{S}_1 \subseteq \mathcal{S}_2 \subseteq \{1, \ldots, n\} \text{ then } \mathcal{S}_2 \in \Gamma.$$

A *minimal* qualified subset $\mathcal{S} \in \Gamma$ is a set of participants such that $\mathcal{S}' \notin \Gamma$ for all proper subsets $\mathcal{S}'$ of $\mathcal{S}$. A *basis* of Γ, denoted by Γ_0, is the set of all minimal qualified subsets.

A common special case of a monotone access structure is a *threshold* structure, where for a threshold k the access structure Γ is defined to be $\{\mathcal{S} \subseteq 2^{\mathcal{P}} \mid |\mathcal{S}| \geq k\}$.

Every access structure Γ has a natural *dual* access structure Γ^*:

$$\mathcal{S} \in \Gamma^* \iff \overline{\mathcal{S}} \notin \Gamma ,$$

where $\overline{S}$ denotes the complement of S in $\{1,\dots,n\}$. If Γ is monotone, then Γ^* is also monotone and we have $(\Gamma^*)^* = \Gamma$. If Γ is a threshold structure, then so is Γ^*. For instance, the dual access structure of $\Gamma = \{S \subseteq 2^{\{1,\dots,n\}} \mid |S| \geq k\}$ is $\Gamma^* = \{S \subseteq 2^{\{1,\dots,n\}} \mid |S| \geq n-k+1\}$.

Definition 2.33. *A* secret sharing scheme *for $\mathcal{P}$, Γ, and a set of secrets Σ consists of three procedures* $\mathtt{gen}_\Gamma$, $\mathtt{ver}_\Gamma$, *and* $\mathtt{rec}_\Gamma$. *The probabilistic algorithm* $\mathtt{gen}_\Gamma$ *takes a secret $\sigma \in \Sigma$ and $n = |\mathcal{P}|$ as input and outputs a share ς_i for every participant $P_i \in \mathcal{P}$. The deterministic algorithm* $\mathtt{rec}_\Gamma$ *takes a qualified subset S and the list $(\varsigma_i)_{i\in S}$ as input and outputs the secret σ. The following must hold:*

$$\forall \sigma \in \Sigma \quad \forall S \in \Gamma : \quad \sigma = \mathtt{rec}_\Gamma\big(S, (\varsigma_i)_{i \in S}\big)\ .$$

The procedure $\mathtt{gen}$ is often run by a third, trusted party D, called the *dealer*, who distributes the shares to the individual participant. But there are also schemes that do not require a dealer, i.e., where $\mathtt{gen}$ is a multi-party protocol among the participants. Before running the algorithm $\mathtt{rec}$, the participants of a qualified subset may have to pool their shares (and thereby make the shares known to each other). However, there exist schemes with algorithms $\mathtt{rec}$ that allow the participants to compute a function of the secret in a distributed manner without revealing any other information about their shares and the secret than the computed result [DDFY94].

A secret sharing scheme may have the following properties:

Perfect: A secret sharing scheme is called *perfect* if all subsets of participants that do not form a qualified set are unable to obtain any information about the secret σ or about shares of other participants. Such schemes are also called information theoretically secure. There are also computationally secure secret sharing scheme, where it is infeasible to compute the secret σ for any subset not in Γ (e.g. [Kra94]).

Ideal: A secret sharing scheme is called *ideal* if the size of the shares equals the size of the secret.

Verifiable: A secret sharing scheme is called *verifiable* if each participant can verify that he/she has indeed obtained a valid share, i.e., the dealer need not be trusted. Such a scheme was first proposed by Chor et al. [CGMA85]. Verifiable secret sharing schemes require a third algorithm $\mathtt{ver}_\Gamma$, that takes as input a share, and outputs **true** if and only if the share is valid. Hence, the participants can convince themselves that the shares are valid. Schemes that also allow other entities to validate the shares of all participants are called *publicly verifiable* [Sta96b].

Benaloh and Leichter [BL90] proposed a simple and elegant construction for realizing a secret sharing scheme for any (monotone) access structures. Schemes obtained by this so-called monotone circuit construction are perfect, but not ideal, since every participant gets as many shares as the number of qualified subsets he/she belongs to, and all shares have the same size as the secret.

Perfect secret sharing schemes have the property that it is possible to construct a complete set of shares given the shares of any non-qualified set and the secret, with the same probability distribution of the shares as when they were generated by $\mathtt{gen}_\Gamma$ [Cra97]. Formally, the algorithm $\mathtt{cmpl}_\Gamma$ takes as inputs the set of participants $\mathcal{P}$, a non-qualified set of participants $\mathcal{N} \notin \Gamma$, the set $\{\varsigma_i | i \in \mathcal{N}\}$ of their shares, and the secret σ, and outputs the set $\{\varsigma_j | j \in \overline{\mathcal{N}}\}$, i.e., we have

$$\mathtt{cpml}_\Gamma(\mathcal{P}, \mathcal{N}, \{\varsigma_i | i \in \mathcal{N}\}, \sigma) = \{\varsigma_j | j \in \overline{\mathcal{N}}\} ,$$

where $\overline{\mathcal{N}}$ denotes the complement of $\mathcal{N}$ in $\{1, \ldots , n\}$.

Finally, as an example, consider secret sharing scheme conceived by Shamir [Sha79]. It is a threshold secret sharing scheme with n participants. A secret σ (an element of a finite field $GF(q)$, with $q > n$) is shared by randomly choosing the coefficients $\alpha_1, \ldots , \alpha_{k-1} \in GF(q)$ of the polynomial

$$f(X) := \alpha_{k-1} X^{k-1} + \ldots + \alpha_1 X + \sigma \pmod{q} ,$$

where k is the threshold k. The share for participant P_i is then calculated as $\varsigma_i := f(p_i)$, where p_i is a publicly known element of $GF(q)$ associated with participant P_i, e.g., $p_i = i$. Given k or more shares, the polynomial f and thus the secret σ can be reconstructed by Lagrange interpolation on the points (p_i, ς_i). This scheme is ideal as well as perfect.

Chapter 3

Proofs of Knowledge About Discrete Logarithms

This chapter provides building blocks for cryptographic systems based on the hardness of the discrete logarithm problem. Various known methods for proving the knowledge of secret keys and for proving that, additionally, they satisfy given predicates, are summarized and unified. Furthermore, a new method for proving that the secret keys satisfy modular relations is presented.

This chapter also provides building blocks for the different group signature schemes and the payment systems described in the next chapters.

3.1 Introduction

Crypto-systems found in literature that are based on the difficulty of the discrete logarithm problem often employ similar techniques for proving knowledge and properties of secret keys. These are for instance proofs of knowledge of discrete logarithms (with respect to a single or to multiple generators) and proofs that known discrete logarithms satisfy some given predicates such as the equality of two of them. In this chapter we summarize and generalize such techniques and present them within a general framework. We also propose new techniques for proving that the secret keys known to the prover satisfy some given modular (polynomial) relations.

All proof systems presented in this chapter can be described as three-move protocols similar to Schnorr's identification protocol depicted in Figure 2.2. These protocols can be shown to be honest-verifier zero-knowledge proofs of knowledge. Furthermore, if they are repeated sequentially sufficiently many times and if the challenge is chosen from a sufficiently small set, the protocols are zero-knowledge proofs of knowledge (cf. Sect. 2.9.4).

Using the techniques introduced in [FS87, FFS88], every honest-verifier zero-knowledge proof can be turned into a signature scheme by replacing the verifier by a hash function, i.e., the challenge is set to the hash value of the commitment (and of the message to be signed). This approach has been formalized by Bellare and Rogaway as the so-called random oracle model [BR93, BR96]. In this model, the hash-function (or the verifier, respectively) is replaced by an oracle. The oracle answers queries with random strings, except that the same query always yields the same answer. Bellare and Rogaway proved that in this model the signature scheme of [GMR88] is secure against an adaptively chosen message attack. Pointcheval and Stern [PS96, Poi96] applied this result to a variant of the ElGamal signature scheme and stated that every signature scheme obtained from an honest-verifier zero-knowledge proof of knowledge is secure against existential forgery under an adaptively chosen message attack.

Finally, using a result of Okamoto [Oka93], it is also possible to derive blind signature schemes from the schemes presented in this chapter (cf. Section 2.8).

Throughout this thesis, we describe the different proofs of knowledge not as protocols but rather as signature schemes derived from these protocols, since we most often use them as such. However, the reader should keep in mind that there always exists a corresponding protocol being a proof of knowledge. To reflect this, we call the schemes *signatures based on proofs of knowledge*, SPK for short. We also say that a signature 'proves' the knowledge of secret keys, although, strictly speaking, a signature can never be a proof in the strict sense as defined in Section 2.9, but only an argument [BCC88].

3.2 Proving Knowledge of Secret Keys

3.2.1 Algebraic Setting

The algebraic setting is as follows. Let G be a finite cyclic group of prime order q, k an integer, and let $g, h, g_1, \ldots, g_k \in G$ be generators of G such that computing discrete logarithms of any group element (apart from the the identity element) with respect to one of the generators is infeasible. Furthermore, the generators should be chosen in a random manner, such that the discrete logarithms of no generator with respect to another are known. Then, the computation of a representation (index tuple) of a group element with respect to multiple generators is as hard as the discrete logarithm problem (see Section 2.3). In practice, G is often chosen as a subgroup of $\mathbb{Z}_p^*$ for some prime p or a group generated by the points of an elliptic curve over some field [Men93]. Depending on the application, it may be important for the participants to assure themselves that G is properly chosen, i.e., has prime order. For instance, in the case of G not having prime order (e.g., $G = \mathbb{Z}_p^*$) and depending on the prime factors of $|G|$, there are a number of attacks known [AV96, Ble96, LL97] that range from existentially forging signature to computing the secret key of signer from given signatures. Finally, let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\ell$ denote a strong collision-resistant hash function (see Section 2.10). We assume that all these parameters are publicly known and accessible.

The prover chooses her secret keys (randomly) from $\mathbb{Z}_q^*$. (Again, we call the prover Peggy and the verifier Vic.) Public keys are then computed as the product of some generators that are exponentiated with some of the secret keys. For instance, the prover could choose $x_1, x_2 \in_R \mathbb{Z}_q^*$ as secret keys and compute her public keys $y_1 := g_1^{x_1} g_5^{x_2}$ and $y_2 := g^{x_1 + 2x_2}$.

3.2.2 First Building Blocks and Notation

The first and simplest building block is an SPK of the discrete logarithm of a public key y to the base g.

Definition 3.1. *A pair* $(c, s) \in \{0,1\}^\ell \times \mathbb{Z}_q$ *satisfying*

$$c = \mathcal{H}(S \| V \| m) \quad \textit{with } S = g \| y \textit{ and } V = g^s y^c$$

is an SPK of the discrete logarithm of a group element y to the base g of the message $m \in \{0,1\}^*$ and is denoted

$$SPK_1\{(\alpha) : y = g^{\alpha}\}(m) \,.$$

Basically, such an SPK is a Schnorr signature (see Section 2.7) with a slightly different argument to the hash function.

An $SPK_1\{\alpha : y = g^{\alpha}\}(m)$ can be computed if a value (secret key) $x \equiv log_g y \pmod q$ is known, by choosing a random integer r from $\mathbb{Z}_q$ and computing $t{:=}g^r$ and then c and s according to

$$c{:=}\mathcal{H}(g\|y\|t\|m)$$

and

$$s{:=}r - cx \pmod q.$$

If the base and the public key are evident from the context, they will sometimes be omitted in the argument to the hash function. Although this "proof" is not interactive, we call t the commitment, c the challenge, and s the response (cf. Section 2.6).

Along with this first building block we also introduced our notation. An expression such as

$$SPK_i\{(\alpha,\beta) : \; y = g^{\alpha} \; \wedge \; z = g^{\beta}h^{\alpha}\}(m)$$

denotes a signature based on a proof of knowledge of values (secret keys) α and β such that the statement on the right hand side of the colon is true. In the above example, the knowledge of α and β such that $y = g^{\alpha} \; \wedge \; z = g^{\beta}h^{\alpha}$ is hold is proven. This is equivalent to proving the knowledge of the discrete logarithm of y to the base g and of a representation of z to the bases g and h and, in addition, that the h-part of this representation equals the discrete logarithm of y to the base g. We will stick to the convention that Greek letters denote the knowledge of the prover. If the message m is the null string, the term (m) after the '$\}$' is omitted. The index i of SPK_i is thought as reference to the definition of a particular SPK_i. For the general understanding of what is proven by a signature, the index is not important. However, when it comes to computing and verifying a signatures, the index i refers to the exact procedures for doing so. With higher indices these procedures will get more complex, but also allow more complex statements to be proven. With regard to the arguments of the hash function, we stick to the following convention:

- the argument includes the message m that is signed (however, m might be the null-string),
- the string S contains the commitment of the knowledge (or statement) that is being proven, and
- the string V contains the parts that ensures the validity of the signature.

Moreover, from the term denoted by V one can derive the verification equations for the interactive protocol corresponding to a SPK scheme. That is, every component of V must then be equal to a component of the commitment. For instance, for our first building block we have the commitment t and $V = g^s y^c$ from which we derive the verification equation $t = g^s y^c$ for the corresponding interactive protocol. This is also how an SPK-signature can be computed: first the commitments are chosen, then, in order to compute the challenge c, the hash function is evaluated, whereby the concatenation of the commitments are included in the argument to the function instead of V. Finally, the responses are calculated such that the components of V are equal to the corresponding components of the included commitment.

The next building block is an SPK of a representation of a public key. The corresponding proof systems were first introduced in [CEvdG88].

Definition 3.2. *A $(k+1)$-tuple $(c, s_1, \ldots, s_k) \in \{0,1\}^\ell \times (\mathbb{Z}_q)^k$ satisfying*

$$c = \mathcal{H}(S\|V\|m) \;\; \textit{with } S = g_1\|\ldots\|g_k\|y \textit{ and } V = y^c \prod_{i=1}^{k} g_i^{s_i}$$

is an SPK of a representation of a group element y with respect to the bases $g_1, \ldots, g_k$ of the message $m \in \{0,1\}^*$. It is denoted by

$$SPK_2\{(\alpha_1, \ldots, \alpha_k) : y = \prod_{i=1}^{k} g_i^{\alpha_i}\}(m) \, .$$

Such a signature can be calculated as follows if values (secret keys) $x_1, \dots, x_k \in \mathbb{Z}_q$ are known such that $y = \prod_{i=1}^{k} g_i^{x_i}$ hold. The prover chooses the integers $r_1, \dots, r_k$ at random from $\mathbb{Z}_q$, computes

$$c := \mathcal{H}(g_1 \| \dots \| g_k \| y \| \prod_{i=1}^{k} g_i^{r_i} \| m),$$

and

$$s_i := r_i - c x_i \pmod{q} \qquad \text{for } i = 1, \dots, k.$$

3.3 Statements About Knowledge

Before we can define more general signature systems, we need some more notation. To express the fact that a public key y_i can be formed using only a subset of the generators $g_1, \dots, g_k$ we use a set $\mathcal{J}_i \subseteq \{1, \dots, k\}$ such that $y_i = \prod_{j \in \mathcal{J}_i} g_j{}^{x_{(i,j)}}$ holds, where $x_{(i,j)}$ are secret keys of the entity to which the public key y_i belongs. Note that the secret keys $x_{(i,j)}$ are in general not numbered consecutively but carry a tuple (i, j) where the first entry is the index of the public key y_i and the second entry is the index of the respective generator g_j. In particular, the secret keys $x_{(i,j)}$ for $j \notin \mathcal{J}_i$ are not defined. If it is clear that (i, j) is a pair of indices, we write x_{ij} instead of $x_{(i,j)}$.

Next we show how to prove the knowledge of a representation of several, say n, public keys at the same time. In principle this can be done by computing n separate signatures $SPK_2\{(\alpha_{ij})_{j \in \mathcal{J}_i} : y_i = \prod_{j \in \mathcal{J}_i} g_j^{\alpha_{ij}}\}(m)$. However, it is possible to merge these signatures by choosing the same challenge for all of them and thus making the resulting signature shorter.

Definition 3.3. *A* $(1 + \sum_{i=1}^{n} |\mathcal{J}_i|)$*-tuple* $(c, (s_{ij})_{i=1,\dots,n; j \in \mathcal{J}_i}) \in \{0,1\}^{\ell} \times (\mathbb{Z}_q)^{\sum_{i=1}^{n} |\mathcal{J}_i|}$ *satisfying*

$$c = \mathcal{H}(S \| V \| m)$$

with

$$S = g_1 \| \dots \| g_k \| y_1 \| \dots \| y_n \| \mathcal{J}_1 \| \dots \| \mathcal{J}_n$$

$$V = y_1^c \prod_{j \in \mathcal{J}_1} g_j^{s_{1j}} \| \dots \| \, y_n^c \prod_{j \in \mathcal{J}_n} g_j^{s_{nj}}$$

is a SPK of the representation of all the public keys $y_1, \dots, y_n$ to some of the bases $g_1, \dots, g_k$ of the message $m \in \{0,1\}^*$ and is denoted

$$SPK_3\{(\alpha_{ij})_{i=1,\dots,n;j\in\mathcal{J}_i} : \bigwedge_{i=1}^{n} y_i = \prod_{j\in\mathcal{J}_i} g_j^{\alpha_{ij}}\}(m)\,.$$

Computing such a signature is similar to simultaneously computing n separate signatures, each proving the knowledge of a representation of a y_i. The prover, knowing all x_{ij}'s, chooses r_{ij}'s ($i = 1, \dots, n; j \in \mathcal{J}_i$) at random from $\mathbb{Z}_q$ and computes the commitments $t_i = \prod_{j\in\mathcal{J}_i} g_j^{r_{ij}}$ for all $i = 1, \dots, n$. Then the challenge is evaluated as

$$c := \mathcal{H}(g_1\|\dots\|g_k\|y_1\|\dots\|y_n\|\mathcal{J}_1\|\dots\|\mathcal{J}_n\|t_1\|\dots\|t_n\|m)$$

and finally $s_{ij} := r_{ij} + cx_{ij} \pmod q$ are computed for ($i = 1, \dots, n; j \in \mathcal{J}_i$). It can easily be verified that the obtained tuple $(c, (s_{ij})_{i=1,\dots,n;j\in\mathcal{J}_i})$ is a valid signature $SPK_3\{(\alpha_{ij})_{i=1,\dots,n;j\in\mathcal{J}_i} : \bigwedge_{i=1}^{n} y_i = \prod_{j\in\mathcal{J}_i} g_j^{\alpha_{ij}}\}(m)$ of the message m.

Note that, by using this technique, it is always possible to merge several signatures to get a single shorter signature.

The next primitive makes it possible to prove the knowledge of a representation of one out of several public keys *without revealing which one*. This primitive was proposed by Cramer et al. [CDS94]. Of course, if the prover does not mind revealing the public key the representation of which she knows, then she can just point out this public key, say y_i, and prove her knowledge of a representation of it with the shorter signature $SPK_2\{(\alpha_{ij})_{j\in\mathcal{J}_i} : y_i = \prod_{j\in\mathcal{J}_i} g_i^{\alpha_{ij}}\}(\cdot)$.

Definition 3.4. *An* $(n + \sum_{i=1}^{n} |\mathcal{J}_i|)$*-tuple* $(c_1, \dots, c_n, (s_{ij})_{i=1,\dots,n;j\in\mathcal{J}_i}) \in (\{0,1\}^{\ell})^n \times (\mathbb{Z}_q)^{\sum_{i=1}^{n}|\mathcal{J}_i|}$ *satisfying*

$$\bigoplus_{i=1}^{n} c_i = \mathcal{H}(S\|V\|m)$$

with

$$S = g_1\|\dots\|g_k\|y_1\|\dots\|y_n\|\mathcal{J}_1\|\dots\|\mathcal{J}_n$$

$$V = y_1^{c_1}\prod_{j\in\mathcal{J}_1} g_j^{s_{1j}}\|\dots\|\,y_n^{c_n}\prod_{j\in\mathcal{J}_n} g_j^{s_{nj}}$$

is an SPK of the discrete logarithm of (at least) one y_i out of the list $\{y_1, \ldots, y_n\}$ to the base g of the message $m \in \{0,1\}^*$ and is denoted

$$SPK_4\{(\alpha_{ij})_{i=1,\ldots,n; j\in\mathcal{J}_i} : \bigvee_{i=1}^{n} y_i = \prod_{j\in\mathcal{J}_i} g_j^{\alpha_{ij}}\}(m).$$

The term $\bigoplus_{i=1}^{n} c_i$ in this definition denotes the bitwise XOR of all c_i's.

The idea behind this scheme is the fact that a single $SPK_2\{(\alpha_{ij})_{j\in\mathcal{J}_i} : y_i = \prod_{j\in\mathcal{J}_i} g_j^{\alpha_{ij}}\}(m)$ can be forged if the challenge is known before the computation of the commitment. The verification condition $\bigoplus_{i=1}^{n} c_i = \mathcal{H}(S\|V\|m)$ of the SPK_4 enables the prover to choose all but one of the c_i's. Thus she can forge all but one of the "partial signatures" each proving the knowledge of a representation of a y_i. It follows that at least for one y_i she must know a representation. Let us finally remark that it is not possible to distinguish between signatures SPK_4 that were computed using the knowledge of a representation of different public keys y_i. This is because the interactive protocol corresponding to the "partial signature" $SPK_2\{(\alpha_{ij})_{j\in\mathcal{J}_i} : y_i = \prod_{j\in\mathcal{J}_i} g_j^{\alpha_{ij}}\}(m)$ is honest-verifier zero-knowledge.

Let us describe how a signature $SPK_4\{(\alpha_{ij})_{i=1,\ldots,n; j\in\mathcal{J}_i} : \bigvee_{i=1}^{n} y_i = \prod_{j\in\mathcal{J}_i} g_j^{\alpha_{ij}}\}(m)$ can be computed by considering a concrete example. We assume that the representation of y_1 is known, say $y_1 = g_1^{x_{11}} g_4^{x_{14}}$. In this case, the prover chooses the integers $r_{11}, r_{14}, (s_{ij})_{i=2,\ldots,n; j\in\mathcal{J}_i}$ randomly from $\mathbb{Z}_q$ and $c_2, \ldots, c_n$ randomly from $\{0,1\}^\ell$ and computes

$$t_1 := g_1^{r_{11}} g_4^{r_{14}} \quad \text{and} \quad t_i := y_i^{c_i} \prod_{j\in\mathcal{J}_i} g_j^{s_{ij}} \text{ for } i = 2, \ldots, n.$$

Then she computes c_1, s_{11}, and s_{14} according to

$$c_1 := \mathcal{H}(g\| \ldots \|\mathcal{J}_n\|t_1\| \ldots \|t_n\|m) \oplus \bigoplus_{i=2}^{n} c_i$$

and

$$s_{11} := r_{11} - c_1 x_{11} \pmod q$$
$$s_{14} := r_{14} - c_1 x_{14} \pmod q$$

The resulting tuple $(c_1, \ldots, c_n, (s_{ij})_{i=1,\ldots,n; j\in\mathcal{J}_i})$ constitutes a valid signature $SPK_4\{(\alpha_{ij})_{i=1,\ldots,n; j\in\mathcal{J}_i} : \bigvee_{i=1}^{n} y_i = \prod_{j\in\mathcal{J}_i} g_j^{\alpha_{ij}}\}(m)$ as can easily be verified.

Let us now look at more complex statements. For instance consider the public keys y_1, y_2, y_3, and y_4 and the statement

$$(y_1 = g^{\alpha_1} \wedge y_2 = g^{\alpha_2}) \vee (y_2 = g^{\alpha_2} \wedge y_3 = g^{\alpha_3}) \vee (y_3 = g^{\alpha_3} \wedge y_4 = g^{\alpha_4}).$$

The goal is to prove the knowledge of values α_1, α_2, α_3, and α_4 such that this statement is true. In other words, the goal is to prove the knowledge of the discrete logarithms of all public keys in one of the sets $\{y_1, y_2\}$, $\{y_2, y_3\}$, or $\{y_3, y_4\}$. In principle, such a proof could be realized using the same technique as for the previous building block: combining the partial signatures

$$\begin{aligned}
&SPK_3\{(\alpha_1, \alpha_2) : y_1 = g^{\alpha_1} \ \wedge \ y_2 = g^{\alpha_2}\}(m)\\
&SPK_3\{(\alpha_2, \alpha_3) : y_2 = g^{\alpha_2} \ \wedge \ y_3 = g^{\alpha_3}\}(m)\\
&SPK_3\{(\alpha_3, \alpha_4) : y_3 = g^{\alpha_3} \ \wedge \ y_4 = g^{\alpha_4}\}(m)
\end{aligned}$$

by XOR-ing their challenges. However, this approach is not very efficient, since the more such subsets we have, the longer such a combined signature becomes.

Fortunately, Cramer et al. [CDS94] presented a much nicer solution for proving such statements. More exactly, it allows the construction of a system for proving the knowledge of a representation of all elements of one out of several defined subsets of the set of public keys $\mathcal{Y} = \{y_1, \dots, y_n\}$ without revealing the subset. To define such a signature system formally, let Γ denote a monotone set of subsets of $\{1, \dots, n\}$ (the indices of the public keys). Note that Γ is an access structure as it is known from secret sharing.

Definition 3.5. *A* $(n + \sum_{i=1}^{n} |\mathcal{J}_i|)$*-tuple* $(c_1, \dots, c_n, (s_{ij})_{i=1,\dots,n; j \in \mathcal{J}_i}) \in (\{0,1\}^\ell)^n \times (\mathbb{Z}_q)^{\sum_{i=1}^{n} |\mathcal{J}_i|}$ *satisfying*

$$\forall \mathcal{S} \in \Gamma^* : \quad \mathtt{rec}_{\Gamma^*}(\mathcal{S}, (c_i)_{i \in \mathcal{S}}) \ = \ \mathcal{H}(S \| V \| m)$$

with

$$\begin{aligned}
S &= g_1 \| \cdots \| g_k \| y_1 \| \cdots\cdots \| y_n \| \mathcal{J}_1 \| \cdots \| \mathcal{J}_n \| \Gamma\\
V &= y_1^{c_1} \prod_{j \in \mathcal{J}_1} g_j^{s_{1j}} \| \cdots \| y_n^{c_n} \prod_{j \in \mathcal{J}_n} g_j^{s_{nj}}
\end{aligned}$$

is an SPK of the representation of all $\{y_i | i \in \mathcal{S}\}$ with respect to the bases $\{g_j | j \in \mathcal{J}_i\}$ for at least one $\mathcal{S} \in \Gamma$, of the message $m \in \{0,1\}^*$. Such a signature is denoted by

$$SPK_5\Big\{(\alpha_{ij})_{i=1,\ldots,n; j\in\mathcal{J}_i} : \bigvee_{\mathcal{S}\in\Gamma}\Big(\bigwedge_{i\in\mathcal{S}} y_i = \prod_{j\in\mathcal{J}_i} g_j^{\alpha_{ij}}\Big)\Big\}(m).$$

Recall that Γ^* denotes the dual access structure of Γ and that $\texttt{rec}_{\Gamma^*}$ is an algorithm that reconstructs the secret if given all shares of a qualified set $\mathcal{S} \in \Gamma^*$, (cf. Section 2.11). If the size of the shares and the size of the commitments do not match, a suitable mapping must be introduced. In the following we assume that the shares have the same size as the c_i's.

The idea behind this signature system is basically the same as the one behind the scheme from Definition 3.4: combining the partial signatures $SPK_2\{(\alpha_{ij})_{j\in\mathcal{J}_i} : y_i = \prod_{j\in\mathcal{J}_i} g_j^{\alpha_{ij}}\}(m)$ by implying conditions on the challenges c_i. Here the conditions are obtained by interpreting the challenges also as shares of a secret sharing scheme for the access structure Γ^*. The secret is the value calculated by the hash function. Due to the relation between the access structures Γ and Γ^*, a set of 'shares' $\{c_j | j \notin \mathcal{S} \in \Gamma\}$ can always be completed to a full set of shares of a secret sharing scheme for Γ^*. Thus a prover knowing representations of all public keys in the set $\{y_i | i \in \mathcal{S}\}$ for some $\mathcal{S} \in \Gamma$ can choose all c_j's for which $j \in \overline{\mathcal{S}}$ and thus forge the partial proofs for these y_j's.

Let us now show in detail how such a signature can be computed. Without loss of generality, we assume that for some integer $u \le n$ the prover knows the representations of $y_1, \ldots, y_u$ and that $\mathcal{S} = \{1, \ldots, u\}$ is contained in Γ. The prover chooses the integers $(r_{ij})_{i=1,\ldots,u; j\in\mathcal{J}_i}$ and $(s_{ij})_{i=u+1,\ldots,n; j\in\mathcal{J}_i}$ randomly from $\mathbb{Z}_q$, picks $c_{u+1}, \ldots, c_n$ randomly from $\{0,1\}^\ell$, and computes

$$\sigma := \mathcal{H}\Big(g_1 \| \ldots \| \Gamma \| \prod_{j\in\mathcal{J}_1} g_j^{r_{1j}} \| \ldots \| \prod_{j\in\mathcal{J}_u} g_j^{r_{uj}} \| y_{u+1}^{c_{u+1}} \prod_{j\in\mathcal{J}_{u+1}} g_j^{s_{(u+1)j}} \| \ldots$$
$$\ldots \| y_n^{c_n} \prod_{j\in\mathcal{J}_n} g_j^{s_{nj}} \| m\Big),$$

$$\{c_1, \ldots, c_u\} := \texttt{cmpl}_{\Gamma^*}\big(\{1, \ldots, n\}, \{u+1, \ldots, n\}, \{c_{u+1}, \ldots, c_n\}, \sigma\big),$$

and

$$s_{ij} := r_{ij} - c_i x_{ij} \pmod{q} \qquad \text{for } i = 1, \ldots, u; j \in \mathcal{J}_i\,.$$

The algorithm $\mathtt{cmpl}_{\Gamma^*}$ completes the set $\{c_{u+1}, \dots, c_n\}$ to a full set of shares for the secret σ according to the access structure Γ^* (cf. Section 2.11).

3.4 Proving the Equality of Secret Keys

In the previous section we have seen how to prove different statements about the knowledge of discrete logarithms and representations. An interesting extension is to prove not only the knowledge of secret keys but also that certain relations among them hold. The remainder of this chapter is devoted to presenting different methods for proving statements about the knowledge of discrete logarithms and representations, and, simultaneously, statements about relations among them.

We start by considering we consider methods for proving the equality of secret keys. The simplest non-trivial case is a signature proving that the discrete logarithms of two public keys with respect to two different bases are equal. Such a scheme was introduced in [CP93] in the context of blind signature schemes.

Definition 3.6. *A pair $(c, s) \in \{0,1\}^\ell \times \mathbb{Z}_q$ satisfying*

$$c = \mathcal{H}(S\|V\|m) \text{ with } S = g\|h\|y\|z \text{ and } V = g^s y^c \| h^s z^c$$

is a signature of the message $m \in \{0,1\}^*$ based on a proof of knowledge and of equality of the discrete logarithm of z with respect to the base h and of the discrete logarithm of y with respect to the base g. It is denoted by

$$SPK_6\{(\alpha) : y = g^\alpha \wedge z = h^\alpha\}(m).$$

An $SPK_6\{\alpha : y = g^\alpha \wedge z = h^\alpha\}(m) = (c, s)$ can be computed as follows if a value $x \equiv \log_g y \pmod{q}$ is known and if $\log_g y \equiv \log_h z \pmod{q}$ holds. One chooses a random integer r from $\mathbb{Z}_q$ and computes $t_1 := g^r$ and $t_2 := h^r$. Then, c and s are calculated according to

$$c := \mathcal{H}(g\|h\|y\|z\|t_1\|t_2\|m)$$

and

$$s := r - cx \pmod{q}.$$

A signature of the type SPK_6 can be seen as two "parallel" signatures $SPK_1\{(\alpha) : y = g^{\alpha}\}(m)$ and $SPK_1\{(\alpha) : z = h^{\alpha}\}(m)$ where the exponent for the commitments, the challenges, and the responses are the same. This technique can be generalized to representations: whenever two responses s_i are equal, the respective elements of the representation(s) are equal, too. This is formalized in the next definition. Note that, when two responses are different it does not imply that the respective elements of the representation(s) are also different, since if the signer/prover chooses different exponents in the commitments, then the responses are different regardless of the value of the secret key.

Definition 3.7. *A* $(u+1)$ *tuple* $(c, s_1, \dots, s_u) \in \{0,1\}^{\ell} \times (\mathbb{Z}_q)^u$ *satisfying*

$$c = \mathcal{H}(S \| V \| m)$$

with

$$S = g_1 \| \dots \| g_k \| y_1 \| \dots \| y_n \| \mathcal{J}_1 \| \dots \| \mathcal{J}_n \| \{e_{ij}\}_{i=1,\dots,n; j \in \mathcal{J}_i}$$

$$V = y_1^c \prod_{j \in \mathcal{J}_1} g_j^{s_{e_{1j}}} \| \dots \| y_n^c \prod_{j \in \mathcal{J}_n} g_j^{s_{e_{nj}}}$$

is a signature of the message $m \in \{0,1\}^*$ *based on a proof of knowledge of representations of* $y_1, \dots, y_n$ *with respect to some of the bases* $g_1, \dots, g_k$ *and that, additionally, some of the elements of the representations are equal. It is denoted*

$$SPK_7\left\{(\alpha_1, \dots, \alpha_u) : \left(y_1 = \prod_{j \in \mathcal{J}_1} g_j^{\alpha_{e_{1j}}}\right) \wedge \dots \wedge \left(y_n = \prod_{j \in \mathcal{J}_n} g_j^{\alpha_{e_{nj}}}\right)\right\}(m),$$

where the indices $e_{ij} \in \{1, \dots, u\}$ *refer to the secrets* $\alpha_1, \dots, \alpha_u$ *and the elements of* $\mathcal{J}_i$ *are indices referring to the base elements* $g_1, \dots, g_k$.

In this definition, we made an exception of our convention for indexing the secret key. Here, they are indexed as $x_1, \dots x_u$. This is necessary since in this case a secret key can appear several times in the representations of the public keys.

To clarify Definition 3.7 let us consider, as an example, the signature $SPK_7\{(\alpha, \beta, \gamma) : y = g_1^{\alpha} g_2^{\beta} g_3^{\alpha} \wedge z = g_1^{\beta} g_4^{\gamma}\}$. Such a signature proves the knowledge of the representation of y with respect to the bases g_1, g_2, and g_3 and of z to the bases g_1 and g_4. Additionally, it proves that the g_1-part and the g_3-part of the representation of y are equal and, finally, that the g_1-part in the representation of z equals the g_2-part in the representation of y.

An SPK_7 can be computed if a u-tuple $(x_1, \ldots, x_u)$ is known that satisfies the considered statement. One first chooses $r_i \in_R \mathbb{Z}_n$ for $i = 1, \ldots, u$, computes c as

$$c := \mathcal{H}\left(g_1 \| \ldots \| \{e_{ij}\}_{i=1,\ldots,n; j \in \mathcal{J}_i} \| \prod_{j \in \mathcal{J}_1} g_j^{r_{e_{1j}}} \| \ldots \| \prod_{j \in \mathcal{J}_n} g_j^{r_{e_{nj}}} \| m\right),$$

and then sets $s_i := r_i - cx_i \pmod{n}$ for $= 1, \ldots, u$.

The approach of this section, namely requiring some of the s_i's to be equal, can be generalized to requiring the s_i's to satisfy some modular relations modulo q. Then it can be concluded that the corresponding secret keys also satisfy these relations. However, this works only when all challenges are the same. Thus, proving statements that include also the $\vee$-connective is not possible since the challenges might be different. However, in [CS97b] it is shown how this approach can be used nevertheless, when the statement to prove is transformed into new one having a special form. Unfortunately, the size of the new statement is exponential in the size of the old statement.

In the next section we describe another, more efficient, method to prove such statements in a more direct way.

3.5 Proving Polynomial Relations Among Secret Keys

3.5.1 Linear Relations

As a start, let us restrict ourselves to linear relations among the secret keys. When the secret keys are discrete logarithms with respect to a single base, it is trivial to prove that a linear relation among them holds. It does not even require the knowledge of the secret keys. As an example, consider the equation $3x_1 + 5x_2 \equiv 6 \pmod{q}$ and let y and z be two public keys. Then, verifying whether the equation holds for the secret keys $x_1 := \log_g y$ and $x_2 := \log_g z$ can be done by computing $y^3 z^5$ and checking whether the result equals g^6.

When the bases of the discrete logarithms are different or the secret keys are elements of representations, a linear relation among them can not in general be verified in such a way. It is only possible if the prover provides some additional public keys such that each secret key is the discrete logarithm of one of these with respect to a single base (g in the above example). However, these additional public keys leak knowledge about the secret keys that was not available before. For instance an equality of two secret keys could then easily be recognized. To circumvent this, these additional public keys can be randomized using an additional base, say h, whose discrete logarithm with respect to any other base used must not be known. We illustrate the resulting method by continuing our example. Let y_1 and y_2 be the public keys and g_1 and g_2 the bases of the logarithms. Again, the goal is to show that $3\log_{g_1} y_1 + 5\log_{g_2} y_2 \equiv 6 \pmod{q}$ holds. Different from before, the entity (the prover) knowing the secret keys $x_1 := \log_{g_1} y_1$ and $x_2 := \log_{g_2} y_2$ must be involved: she chooses two random integers r_1 and r_2 from $\mathbb{Z}_q^*$, computes the additional public keys

$$\tilde{y}_1 := h^{r_1} g^{x_1} \text{ and } \tilde{y}_2 := h^{r_2} g^{x_2} ,$$

and provides the three signatures

$$\begin{aligned}
U_1 &:= SPK_7\{(\alpha,\gamma) : y_1 = g_1^{\alpha} \wedge \tilde{y}_1 = h^{\gamma} g^{\alpha}\},\\
U_2 &:= SPK_7\{(\beta,\delta) : y_2 = g_2^{\beta} \wedge \tilde{y}_2 = h^{\delta} g^{\beta}\}, \text{ and}\\
U_3 &:= SPK_1\{(\varepsilon) : \tilde{y}_1^3 \tilde{y}_2^5 / g^6 = h^{\varepsilon}\}.
\end{aligned}$$

The signatures U_1 and U_2 prove that $\tilde{y}_1$ and $\tilde{y}_2$ correspond to y_1 and y_2, i.e., that the secret keys are the exponents of the g part of the representation of the $\tilde{y}_i$'s. The signature U_3 proves that the considered linear equation among the secret keys holds. More precisely, it proves that the g-part of the representation of $\tilde{y}_1^3 \tilde{y}_2^5 / g^6$ with respect to g and h vanishes, which is only possible if the considered relation holds (otherwise the prover could compute the discrete logarithm of g with respect to the base h).

In some sense, the signature U_3 also proves the knowledge of a discrete logarithm of a public key, namely the public key $\tilde{y}_1^3 \tilde{y}_2^5 / g^6$ that is constructed according to the considered linear equation. Hence this method can be seen as a way of transforming a relation into a public key the discrete logarithm of which can only be known if the relation holds.

3.5.2 Polynomial Relations

Before we can define an SPK for proving that modular relations among the secret keys hold, we have to find a way to apply the approach in the previous section to polynomial relations. Let us explain how this is achieved by continuing the example from the previous section and considering the polynomial $2x_1^4 + x_2 \equiv 3 \pmod{q}$. To construct a public key that corresponds to this polynomial, we need two additional public keys, one with x_1^4 as exponent, the other with x_2 as exponent. For x_2 this is not different from the previous paragraph. However, for x_1^4 this is different, since in this case we have to show that one exponent of the additional public key equals the 4-th power of the discrete logarithm of y_1. One way to do this is by so-called bit-wise proofs (for an example of such proofs we refer to Section 5.3). Unfortunately such proofs are not very efficient. However, if the degree of the polynomial is not too large, it can be shown more efficiently.

First we observe that it is possible to show that the discrete logarithm of a public key, say z_3, is the product of the discrete logarithms of two other public keys, say z_1 and z_2. Namely, the signature

$$SPK_6\{(\alpha,\beta) : z_1 = g^\alpha \wedge z_2 = g^\beta \wedge z_3 = z_2^\alpha\}(m)$$

proves this, as can easily be seen.

With this observation, we can continue our example. To show that $2x_1^4 + x_2 \equiv 3 \pmod{q}$ holds, the prover has to provide the values

$$\tilde{y}_{11}{:=}h^{r_{11}}g^{x_1},\ \tilde{y}_{12}{:=}h^{r_{12}}g^{x_1^2},\ \tilde{y}_{13}{:=}h^{r_{13}}g^{x_1^3},\ \text{ and } \tilde{y}_{14}{:=}h^{r_{14}}g^{x_1^4},$$

where the r_{11}, r_{12}, r_{13}, and r_{14} are randomly chosen from $\mathbb{Z}_q$ and with $x_1 \equiv log_{g_1} y_1 \pmod{q}$. We observe that

$$\tilde{y}_{12} = h^{v_1}\tilde{y}_{11}^{x_1},\ \tilde{y}_{13} = h^{v_2}\tilde{y}_{12}^{x_1},\ \text{ and } \tilde{y}_{14} = h^{v_3}\tilde{y}_{13}^{x_1}$$

holds for some values v_1, v_2, and v_3. Therefore the signature

$$U_1{:=}SPK_7\{(\alpha,\beta,\gamma,\delta,\varepsilon,) \ :\ y_1 = g_1^\alpha \ \wedge\ \tilde{y}_{11} = h^\beta g^\alpha \ \wedge \\ \tilde{y}_{12} = h^\gamma \tilde{y}_{11}^\alpha \ \wedge\ \tilde{y}_{13} = h^\delta \tilde{y}_{12}^\alpha \ \wedge\ \tilde{y}_{14} = h^\varepsilon \tilde{y}_{13}^\alpha\}$$

proves that the g-part in the representation of $\tilde{y}_{14}$ is indeed the 4-th power of the discrete logarithm of y_1 with respect to g_1. Thus, U_1,

$$U_2{:=}SPK_7\{(\zeta,\eta) \ :\ y_2 = g_2^\zeta \ \wedge\ \tilde{y}_2 = h^\eta g^\zeta\},\ \text{and}$$

$$U_3 := SPK_1\{(\vartheta) : \tilde{y}_{14}^2 \tilde{y}_2 / g^3 = h^{\vartheta}\}$$

prove that $2(\log_{g_1} y_1)^4 + \log_{g_2} y_2 \equiv 3 \pmod{q}$ holds. In particular, the signature U_3 proves knowledge of the discrete logarithm of the "public key corresponding to the considered equation".

This technique was already used in [CS97a] and similar techniques have been used in [Dam94, Dam95, Oka97, FO97] with commitment schemes.

Remark 3.1. Instead of providing the additional public keys for all $e-1$ exponents $(x_1, x_1^2, x_1^3, \ldots, x_1^{e-1})$, it would suffice to provide only the keys for $(x_1, x_1^2, x_1^4, x_1^8 \ldots)$, and then prove that x^e can be constructed from these according to the square-and-multiply algorithm (cf. Section 2.2.3). Furthermore, there exists a trade-off between bit-wise proof and the techniques described here: if the exponents become large, bitwise proofs are more efficient. This tradeoff depends on the polynomial that is considered. However, we do not consider this remark in the following definitions.

We have now developed all techniques for defining our next building block. This requires some more notation. Let us therefore redraw the scenario. Given are n public keys, u polynomials, and the goal, which is to prove knowledge of secret keys that are the representations of the n public keys and that, additionally, these secret keys satisfy the u polynomials. More technically, we are given the generators $g, h, g_1, \ldots, g_k$, the public keys

$$y_1 = \prod_{j \in \mathcal{J}_1} g_j^{x_{1j}}, \ldots, y_n = \prod_{j \in \mathcal{J}_n} g_j^{x_{nj}}$$

and the u polynomials

$$\begin{aligned} b_1 &\equiv a_{11}\alpha_1^{e_{11}} + a_{12}\alpha_2^{e_{12}} + \ldots + a_{1w}\alpha_w^{e_{1w}} \pmod{q} \\ &\vdots \\ b_u &\equiv a_{u1}\alpha_1^{e_{u1}} + a_{u2}\alpha_2^{e_{u2}} + \ldots + a_{uw}\alpha_w^{e_{uw}} \pmod{q} \end{aligned}$$

in $\alpha_1, \ldots, \alpha_w$, where w denotes the number of secret keys $w = \sum_{i=1}^{n} |\mathcal{J}_i|$. We require that for all i and j ($i = 1, \ldots, u; j = 1, \ldots, w$) it holds that if $a_{ij} = 0$ then $e_{ij} = 0$, and if $a_{ij} \neq 0$ then[1] $e_{ij} > 0$. Furthermore, we

[1] The case of negative exponents could be included, too. Then, the additional public keys $\tilde{y}_{ij}$'s must also be provided for the negative exponents. This can be done using the fact $\log_g y \equiv (\log_y g)^{-1} \pmod{q}$.

need a one-to-one mapping from the secret keys to the w variables of the polynomials: let $\mathtt{in}(\cdot)$ denote such a mapping from $\{1,\ldots,w\}$ to the indices $\{1,\ldots,n\} \times (\mathcal{J}_1 \cap \ldots \cap \mathcal{J}_n)$ of the secret keys. Finally, let $\hat{e}_j$ denote $\max\{e_{1j},\ldots,e_{uj}\}$ for $j = 1,\ldots,w$.

Definition 3.8. *A* $(2\sum_{i=1}^{w} \hat{e}_i + 1 + w + u)$ *tuple* $(\tilde{y}_{11},\ldots,\tilde{y}_{1\hat{e}_1},\tilde{y}_{21},\ldots,$ $\tilde{y}_{w\hat{e}_w}, c, (s_{ij})_{i=1,\ldots,n;j\in\mathcal{J}_i}, (\tilde{s}_{ij})_{i=1,\ldots,w;j=1,\ldots,\hat{e}_i}, \tilde{r}_1,\ldots,\tilde{r}_u) \in G^{\sum_{i=1}^{w}\hat{e}_i} \times \{0,1\}^{\ell} \times$ $(\mathbb{Z}_q)^{w+\sum_{i=1}^{w}\hat{e}_i+u}$ *satisfying*

$$c = \mathcal{H}\left(S_1 \| S_2 \| V_1 \| V_2 \| V_3 \| m\right)$$

with

$$\begin{aligned}
S_1 &= g_1 \| \ldots \| g_k \| g \| h \| y_1 \| \ldots \| y_n \| \tilde{y}_{11} \| \ldots \| \tilde{y}_{1\hat{e}_1} \| \tilde{y}_{21} \| \ldots \| \tilde{y}_{w\hat{e}_w} \| \\
S_2 &= \mathcal{J}_1 \| \ldots \| \mathcal{J}_n \| \mathtt{in} \| g^{-b_1} \textstyle\prod_{j=1}^{w} \tilde{y}_{je_{1j}}^{a_{1j}} \| \ldots \| \, g^{-b_u} \textstyle\prod_{j=1}^{w} \tilde{y}_{je_{uj}}^{a_{uj}} \\
V_1 &= y_1^c \prod_{j\in\mathcal{J}_1} g_j^{s_{1j}} \| \ldots \| \, y_n^c \prod_{j\in\mathcal{J}_n} g_j^{s_{nj}} \\
V_2 &= \tilde{y}_{11}^c h^{\tilde{s}_{11}} g^{s_{\mathtt{in}(1)}} \| \ldots \| \, \tilde{y}_{1\hat{e}_1}^c h^{\tilde{s}_{1\hat{e}_1}} \tilde{y}_{1(\hat{e}_1-1)}^{s_{\mathtt{in}(1)}} \| \tilde{y}_{21}^c h^{\tilde{s}_{21}} g^{s_{\mathtt{in}(2)}} \| \ldots \\
&\qquad \ldots \| \, \tilde{y}_{w\hat{e}_w}^c h^{\tilde{s}_{w\hat{e}_w}} \tilde{y}_{w(\hat{e}_w-1)}^{s_{\mathtt{in}(w)}} \\
V_3 &= \left(g^{-b_1} \prod_{j=1}^{w} \tilde{y}_{je_{1j}}^{a_{1j}}\right)^c h^{\tilde{r}_1} \| \ldots \| \left(g^{-b_u} \prod_{j=1}^{w} \tilde{y}_{je_{uj}}^{a_{uj}}\right)^c h^{\tilde{r}_u}
\end{aligned}$$

is a signature of the message $m \in \{0,1\}^*$ *based on a proof*

- *of knowledge of the representations of* y_i *with respect to the bases* $\{g_j | j \in \mathcal{J}_i\}$ *for all* $i = 1,\ldots n$, *and*
- *that, additionally, the secret keys* $\alpha_{\mathtt{in}(1)},\ldots,\alpha_{\mathtt{in}(w)}$ *satisfy the* u *polynomials*

$$b_i \equiv \sum_{j=1}^{w} a_{ij}\alpha_{\mathtt{in}(j)}^{e_{ij}} \pmod{q} \text{ for } i = 1 \ldots u.$$

It is denoted by

$$SPK_8\Big\{(\alpha_{ij})_{i=1,\ldots,n;j\in\mathcal{J}_i} : \big(y_1 = \prod_{j\in\mathcal{J}_1} g_j^{\alpha_{1j}}\big) \wedge \ldots \wedge \big(y_n = \prod_{j\in\mathcal{J}_n} g_j^{\alpha_{nj}}\big) \wedge$$
$$\big(b_1 \equiv \sum_{j=1}^{w} a_{1j}\alpha_{\mathtt{in}(j)}^{e_{1j}} \pmod{q}\big) \wedge \ldots \wedge \big(b_u \equiv \sum_{j=1}^{w} a_{uj}\alpha_{\mathtt{in}(j)}^{e_{uj}} \pmod{q}\big)\Big\}(m).$$

Let us consider the different parts of the argument of the hash function in this definition. The S_1-part is the usual inclusion of the bases and public keys. In the S_2-part the statement that is proven is committed, where $\mathtt{in}$ denotes a description of the mapping $\mathtt{in}(\cdot)$. The V_1-part assures that the representations of the public keys $y_1, \ldots, y_n$ are known, the V_2-part guarantees that the $\tilde{y}_i$'s encode the secret keys and their powers, and finally, the V_3-part assures that the u polynomial relations hold.

Remark 3.2. To improve the efficiency of the SPK_8 (and of the SPK_9) they can be modified to use addition chains[2] [Knu81] for the $\hat{e}_i$'s. The chain for $\hat{e}_i$ must of course contain $e_{1i}, \ldots, e_{ui}$. Then the prover would have to provide only the $\tilde{y}_{ij}$'s for which j is an element of the addition chain for $\hat{e}_i$. This induces of course that the relations among the $\tilde{y}_{ij}$'s must be proved in a slightly different way. To illustrate this, consider the addition chain $(1,2,3,6)$ for $\hat{e}_1 = 6$ and let $y_1 := g_1^{x_1}$. Then

$$U_1 := SPK_7\{(\alpha,\beta,\gamma,\delta,\varepsilon,\zeta,\vartheta) : y_1 = g_1^{\alpha} \wedge \tilde{y}_{11} = h^{\gamma} g^{\alpha} \wedge \\ \tilde{y}_{12} = h^{\delta} \tilde{y}_{11}^{\alpha} \wedge \tilde{y}_{13} = h^{\varepsilon} \tilde{y}_{12}^{\alpha} \wedge \tilde{y}_{13} = h^{\zeta} g^{\beta} \wedge \tilde{y}_{16} = h^{\vartheta} \tilde{y}_{13}^{\beta}\}$$

proves that the exponents in the g-part of the $\tilde{y}_{1j}$'s are x_1 powered to j (note that $\beta \equiv \alpha^3 \pmod{q}$).

Theorem 3.1. *The interactive protocol corresponding to the SPK_8 is honest-verifier zero-knowledge and a proof of knowledge of the representations $(\alpha_{ij})_{j \in \mathcal{J}_i}$ of all $y_i, i = 1, \ldots, n$, with respect to the bases g_j for $j \in \mathcal{J}_i$ and the fact that these representations satisfy the u relations*

$$b_i \equiv \sum_{j=1}^{w} a_{ij} \alpha_{\mathtt{in}(j)}^{e_{ij}} \pmod{q} \ \textit{for}\ i = 1 \ldots u.$$

Proof (sketch). Honest-verifier zero-knowledge: We show how a communication by the prover with an honest verifier can be simulated. First choose the integers $(s_{ij})_{i=1,\ldots,n; j \in \mathcal{J}_i}$, $(\tilde{s}_{ij})_{i=1,\ldots,w; j=1,\ldots,\hat{e}_i}, \tilde{r}_1, \ldots, \tilde{r}_u$ at random from $\mathbb{Z}_q$ and c randomly from $\{0,1\}^{\ell}$. Furthermore, choose all $(\tilde{y}_{ij})_{i=1,\ldots,w; j=1,\ldots,\hat{e}_i}$ randomly from G. Then all commitments, i.e., all

[2] An addition chain for an integer n is an ascending sequence $1 = a_0 < a_1 < \ldots < a_r = n$ such that for $1 \leq i \leq r$ we have $a_i = a_j + a_k$ for some j and $k, 0 \leq k \leq j < i$.

the values that are included in the V_1-, V_2-, and V_3-part of the argument of the hash function (see Definition 3.8), can be computed (e.g., $t_1 = y_1^c \prod_{j \in \mathcal{J}_1} g_j^{s_{1j}}$). These commitments, the challenge c, and the response-tuple $((s_{ij})_{i=1,\dots,n;j\in\mathcal{J}_i}, (\tilde{s}_{ij})_{i=1,\dots,w;j=1,\dots,\hat{e}_i}, \tilde{r}_1, \dots, \tilde{r}_u)$ constitute a simulated protocol view that is statistically independent from the view of a protocol run of an honest verifier with the real prover and hence the protocol is perfect honest-verifier zero-knowledge.

Proof of knowledge: Here we show only that from two views of the protocols with the same commitments one can compute the secret keys and that these satisfy the modular relations. The rest of the construction of a knowledge extractor is similar to that given in Section 2.9 for the Schnorr identification protocol.

Assume that we are given the views of two runs of the interactive protocol with the same commitments but with different challenges c and $\dot{c}$ and the responses

$$(s_{ij})_{i=1,\dots,n;j\in\mathcal{J}_i}, (\tilde{s}_{ij})_{i=1,\dots,w;j=1,\dots,\hat{e}_i}, \tilde{r}_1, \dots, \tilde{r}_u, \text{ and}$$
$$(\dot{s}_{ij})_{i=1,\dots,n;j\in\mathcal{J}_i}, (\dot{\tilde{s}}_{ij})_{i=1,\dots,w;j=1,\dots,\hat{e}_i}, \dot{\tilde{r}}_1, \dots, \dot{\tilde{r}}_u.$$

We now have to show that from these two views it is possible to compute values that are representations of the y_i's and that they satisfy all u relations.

From these challenges and responses we get the equalities

$$y_i^c \prod_{j\in\mathcal{J}_i} g_j^{s_{ij}} = y_i^{\dot{c}} \prod_{j\in\mathcal{J}_i} g_j^{\dot{s}_{ij}} \qquad \text{for } i = 1, \dots, n$$

and thus can compute the values

$$x_{ij} := \frac{\dot{s}_{ij} - s_{ij}}{c - \dot{c}} \pmod q \qquad \text{for } i = 1, \dots, n; j \in \mathcal{J}_i$$

such that

$$y_i = \prod_{j\in\mathcal{J}_i} g_j^{x_{ij}} \qquad \text{for } i = 1, \dots, n$$

holds. Similarly, from the $\tilde{s}_{ij}$'s and the $\dot{\tilde{s}}_{ij}$'s we can compute values r_{ij} such that

$$\tilde{y}_{ij} = h^{r_{ij}} g^{x^j_{\mathtt{in}(i)}} \qquad \text{for } i = 1, \dots, w, j = 1, \dots, \hat{e}_i$$

holds. It remains to show that the x_{ij}'s we have computed actually satisfy the u modular relations. Consider the values $\hat{y}_i$ that are computed according to the u given polynomials, i.e., computed as

$$\hat{y}_i := g^{-b_i} \prod_{j=1}^{w} \tilde{y}_{je_{ij}}^{a_{ij}} = g^{-b_i} h^{\sum_{j=1}^{w} a_{ij} r_{(j,e_{ij})}} g^{\sum_{j=1}^{w} a_{ij} x_{\mathtt{in}(j)}^{e_{ij}}} \text{ for } i = 1, \ldots, u,$$

where in the second term the $\tilde{y}_{ij}$'s are replaced by their representation with respect to g and h. Recall that the g-part of a value $\hat{y}_i$ vanishes if the x_{ij}'s satisfy i-th polynomial. From the two views one can also compute values $v_1 := (\dot{\tilde{r}}_1 - \tilde{r}_1)/(c - \dot{c}) \pmod{q}, \ldots, v_u := (\dot{\tilde{r}}_v - \tilde{r}_v)/(c - \dot{c}) \pmod{q}$ such that $\hat{y}_i = h^{v_i}$ holds for $i = 1, \ldots, u$. Therefore, if for some $\hat{y}_i$ the g-part does not vanish, i.e., if $\sum_{j=1}^{w} a_{ij} x_{\mathtt{in}(j)}^{e_{ij}}$ is not equal to b_i modulo q, then one can compute the discrete logarithm of h to the base g using the two different representations of this $\hat{y}_i$. Since this is assumed to be infeasible, the x_{ij}'s must satisfy all u relations. □

The following example shows how such a signature can be computed. Let $y_1 = g_1^{x_{11}} g_2^{x_{12}}$ be a public key and let $x_{11}^2 + 4x_{12}^3 + x_{12} \equiv 7 \pmod{q}$ hold for the secret keys x_{11} and x_{12}. Then a signature

$$SPK_8\{(\alpha, \beta) : \; y_1 = g_1^{\alpha} g_2^{\beta} \;\wedge\; (\alpha^2 + 4\beta^3 + \beta \equiv 7 \pmod{q})\;\}(m)$$

of the message m can be computed as follows. First calculate the additional public keys by choosing $v_{11}, v_{12}, v_{21}, v_{22}$, and v_{23} at random from $\mathbb{Z}_q^*$ and compute

$$\tilde{y}_{11} := h^{v_{11}} g^{x_{11}}, \; \tilde{y}_{12} := h^{v_{12}} \tilde{y}_{11}^{x_{11}},$$
$$\tilde{y}_{21} := h^{v_{21}} g^{x_{12}}, \; \tilde{y}_{22} := h^{v_{22}} \tilde{y}_{21}^{x_{12}}, \text{ and } \tilde{y}_{23} := h^{v_{23}} \tilde{y}_{22}^{x_{12}}.$$

Then choose $r_1, r_2, r_{11}, r_{12}, r_{21}, r_{22}, r_{23}$, and u at random from $\mathbb{Z}_q^*$ and compute

$$t_1 := g_1^{r_1} g_2^{r_2},$$
$$t_{11} := h^{r_{11}} g^{r_1}, \; t_{12} := h^{r_{12}} \tilde{y}_{11}^{r_1},$$
$$t_{21} := h^{r_{21}} g^{r_2}, \; t_{22} := h^{r_{22}} \tilde{y}_{21}^{r_2}, \; t_{23} := h^{r_{23}} \tilde{y}_{22}^{r_2},$$
$$t_h := h^{u},$$

and

$$c := \mathcal{H}\big(g_1 \| g_2 \| g \| h \| y_1 \| \tilde{y}_{11} \| \tilde{y}_{12} \| \tilde{y}_{21} \| \tilde{y}_{22} \| \tilde{y}_{23} \| \{1,2\} \| \{11,12\} \| \\ \tilde{y}_{12}\, \tilde{y}_{23}^4\, \tilde{y}_{21} / g^7 \| t_1 \| t_{11} \| t_{12} \| t_{21} \| t_{22} \| t_{23} \| t_h \| m\big),$$

where $\{11,12\}$ is the description of the mapping $\mathtt{in}(\cdot)$. Finally, one can compute

$$\begin{aligned} &s_{11} := r_1 - c x_{11} \pmod{q},\ s_{12} := r_2 - c x_{12} \pmod{q}, \\ &\tilde{s}_{11} := r_{11} - c v_{11} \pmod{q},\ \tilde{s}_{12} := r_{12} - c v_{12} \pmod{q}, \\ &\tilde{s}_{21} := r_{21} - c v_{21} \pmod{q},\ \tilde{s}_{22} := r_{22} - c v_{22} \pmod{q}, \\ &\tilde{s}_{23} := r_{23} - c v_{23} \pmod{q}, \text{ and} \\ &\tilde{r}_1 := u - c((v_{12} + x_{11} v_{11}) + 4(v_{23} + x_{21} v_{22} + x_{21}^2 v_{21}) + v_{21}) \pmod{q} \end{aligned}$$

and thus gets the tuple

$$(\tilde{y}_{11}, \tilde{y}_{12}, \tilde{y}_{21}, \tilde{y}_{22}, \tilde{y}_{23}, c, s_{11}, s_{12}, \tilde{s}_{11}, \tilde{s}_{12}, \tilde{s}_{21}, \tilde{s}_{22}, \tilde{s}_{23}, \tilde{r}_1)$$

which constitutes an $SPK_8\{(\alpha,\beta) :\ y_1 = g_1^{\alpha} g_2^{\beta}\ \wedge\ (\alpha^2 + 4\beta^3 + \beta = 7 \pmod{q})\ \}(m)$. This is because the following equations hold:

$$\begin{aligned} &t_1 = y_1^c g_1^{s_{11}} g_2^{s_{12}} \\ &t_{11} = \tilde{y}_{11}^c h^{\tilde{s}_{11}} g^{s_{11}},\ t_{12} = \tilde{y}_{12}^c h^{\tilde{s}_{12}} \tilde{y}_{11}^{s_{11}}, \\ &t_{21} = \tilde{y}_{21}^c h^{\tilde{s}_{21}} g^{s_{12}},\ t_{22} = \tilde{y}_{22}^c h^{\tilde{s}_{22}} \tilde{y}_{21}^{s_{12}},\ t_{23} = \tilde{y}_{23}^c h^{\tilde{s}_{23}} \tilde{y}_{22}^{s_{12}}, \text{ and} \\ &t_h = \left(\frac{\tilde{y}_{12}\, \tilde{y}_{23}^4\, \tilde{y}_{21}}{g^7}\right)^c h^{\tilde{r}_1}. \end{aligned}$$

This technique of relating the validity of a relation among secret keys to the knowledge of a discrete logarithm allows the application of the method of [CDS94] (see SPK_5) for proving statements about the knowledge of secret keys and about relations among them that also contain $\vee$-connectives. For the sake of an easier notation, let $\hat{y}_1, \ldots, \hat{y}_u$ denote the public keys that are constructed according to the polynomial relations, i.e.,

$$\hat{y}_i := g^{-b_i} \prod_{j=1}^{w} \tilde{y}_{je_{ij}}^{a_{ij}} \qquad \text{for } i = 1, \ldots, u$$

Note that a verifier can compute $\hat{y}_1, \ldots, \hat{y}_u$ from the additional public keys $\tilde{y}_{ij}$, thus the $\hat{y}_i$'s need not be part of the signature.

Using these $\hat{y}_i$'s, we can now formulate statements about knowledge of representations and about relations among the secret keys as we did in the case of SPK_5: Let Γ be a set of subsets of $\{1, \ldots, n, n+1, \ldots, n+u\}$, where $1, \ldots, n$ correspond to $y_1, \ldots, y_n$ and $n+1, \ldots, n+u$ correspond to $\hat{y}_1, \ldots, \hat{y}_u$.

Definition 3.9. *A* $(2\sum_{i=1}^{w} \hat{e}_i + n + w + 2u)$ *tuple* $(\tilde{y}_{11}, \ldots, \tilde{y}_{1\hat{e}_1}, \tilde{y}_{21}, \ldots, \tilde{y}_{w\hat{e}_w}, c_1, \ldots c_{n+u}, (s_{ij})_{i=1,\ldots,n; j\in\mathcal{J}_i}, (\tilde{s}_{ij})_{i=1,\ldots,w; j=1,\ldots,\hat{e}_i}, \tilde{r}_1, \ldots, \tilde{r}_u) \in G^{\sum_{i=1}^{w}\hat{e}_i} \times (\{0,1\}^{\ell})^{n+u} \times (\mathbb{Z}_q)^{w+\sum_{i=1}^{w}\hat{e}_i+u}$ *satisfying*

$$\forall \mathcal{S}' \in \Gamma^* : \quad \mathtt{rec}_{\Gamma^*}(\mathcal{S}', (c_i)_{y_i\in\mathcal{S}'}) = \mathcal{H}(S_1 \| S_2 \| V_1 \| V_2 \| V_3 \| m)$$

with

$$\begin{aligned}
S_1 &= g_1 \| \ldots \| g_k \| g \| h \| y_1 \| \ldots \| y_n \| \tilde{y}_{11} \| \ldots \| \tilde{y}_{1\hat{e}_1} \| \tilde{y}_{21} \| \ldots \| \tilde{y}_{w\hat{e}_w} \\
S_2 &= \mathcal{J}_1 \| \ldots \| \mathcal{J}_n \| \mathtt{in} \| g^{-b_1} \textstyle\prod_{j=1}^{w} \tilde{y}_{je_{1j}}^{a_{1j}} \| \ldots \| g^{-b_u} \textstyle\prod_{j=1}^{w} \tilde{y}_{je_{uj}}^{a_{uj}} \| \Gamma \\
V_1 &= y_1^{c_1} \prod_{j\in\mathcal{J}_1} g_j^{s_{1j}} \| \ldots \| y_n^{c_n} \prod_{j\in\mathcal{J}_n} g_j^{s_{nj}} \\
V_2 &= \tilde{y}_{11}^{c_{\mathtt{pr}_1(\mathtt{in}(1))}} h^{\tilde{s}_{11}} g^{s_{\mathtt{in}(1)}} \| \ldots \| \tilde{y}_{1\hat{e}_1}^{c_{\mathtt{pr}_1(\mathtt{in}(1))}} h^{\tilde{s}_{1\hat{e}_1}} \tilde{y}_{1(\hat{e}_1-1)}^{s_{\mathtt{in}(1)}} \| \\
&\qquad \tilde{y}_{21}^{c_{\mathtt{pr}_1(\mathtt{in}(2))}} h^{\tilde{s}_{21}} g^{s_{\mathtt{in}(2)}} \| \ldots \| \tilde{y}_{w\hat{e}_w}^{c_{\mathtt{pr}_1(\mathtt{in}(n))}} h^{\tilde{s}_{w\hat{e}_w}} \tilde{y}_{w(\hat{e}_w-1)}^{s_{\mathtt{in}(w)}} \\
V_3 &= \left(g^{-b_1} \prod_{j=1}^{w} \tilde{y}_{je_{1j}}^{a_{1j}}\right)^{c_{n+1}} h^{\tilde{r}_1} \| \ldots \| \left(g^{-b_u} \prod_{j=1}^{w} \tilde{y}_{je_{uj}}^{a_{uj}}\right)^{c_{n+u}} h^{\tilde{r}_u}
\end{aligned}$$

is a signature of the message $m \in \{0,1\}^*$ *based on a proof*

- *of knowledge of the representations of all* $\{y_i | i \in \mathcal{S}\}$ *with respect to the bases* $\{g_j | j \in \mathcal{J}_i\}$, *and*
- *that, additionally, the secret keys* $(\alpha_{ij})_{i\in\mathcal{S}, i\le n; j\in\mathcal{J}_i}$ *satisfy*

$$b_i = \sum_{j=1}^{w} a_{ij} \alpha_{\mathtt{in}(j)}^{e_{ij}} \pmod{q} \text{ for all } i \,|\, (i+n) \in \mathcal{S}.$$

for at least one $\mathcal{S} \in \Gamma$. Such a signature is denoted by

$$SPK_9\Big\{(\alpha_{ij})_{i=1,\dots,n;j\in\mathcal{J}_i} : \bigvee_{\mathcal{S}\in\Gamma}\Big(\big(\bigwedge_{i\in\mathcal{S},i\leq n} y_i = \prod_{j\in\mathcal{J}_i} g_j^{\alpha_{ij}}\big)\wedge$$
$$\big(\bigwedge_{(i+n)\in\mathcal{S}} b_i = \sum_{j=1}^{w} a_{ij}\alpha_{\mathtt{in}(j)}^{e_{ij}} \pmod{q}\big)\Big)\Big\}(m).$$

In this definition $\mathtt{pr}_1(\mathtt{in}(\cdot))$ denotes the first projection of the index-tuple that is output by $\mathtt{in}(\cdot)$). For instance, if $\mathtt{in}(8)$ is (3,7), then $\mathtt{pr}_1(\mathtt{in}(8))$ is 3. In particular, $\mathtt{pr}_1(\mathtt{in}(1)) = 1$ and $\mathtt{pr}_1(\mathtt{in}(w)) = n$ holds.

The security properties of the SPK_9 follow from those of the SPK_5 and the SPK_8. The way of computing an SPK_9 should be clear from the way an SPK_5 and an SPK_8 are computed.

3.5.3 Related Work

In this subsection we compare our method for proving the validity of modular relations with that of Brands [Bra97].

The idea behind the scheme of Brands is as follows (translated into our setting, i.e., using the scheme of Brands in the SPK_8 and SPK_9). The known secret keys are committed with a single additional public key, i.e., if $x_1, \dots, x_k$ are the secret keys, the additional public key is $\tilde{y} := \prod_{i=1}^{k} g_i^{x_i}$, where $\log_{g_j} g_i$, $(i \neq j)$, must not be known. Then, modular relations hold if the prover can show that she knows a representation of $\tilde{y}$ with respect to a set of bases that is constructed from $g_1, \dots, g_k$ according to the considered relations.

A comparison between the efficiency of Brands' scheme with ours depends strongly on the proved statements. In the following we consider only the length of signatures (or the number of values communicated in the interactive case). If the statement to be proved contains no $\vee$-connectives, the scheme of Brands is more efficient. When the statement also contains $\vee$-connectives, our approach is more efficient.

Finally, let us remark that the scheme of Brands makes it possible to use the logical NOT in statements about equations. However, this could easily be added to our schemes as follows. First note that if a modular relation does not hold, then the g-part of the representation of the $\hat{y}_i$

that corresponds to the considered relation does not vanish. Thus the prover must show that this is the case by providing the signature

$$SPK_2\{(\alpha,\beta) : g = h^{\alpha}\hat{y}_i^{\beta}\}.$$

Of course, these signatures must be appropriately merged into the SPK_8 and SPK_9.

Chapter 4

Efficient and Generalized Group Signature Schemes

A group signature scheme allows a member of a group to sign messages anonymously on behalf of the group. In the case of later dispute a designated group manager can revoke the anonymity and identify the originator of a signature. In this chapter we describe a new efficient group signature scheme. Furthermore, we present a model and the first realization of a generalized group signature scheme. This type of group signature scheme allows the definition of coalitions of group members that are able to sign on the group's behalf. These results have appeared in [Cam97]. The schemes presented in this chapter, as well as all previously proposed schemes, have the property that the size of the group's public key and/or the length of signatures are linear in the number of group members. In the next chapter we present group signature schemes where all parameters are independent of the number of group members.

4.1 Introduction

In [CvH91] Chaum and van Heyst proposed a new type of signature scheme for a group of entities, called *group signatures*. Such a scheme allows a group member to sign a message on the group's behalf such that everybody can verify the signature but no one can find out which group member provided it. However, there is a trusted third party, called the group manager, who can reveal the identity of the originator of a signature in the case of later dispute. This act is referred to as "opening" a signature or also as revocation of a signer's anonymity. The group manager can either be a single entity or a number of coalitions of several entities (e.g., group members). This concept can be generalized to allow designated subsets of all group members to jointly sign a message on behalf of the group.

Group signatures could for instance be used by a company for authenticating price lists, press releases, or digital contracts. The customers need to know only a single company public key to verify signatures. The company can hide any internal organizational structures and responsibilities, but can still find out which employee (i.e., group member) has signed a particular document.

4.1.1 Related Work

Apart from group signature schemes, there exist several other group-oriented concepts for signature schemes, most notably multi-signatures [Boy89, CH89, OO93] and proxy signatures [MUO96]. Multi-signatures can be seen as generalized group signatures without the possibility of identifying the signers later, while proxy signatures are group signatures that do not provide the signer with anonymity.

Solutions for group signature schemes were first presented in [CvH91] and later in [CP95, Che94]. We discuss these schemes briefly. In [CvH91] four different schemes were proposed. Three of them require the group manager to contact each group member in order to find out who signed a message. These schemes provide computational anonymity, whereas the fourth scheme provides information-theoretic anonymity. For two of the schemes it is not possible to add a new member after the scheme has been set up (including the scheme giving information-theoretic anonymity). In none of the proposed schemes does it seem possible to distribute the functionality of the group manager efficiently.

Later, Chen and Pedersen proposed two new schemes in [CP95, Che94] providing information theoretic anonymity and computational anonymity, respectively. These schemes allow the addition of new members after the setup of the system and the distribution of the functionality of the group manager. They are based on proofs of knowledge of discrete logarithms of one out of several public keys, each belonging to a group member. These proofs have the special property that, when one knows all secret keys, one can tell which one was used in the proof. Two such proofs are used in parallel, each group member has a public/secret key pair for each the two proofs. One of the two secret keys of each group member is made known to the group manager. Hence she[1] can tell which member signed, but not sign on behalf of members, since she lacks the knowledge of the members' second secret key. However, this solution has the drawback that the group manager can falsely accuse a group member of having signed a message: to this end she would compute one of the proofs of knowledge using the known secret key of the member she wants to accuse. This risk can be weakened, but not prevented, by sharing the functionality of the group manager.

Recently and independently, Petersen [Pet97] proposed an (ordinary) group signature scheme that is similar to that presented in Section 4.3.1. Furthermore, he also originated a threshold group signature scheme which is a generalized group signature scheme for a threshold authority structure.

4.1.2 The Schemes Presented in This Chapter

In this chapter we describe an efficient group signature scheme where the manager cannot falsely accuse group members (even if she is also a group member). Furthermore, we also present the first generalized group signature scheme. In both schemes, the functionality of opening signatures can be shared among several entities such that the identity of a signer can still be revealed efficiently by them. Both schemes allow the addition (or removal) of group members after the initial setup. They provide computational anonymity which we believe is satisfactory, because the security of the signature scheme itself is also computational (as is the case for all signature schemes). Moreover, both schemes are secure in the random oracle model [BR93, BR96].

[1] It is at times convenient to call the group manager Maude. Thus 'she' refers to the group manager and 'he' refers to some group member

4.2 Our Model of Group Signature Schemes

In this section we define the concept of (generalized) group signature schemes. Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be a set of *group members* and M be a designated entity, called *group manager*. Let Γ be a subset of $2^{\{1,\ldots,n\}}$. A set of group members $\{P_i | i \in \mathcal{S}, \mathcal{S} \in \Gamma\}$ is called an *authorized coalition* and Γ is called *authority structure*. The structure must be *monotone*, i.e., for two sets $\mathcal{S}$ and $\mathcal{S}' \in 2^{\{1,\ldots,n\}}$, if $\mathcal{S} \in \Gamma$ and $\mathcal{S}' \supseteq \mathcal{S}$, then also $\mathcal{S}' \in \Gamma$. The special case where we have the non-monotone authority structure $\Gamma = \{\{1\}, \{2\}, \ldots, \{n\}\}$ is called a *simple* group signature scheme.

A (generalized) group signature scheme for $\mathcal{P}$ and M with respect to Γ consists of four procedures:

setup: A probabilistic interactive protocol between the group manager M and the members of $\mathcal{P}$. On input Γ this protocol outputs the group's public key $\mathcal{Y}$, a secret key x_i to each group member $P_i \in \mathcal{P}$, and an *opening secret key* ω to the group manager M.

sign: A probabilistic interactive protocol between members $P_i | i \in \mathcal{S}$ for some $\mathcal{S} \in \Gamma$. On input of a message m, the group's public key $\mathcal{Y}$, the access structure Γ , the coalition $\mathcal{S}$, and the secret keys x_i of the group members $P_i | i \in \mathcal{S}$, this multi-party protocol outputs a signature s of m.

verify: On input a message m, the group's public key $\mathcal{Y}$, the structure Γ, and a signature s, this algorithm outputs yes if and only if the signature is correct.

open: An algorithm that takes as input a message m, the group public key $\mathcal{Y}$, the structure Γ, a signature s and the revocation secret key ω. If s is a valid group signature of m with respect to $\mathcal{Y}$, the algorithm outputs $\mathcal{S} \in \Gamma$ and a proof that the group members $P_i | i \in \mathcal{S}$ indeed signed m.

The requirement that open also outputs a proof is often omitted but is essential when minimizing the trust to be put into the group manager.

The group publishes its public key $\mathcal{Y}$, the authority structure Γ, and some system parameters. A group signature scheme must satisfy the following properties:

1. Only authorized coalitions $\{P_i | i \in \mathcal{S}, \mathcal{S} \in \Gamma\}$ of group members can sign (unforgeability). The correctness of a signature can be publicly verified using $\mathcal{Y}$ and Γ.
2. Given a signature, it is neither possible to find out which coalition of group members signed a message (anonymity) nor whether two different signatures were signed by the same coalition (unlinkability).
3. Group members can neither prevent the opening of a signature nor sign on behalf of other group members/coalitions (opening of signatures). This must be infeasible even if the group manager is involved (security against framing).
4. The group manager must only be involved in the procedures `setup` and `open`.

The following natural properties should also be satisfied by a group signature scheme. Property 7 was stated as an open problem in [CvH91] and achieved first in [CP95, Che94].

5. The group manager is only trusted not to open signatures when this is not required, but is not trusted with regard to anything else.
6. It should be possible to assign the different roles of the group manager, namely managing group-membership of entities and opening signatures[2], to different parties. We call these roles membership manager and revocation manager.
7. To increase the security against a cheating group manager, it should be possible to share the roles among a set of entities (e.g., the members of the group).

When considering the efficiency of a scheme, the following parameters are of particular relevance:

- the efficiency of the procedures `sign` and `verify`,
- the length of signatures,

[2] We also refer to the act of opening signature as the revocation of a group member's anonymity.

- the size of the group's public key $\mathcal{Y}$, and
- the efficiency of the procedures `setup` and `open`.

4.3 Constructions of the Schemes

The observation leading to our constructions is that the signature systems

$$SPK_4\Big\{(\alpha_1,\dots,\alpha_n) : \bigvee_{i=1}^{n} y_i = g^{\alpha_i}\Big\}(m)$$

and

$$SPK_5\Big\{(\alpha_1,\dots,\alpha_n) : \bigvee_{S\in\Gamma} (\bigwedge_{i\in S} y_i = g^{\alpha_i})\Big\}(m)$$

defined in Chapter 3 already satisfy most properties of a simple and a generalized group signature scheme, respectively, if the group's public key is $\mathcal{Y} = (y_1,\dots,y_n)$ and each group member P_i knows the discrete logarithm of y_i. The only missing properties are related to the group manager's capability of "opening" a signature. In the following we present efficient solutions to achieve these missing properties by using a variation of the ElGamal encryption scheme and the techniques discussed in the previous chapter. Furthermore, these solutions allow a simple way to distribute the functionality of the group manager. The latter is described in Section 4.4.

4.3.1 An Efficient Simple Group Signature Scheme

The algebraic setting is as follows. Let G be a finite cyclic group of prime order q and let g be a generator of G such that computing discrete logarithms to the base g is not feasible. Furthermore, let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\ell$ ($\ell \approx 160$) denote a collision-resistant hash function.

The group manager M randomly chooses her secret key ω from $\mathbb{Z}_q$ and computes her public key $z{:=}g^\omega$ and the commitment $SPK_1\{(\alpha) : z = g^\alpha\}(M)$ to it. The group manager's commitment is sent to all group members. Each group member P_i

- chooses his secret key x_i randomly from $\mathbb{Z}_q$,
- computes $y_i := g^{x_i}$,
- commits to y_i by computing $SPK_1\{(\alpha) : y_i = g^{\alpha}\}(P_i)$, and finally
- sends y_i and his commitment to the group manager.

The group publishes $\mathcal{Y} := (y_1, \ldots, y_n)$ together with the group manager's public key z and the system parameters G, g, q, and $\mathcal{H}$.

The missing properties are obtained in that, to sign a message m, a group member encrypts one of the public keys of $\mathcal{Y} = \{y_1, \ldots, y_n\}$ for the group manager and provides a signature (SPK) of the message m proving that

- he encrypted one of the y_i's with respect to the group manager's public key and that
- he knows the discrete logarithm of the encrypted public key.

It follows from these two proofs that the group member must have encrypted his public key and thus the group manager can later identify the group member as the signer by simple decryption.

Such a group signature scheme could in principle be realized with an ElGamal encryption and a single signature of the type SPK_9. However, using the technique for proving equality of secret keys as described in Section 3.4 one can construct a more efficient scheme. Therefore, we define a new building block.

Definition 4.1. *A $2n$-tuple $(c_1, \ldots, c_n, s_1, \ldots, s_n) \in (\{0,1\}^{\ell})^n \times (\mathbb{Z}_q)^n$ satisfying the equation*

$$\bigoplus_{i=1}^{n} c_i = \mathcal{H}(S \| V \| m)$$

with

$$S = g \| h \| y_1 \| z_1 \| \cdots \| y_n \| z_n$$

$$V = y_1^{c_1} g^{s_1} \| z_1^{c_1} h^{s_1} \| \cdots \| y_n^{c_n} g^{s_n} \| z_n^{c_n} h^{s_n}$$

is a signature of the message $m \in \{0,1\}^*$ based on a proof of knowledge and of equality of the discrete logarithm of y_i with respect to the base g and of the *discrete logarithm z_i with respect to the base h for (at least) one i, $1 \leq i \leq n$. Such a signature is denoted*

$$SPK_{10}\Big\{(\alpha_1,\ldots,\alpha_n) : \bigvee_{i=1}^{n}(y_i = g^{\alpha_i} \wedge z_i = h^{\alpha_i})\Big\}(m)$$

The term $\bigoplus_{i=1}^{n} c_i$ found in this definition denotes the bitwise XOR of all c_i's. Similar to a signature of the type $SPK_6\{\alpha : y_i = g^{\alpha} \wedge z_i = h^{\alpha}\}$, a signature of the type SPK_{10} can be seen as two parallel signatures $SPK_4\{(\alpha_1,\ldots,\alpha_n) : \bigvee_{i=1}^{n} y_i = g^{\alpha_i}\}$ and $SPK_4\{(\alpha_1,\ldots,\alpha_n) : \bigvee_{i=1}^{n} z_i = h^{\alpha_i}\}$ where the exponents for the commitments, the challenges c_i, and the responses s_i, respectively, are equal for each i (cf. Sect. 3.4). From this it should be clear how a signature of the type SPK_{10} is computed. We note that two signatures are perfectly indistinguishable (see Section 2.9) with respect to which i the discrete logarithms of y_i and z_i are known for and are equal, and that the corresponding interactive protocol is honest-verifier zero-knowledge. Thus this signature scheme is secure in the random oracle model.

The notion SPK_{10} allows us now to formally present our efficient simple group signature scheme. To generate a signature of a message m, the group member P_j executes the following steps:

1. choose a randomly in $\mathbb{Z}_q$
2. encrypt y_j by computing $A := z^a$ and $B := y_j g^a$
3. compute the signature $(c_1,\ldots,c_n,s_1,\ldots,s_n) :=$
$$V_1 := SPK_{10}\Big\{(\alpha_1,\ldots,\alpha_n) : \bigvee_{i=1}^{n}\big(A = z^{\alpha_i} \;\wedge\; (B/y_i) = g^{\alpha_i}\big)\Big\}(m)$$
4. compute the signature $(\tilde{c},\tilde{s}) := V_2 := SPK_1\{(\alpha) : B = g^{\alpha}\}(V_1)$

The computed group signature is (A, B, V_1, V_2) and can be verified by checking the correctness of the signatures V_1 and V_2.

Remark 4.1. Signatures of this scheme could even be made shorter by merging the two signatures V_1 and V_2 into a single one. This is achieved, for instance, by defining $\tilde{c}$ to be the XOR of all the c_i's.

The signature V_1 assures that (A, B) is the encryption of one of the public keys in the list $\mathcal{Y}$ while the signature V_2 guarantees that the signer actually knows the discrete logarithm of the public key encrypted in (A, B). The signer thus indirectly proves his knowledge of the discrete logarithm of a public key contained in $\mathcal{Y}$ and therefore that he is a member of the group $\mathcal{P}$. It can easily be seen that only group members can sign messages.

To open a valid signature the group manager decrypts (A, B) and immediately obtains the public key of the signer. Assume that the group member P_j was the signer. By providing

$$SPK_6\{(\alpha) : z = g^{\alpha} \wedge A = (B/y_j)^{\alpha}\}(P_j)$$

the group manager can prove that she opened the signature correctly and that indeed P_j has issued this signature.

4.3.2 A Generalized Group Signature Scheme

The system parameters for this scheme are the same as for the simple group signature scheme. In addition to the group members' public keys, the group manager's public key and to the system parameters, the authority structure Γ is also published.

The ideas behind the generalized scheme are more or less the same as those for the simple scheme. To sign a message m all members of an authorized coalition prove that each of them encrypted an element of $\mathcal{Y} = \{y_1, \ldots y_n\}$ and that they know the discrete logarithms of the encrypted values. Furthermore, they must also prove that the encrypted elements are all different. The problem with this approach is that the number of encryptions equals the size of the coalition, which should be kept secret. Therefore, the coalition must also encrypt some dummy values in order to provide n encryptions.

A generalized group signature scheme could in principle be realized with ElGamal-encryptions and a single signature of the type SPK_9. However, for the same reasons as for the simple group signature scheme, we define a new building block to get a more efficient scheme.

Definition 4.2. *A* $2n$*-tuple* $(c_1, \ldots, c_n, s_1, \ldots, s_n) \in (\{0,1\}^{\ell})^n \times (\mathbb{Z}_q)^n$ *satisfying the equation*

$$\forall \mathcal{S}' \in \Gamma^* : \quad \mathtt{rec}_{\Gamma^*}(\mathcal{S}', (c_i)_{i \in \mathcal{S}'}) \; = \; \mathcal{H}(S \| V \| m)$$

with

$$S = g\|h\|y_1\|z_1\|\ldots\|y_n\|z_n$$
$$V = y_1^{c_1}g^{s_1}\|z_1^{c_1}h^{s_1}\|\ldots\|y_n^{c_1}g^{s_n}\|z_n^{c_1}h^{s_n}$$

is a signature of the message $m \in \{0,1\}^*$ based on a proof of knowledge and of equality of the discrete logarithm of y_i with respect to the base g and of the discrete logarithm z_i with respect to the base h for (at least) all i for which $P_i \in \mathcal{S}$ for some $\mathcal{S} \in \Gamma$. Such a signature is denoted

$$SPK_{11}\Big\{(\alpha_1,\ldots,\alpha_n) : \bigvee_{\mathcal{S}\in\Gamma}\big(\bigwedge_{i\in\mathcal{S}}(y_i = g^{\alpha_i} \wedge z_i = h^{\alpha_i})\big)\Big\}(m).$$

Recall that Γ^* denotes the dual access structure of Γ and that $\mathtt{rec}_{\Gamma^*}$ is an algorithm that reconstructs the secret from all shares of a qualified set $\mathcal{S} \in \Gamma^*$, (cf. Section 2.11). If the size of the shares and the size of the c_i's do not match, an appropriate matching must be introduced. In the sequel we assume that the sizes match. Again, we note that two signatures are perfectly indistinguishable (see Section 2.9) with respect to which y_i's the discrete logarithms are known for and that the corresponding interactive protocol is honest-verifier zero-knowledge and therefore this signature scheme is secure in the random oracle model.

The computation of such a signature is similar to that of a signature of the type SPK_{10} described in the previous subsection. An example is given in the next section.

We are now ready for a formal description of the generalized group signature scheme. To sign a message m on the group's behalf, the group members forming an authorized set $\{P_i | i \in \mathcal{S}, \mathcal{S} \in \Gamma\}$ execute the following steps together:

1. - choose $a_1,\ldots,a_n$, and b_i for all i with $i \notin \mathcal{S}$ randomly in $\mathbb{Z}_q$
   - for all $j \in \mathcal{S}$ encrypt y_j: $A_j := z^{a_j}$, $B_j := y_j g^{a_j}$
   - for all $i \notin \mathcal{S}$ encrypt g^{b_i}: $A_i := z^{a_i}$, $B_i := g^{b_i} g^{a_i}$
2. compute the signature $V_1 := (c_1,\ldots,c_n,s_1,\ldots,s_n) :=$

$$SPK_{11}\Big\{(\alpha_1,\ldots,\alpha_n) : \bigvee_{\mathcal{S}\in\Gamma}\big(\bigwedge_{i\in\mathcal{S}}((B_i/y_i) = g^{\alpha_i} \ \wedge \ A_i = z^{\alpha_i})\big)\Big\}(m)$$

3. compute the signature $(\tilde{c}, \tilde{s}_1, \ldots, \tilde{s}_n) :=$

$$V_2 := SPK_3\Big\{(\alpha_1, \ldots, \alpha_n) : \bigwedge_{i=1}^{n} B_i = g^{\alpha_i}\Big\}(V_1)$$

Member P_j must calculate parts of the signature V_1 and V_2 alone in order to hide his secret key from the other members. All other computations should be performed by all group members on their own in order to assure themselves of the correctness of the outcome. The random choices in these common computations must be agreed upon by the group members in advance, for instance by choosing a random string each, committing to the string by hashing it, exchanging these commitments, then exchanging the random strings, and finally taking the XOR of all these random strings. The resulting group signature is $(A_1, B_1, \ldots, A_n, B_n, V_1, V_2)$ and can be verified by checking the correctness of the SPK's V_1 and V_2.

The signature V_1 assures that the list $((A_1, B_1), \ldots, (A_n, B_n))$ contains the encryptions of the y_j's $\in \mathcal{Y}$ such that the corresponding P_j's form an authorized coalition. The signature V_2 assures that the authorized coalition was really involved, i.e., that the discrete logarithms of the encrypted y_j's are known. The inclusion of V_1 into V_2 as message prevents the reuse of the signature V_2 in another run of the scheme.

Remark 4.1 also applies for this group signature scheme.

Again, it is easy to see that the group manager can find out which coalition provided the signature by checking the validity of the signature and decrypting all pairs (A_j, B_j). Note that a coalition cannot encrypt a public key of a member not participating in the signing because in this case it could not provide the corresponding part in the signature V_2 and therefore the group signature would not be valid. By providing

$$SPK_6\{(\alpha) : z = g^{\alpha} \wedge A_j = (B_j/y_j)^{\alpha}\}(P_j)$$

for all P_j having participated in the signing, the group manager can assure that she opened the signature correctly.

4.3.3 An Example: A Threshold Group Signature Scheme

In this section we give an example for a generalized group signature scheme with a threshold authority structure. Let k be the minimum number of members that must cooperate in order to sign and let $f(x) := \sum_{i=0}^{k-1} u_i x^i \pmod{p}$ be the polynomial of a secret sharing scheme with threshold k over $\mathbb{Z}_p^*$ where $p \approx 2^\ell < q$ is a prime (cf. Section 2.11). Since the shares are elements of $\mathbb{Z}_p^*$, we define the c_i's to be elements of $\mathbb{Z}_p^*$, too.

To generate a signature of a message m, the group members forming an authorized coalition $\{P_i | i \in \mathcal{S}, \mathcal{S} \in \Gamma\}$, i.e., at least k group members, execute the steps below. In both steps it is indicated when the calculations must be performed by a specific member on his own. All other computations should be performed by all coalition members using the agreed-on random string.

1. - choose $a_1, \ldots, a_n$, and b_i for all i with $i \notin \mathcal{S}$ randomly in $\mathbb{Z}_q$
   - for all $j \in \mathcal{S}$, member P_j encrypts y_j: $A_j := z^{a_j}$, $B_j := y_j g^{a_j}$
   - for all $i \notin \mathcal{S}$ encrypt g^{b_i}: $A_i := z^{a_i}$, $B_i := g^{b_i} g^{a_i}$

2. compute $V_1 := (u_0, \ldots, u_{k-1}, s_1, \ldots, s_n)$

   $$SPK_{11}\Big\{(\alpha_1, \ldots, \alpha_n) : \bigvee_{\mathcal{S}\in\Gamma} \big(\bigwedge_{i\in\mathcal{S}}((B_i/y_i) = g^{\alpha_i} \wedge A_i = z^{\alpha_i})\big)\Big\}(m):$$

   - for all $j \in \mathcal{S}$, member P_j chooses r_j randomly in $\mathbb{Z}_q$ and calculates $t_{z,j} := z^{r_j}$ and $t_{g,j} := g^{r_j}$
   - for all $i \notin \mathcal{S}$ choose $r_i \in_R \mathbb{Z}_q$, $c_i \in_R \mathbb{Z}_p$, and compute $t_{z,i} := z^{r_i} A_i^{c_i}$ and $t_{g,i} := g^{r_i} (\frac{B_i}{y_i})^{c_i}$
   - $c := \mathcal{H}\left(z \| g \| A_1 \| \frac{B_1}{y_1} \| \ldots \| A_n \| \frac{B_n}{y_n} \| t_{z,1} \| t_{g,1} \| \ldots \| t_{z,n} \| t_{g,n} \| m\right)$
   - choose $u_0, \ldots, u_{k-1}$ such that $f(i) \equiv c_i \pmod{p}$ for all $i | i \notin \mathcal{S}$ and $f(0) \equiv c \pmod{p}$
   - for all $j \in \mathcal{S}$, member P_j computes $c_j := f(j) \pmod{p}$ and $s_j := r_j - c_j a_j \pmod{q}$
   - for all $i \notin \mathcal{S}$ set $s_i := r_i$

3. calculate $(\tilde{c},\tilde{s}_1,\dots,\tilde{s}_n) :=$

   $$V_2 := SPK_3\left\{(\alpha_1,\dots,\alpha_n) : \bigwedge_{i=1}^{n} B_i = g^{\alpha_i}\right\}(V_1):$$

   - for all $j \in \mathcal{S}$, member P_j chooses $\tilde{r}_j$ randomly in $\mathbb{Z}_q$ and computes $\tilde{t}_j := g^{\tilde{r}_i}$
   - for all $i \notin \mathcal{S}$, choose $\tilde{r}_i$ randomly in $\mathbb{Z}_q$ and compute $\tilde{t}_i := g^{\tilde{r}_i}$
   - $\tilde{c} := \mathcal{H}(g\|B_1\|\dots\|B_n\|\tilde{t}_1\|\dots\|\tilde{t}_n\|m\|u_0\|\dots\|u_{k+1}\|s_1\|\dots\|s_n)$
   - for all $j \in \mathcal{S}$, member P_j computes $\tilde{s}_j := \tilde{r}_j - \tilde{c}(x_j + a_j) \pmod{q}$
   - for all $i \notin \mathcal{S}$, compute $\tilde{s}_i := \tilde{r}_i - \tilde{c}(b_i + a_i) \pmod{q}$

The group signature of m is the tuple $(A_1, B_1, \dots, A_n, B_n, V_1, V_2)$. Instead of all c_i's, the values $u_0, \dots, u_{k-1}$ are part of V_1. This makes the signature shorter but equally secure since c and all c_i's are uniquely determined by $u_0, \dots, u_{k-1}$ through $f(\cdot)$.

The group signature can be verified by checking the following equations:

$$u_0 = \mathcal{H}\left(z\middle\|g\middle\|A_1\middle\|\frac{B_1}{y_1}\middle\|\dots\middle\|A_n\middle\|\frac{B_n}{y_n}\middle\|z^{s_1}A_1^{c_1}\middle\|g^{s_1}\left(\frac{B_1}{y_1}\right)^{c_1}\middle\|\dots\right.$$
$$\left.\dots\middle\|z^{s_n}A_n^{c_n}\middle\|g^{s_n}\left(\frac{B_n}{y_n}\right)^{c_n}\middle\|m\right)$$

and

$$\tilde{c} = \mathcal{H}(g\|B_1\|\dots\|B_n\|g^{\tilde{s}_1}B_1^{\tilde{c}}\|\dots\|g^{\tilde{s}_n}B_n^{\tilde{c}}\|m\|u_0\|\dots\|u_{k+1}\|s_1\|\dots\|s_n)$$

where

$$c_i \equiv f(i) \equiv u_0 + u_1 i + \dots + u_{k-1} i^{k-1} \pmod{p} \text{ for } 1 \leq i \leq n.$$

4.3.4 Security Properties

Let us briefly discuss the security properties of the generalized group signature scheme (which hold also for the simple scheme).

Signatures are unlinkable and anonymous: The unlinkability of two or more group signatures originated by the same members follows from the fact that signatures of the type SPK_{11} and SPK_{10} are unlinkable and from the fact that the y_i's are probabilistically encrypted. This also guarantees anonymity.

Non-authorized coalitions cannot sign: Because the signatures schemes of the type SPK_1, SPK_3, SPK_{10}, and SPK_{11} are secure in the random oracle model, so are the two group signature schemes. In other words, if a non-authorized coalition was able to sign, then it could also compute the discrete logarithms of the public keys of some entities that form an authorized coalition, which is assumed to be infeasible.

An authorized coalition cannot sign on another coalition's behalf: There are two cases. First, a coalition might try to sign on behalf of an other coalition that includes members that are not present. Again, we consider the random oracle model. If the coalition succeeded, they could also compute the discrete logarithms of the public keys of the members that are not present. Since this is assumed to infeasible, this first attack is not possible. In the other case, where a coalition S contains a true subset C that is also authorized, some members P_j with $j \in S/C$ might try to influence the computation of the signature such that they do not appear as signers. This is prevented if the members of the coalition assure themselves that the mutually agreed random string is indeed used for making all random choices in all the computations.

The group manager cannot falsely accuse members: Because the group manager must provide a proof, that public key(s) of the member(s) she accuses to be the signer(s) are indeed encrypted in the signatures, she cannot falsely accuse members.

4.3.5 Efficiency Considerations

With regard to efficiency, all algorithms except open have efficiency linear in the number of group members. The size of the group's public key and the length of signatures are also linear in the number of group members. The algorithm open is independent of the group's size (however, finding the identity of a signer given his key requires searching in a database).

Comparing the second scheme of [CP95, Che94] and our simple group signature scheme, it turns out that our scheme is approximately four times more efficient in terms of computations by the signer and signatures are about a quarter of the length. Furthermore, the algorithm open of [CP95, Che94] has an efficiency that is linear in the group's size.

4.4 Extensions

In this section we show how the functionality of the group manager can be shared among several parties (e.g., among the group members) and we present a method for reducing the size of the group's public key.

4.4.1 Sharing the Functionalities of the Group Manager

First of all, note that it is trivial to divert the group manager's tasks to a membership manager, who is responsible for the membership list (i.e., publishing a new group's public key upon inclusion or exclusion of group members), and to a revocation manager, who has the task of opening signatures.

To obtain higher security against fraudulent opening of signatures, the capability of the revocation manager can be shared among several managers according to an access structure such that only predefined subsets of the managers are able to cooperatively open a signature. This can be achieved by sharing the group manager's secret key ω among several managers with some secret sharing scheme. However, the secret sharing scheme must enable an exponentiation of group elements with ω and ω^{-1} by the revocation managers in a distributed manner without leaking information about the shares. Such secret sharing schemes are discussed in Chapter 7.

Remark 4.2. Computations with ω^{-1} by the revocation managers can be avoided if the group signature schemes are modified to use normal ElGamal encryption, i.e., $A_i{:=}g^{a_i}$ and $B_i{:=}y_i z^{a_i}$ (if the logarithm of y_i is known) and $A_i{:=}g^{a_i}$ and $B_i{:=}g^{b_i} z^{a_i}$ (otherwise). Then, for the generalized group signature scheme the SPK's in Step 2 and 3 must be adjusted (for the other scheme, the changes are similar): in Step 2 the SPK V_1

must be

$$SPK_{11}\Big\{(\alpha_1,\dots,\alpha_n): \bigvee_{S\in\Gamma} (\bigwedge_{i\in S}(A_i = g^{\alpha_i} \wedge (B_i/y_i) = z^{\alpha_i}))\Big\}(m)$$

and in Step 3 the SPK V_2 must be

$$SPK_3\Big\{(\alpha_1,\beta_1,\dots,\alpha_n,\beta_n): \bigwedge_{i=1}^{n} B_i = g^{\alpha_i} z^{\beta_i}\Big\}(V_1).$$

This change would make the signatures somewhat longer, but the public key of a group member who has signed can be computed simply by B_i/A_i^{ω}.

4.4.2 Reducing the Size of the Group's Public Key

The size of the group's public key can be reduced using a technique proposed by Blom for public key distribution [Blo85]. Let E be a publicly known generator matrix of an (n,k) MDS code[3] over $\mathbb{Z}_q$. The group's public key now is set to $\{y_1,\dots,y_k\}$ and the public key of member P_j is computed as

$$\tilde{y}_j := \prod_{i=1}^{k} y_i^{e_{ij}},$$

where e_{ij} denotes the element of E in row i and column j. These public keys are then used in Step 2 of the signature generating procedure. The secret keys of the individual group members are computed similarly. This method has the disadvantages that a trusted third party is needed to compute the group's public and secret keys, and that, if more than k group members collude, they can find out all secret keys and therefore sign on behalf of any authorized set. Hence there exists a trade-off between the size of the group's public key and the security.

[3] An (n,k) code over $\mathbb{Z}_q$ having generator matrix E encodes words $a \in (\mathbb{Z}_q)^k$ onto codewords $b \in (\mathbb{Z}_q)^n$, i.e., $b = aE$. A code is maximum-distance-separable (MDS), if the minimal distance between any two codes words equals $n - k + 1$ (see e.g., [Bla83]).

Chapter 5

Group Signature Schemes for Large Groups

In this chapter we present three group signature schemes where the size of the group's public key and the length of signatures are independent of the number of group members. In order to realize such schemes we employ novel techniques of independent interest, such as efficient proofs of knowledge of double discrete logarithms, of e-th roots of discrete logarithms, and of e-th roots of components of representations. A method for proving the knowledge of a signature is of particular interest. An extended abstract of this chapter has appeared in [CS97a].

5.1 Introduction

In all previously proposed group signature schemes, including those described in the previous chapter, the length of the group's public key is at least linear in the size of the group and therefore also the running time of the verification algorithm depends on the number of group members. In some schemes the length of the signature and the running time of the signing algorithm also depend on the group's size. This makes such schemes impractical for large groups. Furthermore, if new members are added to the group, it is necessary to modify at least the group's public key.

In this chapter we present the first group signature schemes which overcome these problems[1]. The lengths of the group's public key and of the signatures, as well as the computational effort for signing and verifying, are independent of the number of group members. Furthermore, new members can be added to the group without modifying the public key. A detailed comparison of the schemes in this chapter and those of the preceding chapter is found at the end of this chapter.

Recently, Kilian and Petrank proposed the concept of identity escrow [KP97], which is in essence the same as the concept of group signatures. Taken as a group signature scheme, their solution also achieves the goal of the size of the group's public key and the length signatures being independent of the number of group members. Their solution uses bit commitment schemes for proving the knowledge of a signature and hence is much less efficient than the schemes presented in this chapter.

5.2 The Basic Idea

The following is a simple construction of a group signature scheme which achieves the target of having the size of the group's public key *and* the length of the signatures independent of the number of group members. The group's public key consists basically of the group manager's public key of an ordinary digital signature scheme. Each group member generates a number of public and secret key pairs of a (possibly different) ordinary signature scheme. The group members' public keys, now called membership keys, are all signed/certified by the group manager. With this set up, a group member can sign a message by providing one of these membership keys, the corresponding certificate, and an ordinary signature of the message with respect to the provided membership key. Since the group manager knows which membership key belongs to which member, she can easily open signatures. Of course, a membership key can be used only once, since otherwise signatures would be linkable. Thus this solution is impractical. In principal, this could be overcome by only proving the knowledge of a membership key and a certificate, rather then presenting them. Such

[1] The only previously proposed schemes with fixed size public keys [PLW95, KPW96] were broken [Cam, Mic96].

a proof can be given using general (zero-knowledge) proof-techniques such as those of Brassard et al. [BCC88] or Boyar et al. [BP96]. Furthermore, since the membership key is no longer presented, a signer must also provide some information that allows the group manager to open a signature later on. More technically, such a scheme can be constructed as follows.

The group manager, we call her Maude, computes a key pair (y_M, x_M) of an ordinary digital signature scheme $(\mathtt{sig}, \mathtt{ver})$ and a key pair (y_R, x_R) of a public-key encryption scheme $(\mathtt{enc}, \mathtt{dec})$ (cf. Section 2.5) and publishes the two public keys y_R and y_M as the group's public key. The index M stands for her role as membership manager and the index R for her role as revocation manager. A group member, let us call him Arto, can join the group in the following way: he chooses a key pair (z, x) of some digital signature scheme[2] $(\mathtt{sig}', \mathtt{ver}')$ where x is his *secret key* and z his *membership key*. He commits himself to z, for instance by signing it, then sends z to the group manager. If Arto qualifies for group membership, Maude sends to him the *membership certificate* $v = \mathtt{sig}(z, x_M)$. Arto's stores the triple (x, z, v).

To sign a message m on behalf of the group, Arto encrypts z using Maude's encryption key, i.e., computes $d := \mathtt{enc}(z' \| r, y_R)$, where r is a sufficiently large random string. Arto further computes a non-interactive minimum-disclosure proof p that he knows values z', v', r', and s' $(= \mathtt{sig}'(m, x))$ satisfying

$$d = \mathtt{enc}(z' \| r', y_R)\,, \ \mathtt{ver}(z', v', y_M) = \mathtt{true}\,, \text{and } \mathtt{ver}'(m, s', z') = \mathtt{true}\,.$$

The resulting signature of the message m consists of the pair (d, p) and can be verified by checking the proof p.

To open such a signature, group manager Maude decrypts d and so obtains the membership key z of Arto. She can prove that Arto has indeed been the signer by providing z, Arto's commitment to it, and a non-interactive proof that d encrypts z.

It can easily be verified that the security properties required from a group signature scheme (cf. Chapter 4) hold:

1. Only group members knowing a membership certificate can construct a valid proof p.

[2] Although this scheme can be different from the one the group manager uses, it must be the same for all group members.

2. Because the proof p does not reveal "useful" information about x, z, s, or v, and because z is probabilistically encrypted due to the inclusion of the random string r, signatures are anonymous and unlinkable.

3. Group members cannot circumvent the opening of signatures because they prove that the value d contains their membership key.

4. The group manager is not involved in signing and verifying.

5. The group manager needs only be trusted with respect to opening of signatures. In particular, she cannot sign on behalf of group members since $(\mathtt{sig}', \mathtt{ver}')$ is a signature scheme and neither can she open signature falsely since she must prove that a membership key z is encrypted in d.

6. The roles of membership manager and revocation manager can easily be assigned to different entities by letting the membership manager choose the key pair (y_M, x_M) and the revocation manager the key pair (y_R, x_R).

7. The functionality of the the membership manager and the revocation manager can be shared among several entities if the chosen signature scheme for issuing certificates and the encryption scheme of the revocation manager allows it. We note that such schemes exist and refer to our explicit realizations.

The disadvantage of this solution is that the general techniques for proving statements in minimum-disclosure make the resulting signatures impractically long. The next sections introduces techniques for the construction of a more efficient scheme based on proofs of knowledge of double discrete logarithms and of roots of logarithms.

5.3 Building Blocks

Apart from the building blocks described in Chapter 3, our group signature schemes employ systems for proving the knowledge of double discrete logarithms and of e-th roots of discrete logarithms. Since the algebraic setting in this chapter is different from that in Chapter 3, we also describe how those signature schemes/proof systems can be adapted.

5.3.1 Double Discrete Logarithms and Roots of Logarithms

Let $G = \langle g \rangle$ be a cyclic group of order n and a be an element of $\mathbb{Z}_n^*$. A double discrete logarithm of $y \in G$ to the bases g and a is an integer x satisfying

$$g^{(a^x)} = y ,$$

if such an x exists.

An e-th root of the discrete logarithm of $y \in G$ to the base g is an integer x satisfying

$$g^{(x^e)} = y ,$$

if such an x exists.

For the rest of this chapter, we assume that a cyclic group $G = \langle g \rangle$ of order n is given, where n is a publicly known RSA-modulus. Furthermore, a is an element of $\mathbb{Z}_n^*$ with large multiplicative order. The parameters n, G, g, and a should be chosen such that computing discrete logarithms in G to the base g and in $\mathbb{Z}_n^*$ to the base a is infeasible. Since n is required to be an RSA-modulus, computing roots in $\mathbb{Z}_n^*$ is also infeasible without knowing the factorization of n. Finally, let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\ell$ ($\ell \approx 160$) denote a collision-resistant hash function.

5.3.2 Proofs of Knowledge of Discrete Logarithms and Representations

Our group signature schemes build on the the signature/proof systems described in Chapter 3. However, in the current chapter the algebraic setting is different from that in Chapter 3. The following two paragraphs describe how those building blocks must be adapted in order to remain secure in the random oracle model, i.e., in order that the corresponding interactive protocols remain honest-verifier zero-knowledge proofs of knowledge.

Let us first consider the case where g, or other generators of G, serve as a base. Since the order $n = pq$ of G, where p and q are two large primes, is not prime, an upper bound 2^ℓ on the challenge c is needed, i.e., c

must be smaller than both prime-factors of n. This is to prevent the difference of two random challenges from being congruent to 0 modulo one of the prime factors of n, since then the knowledge extractor would fail to find a witness. On the other hand c must not be too small, i.e., polynomial in the input size, since the success probability of the knowledge extractor would otherwise be too small. An example of an upper bound is $\ell = 0.4 \log n$ assuming that p and q are $\approx 0.5 \log n$. Finally, we note that all involved entities should verify that the order of G is indeed the product of two equally large primes and that the generators are indeed generators of G. Otherwise, for instance if g had only order p and p were small, then it would be possible to compute discrete logarithms to the base g for the entities knowing p.

The other case to consider is when a serves as a base (we assume that it is prime, however)[3]. If the prover does not know a's order[4], an upper-bound is needed on those secret keys that are used as exponents with a as base, e.g., $x \in \{0, \ldots, 2^{\lambda} - 1\}$ for some λ. Of course, λ must still be large, since otherwise it would be feasible to compute discrete logarithms using Pollard's lambda algorithm using at most $O(2^{\lambda/2})$ group operations. An example is $\lambda = |n|$. As a consequence, to achieve statistical (honest-verifier) zero-knowledgeness, the randomizers r used as exponents for the commitments must be chosen from a larger set than x. If the challenge c is chosen from $\{0, \ldots, 2^{\ell} - 1\}$, then the r's must be chosen from $\{0, \ldots, 2^{(\lambda+\ell)\epsilon} - 1\}$, where ϵ is a constant > 1.

5.3.3 Proofs of Knowledge of Double Discrete Logarithms

A new building block for our group signature schemes are signatures based on proofs of knowledge of the double discrete logarithm of an element of G. Unfortunately, such proofs are far less efficient than proofs of knowledge of ordinary discrete logarithms. As in Chapter 3, we present the building block as a signature scheme. It is based on techniques introduced in [Sta96a, Sta96b] for constructing particular verifiable secret sharing schemes.

[3] If the order is not prime, but for instance the product of two large primes, the challenge c also must be upper-bounded.

[4] Nevertheless he should be assured that a's order is large and not smooth. How this can be achieved is discussed in Section 5.4.1.

Definition 5.1. *Let $k \leq \ell$ be a security parameter. A $(k+1)$ tuple $(c, s_1, ..., s_k)$ $\in \{0,1\}^\ell \times \{-(2^\lambda - 1), \ldots, 2^{\epsilon\lambda} - 1\}^k$ satisfying*

$$c = \mathcal{H}(S \| V \| m) \ \ with \ \ S = g \| a \| y \ \ and \ \ V = t_1 \| \ldots \| t_k,$$

where

$$t_i = \begin{cases} g^{(a^{s_i})} & if \, c[i] = 0 \\ y^{(a^{s_i})} & otherwise, \end{cases}$$

is a signature of the message $m \in \{0,1\}^*$ based on a proof of knowledge of a double discrete logarithm of y to the bases g and a. It is denoted

$$SPK_{12}\{(\alpha): \ y = g^{(a^\alpha)} \}(m).$$

An $SPK_{12}\{(\alpha): \ y = g^{(a^\alpha)} \}(m)$ can be computed if the double discrete logarithm x of the group element y to the bases g and a is known, i.e., $y = g^{a^x}$. We assume that there is an upper bound λ on the length of x, i.e., $0 \leq x < 2^\lambda$. Let $\epsilon > 1$ be a constant[5]. One first computes the values

$$\bar{t}_i := g^{(a^{r_i})}$$

for $i = 1, \ldots, k$ with randomly chosen r_i's $\in \{0, \ldots, 2^{\epsilon\lambda} - 1\}$. Then, c is set to $\mathcal{H}(m \,\|\, y \,\|\, g \,\|\, a \,\|\, \bar{t}_1 \,\|\, \ldots \,\|\, \bar{t}_k)$, and finally,

$$s_i := \begin{cases} r_i & \text{if } c[i] = 0 \\ r_i - x & \text{otherwise} \end{cases}$$

is computed for $i = 1, \ldots, k$. It can easily be verified that the resulting tuple $(c, s_1, \ldots, s_k)$ satisfies the verification equations. Note that if the order of $a \in \mathbb{Z}_n^*$ is known, the computations of the s_i can be "reduced" modulo this order (as can the r_i's).

Theorem 5.1. *The interactive protocol corresponding to the SPK_{12} is* honest-verifier zero-knowledge *and a* proof of knowledge *of the double discrete logarithm of y with respect to the bases g and a for $\lambda = p(|g|)$ and any $\epsilon > 1$, where $p(\cdot)$ denotes any polynomial.*

[5] Using the upper-bound 2^λ and the parameter ϵ is in the prover's interest, since for these values the corresponding interactive proof is (honest-verifier) zero-knowledge (see Theorem 5.1).

Proof (sketch). Proof of knowledge: We only show how a double discrete logarithm of $y = g^{a^x}$ can be computed from two different views having the same commitments. Without loss of generality, we assume that the j-th bits of c and $\tilde{c}$ differ and that $c[j] = 0$. Then we ha'e

$$t_j = g^{(a^{s_j})} = y^{(a^{\tilde{s}_j})} = g^{a^x(a^{\tilde{s}_j})}$$

and thus $a^x \equiv a^{s_j - \tilde{s}_j} \pmod{n}$ holds. Hence, we can compute (in $\mathbb{Z}$)

$$x := s_j - \tilde{s}_j.$$

Honest-verifier zero-knowledgeness: The simulator randomly chooses c from $\{0, \ldots, 2^\ell - 1\}$ and all $r_i = s_i$ from $\{0, \ldots, 2^{\epsilon\lambda} - 1\}$. Using these values, the simulator computes

$$t_i := \begin{cases} g^{(a^{s_i})} & \text{if } c[i] = 0 \\ y^{(a^{s_i})} & \text{otherwise} \end{cases}$$

for $i = 1, \ldots, k$. To prove that these values are statistical indistinguishable from a view of a protocol run with the prover, it suffices to consider the probability distribution $P_{S_1,\ldots,S_k}(s_1, \ldots, s_k)$ of the s_i's of the prover and $P_{R_1,\ldots,R_k}(r_1, \ldots, r_k)$ according to which the simulator chooses the r_i's. The latter is $\prod_{i=1}^k P_{R_i}(r_i)$, where $P_{R_i}(r_i)$ it the uniform distribution over $\{0, \ldots, 2^{\lambda\epsilon} - 1\}$. If the prover chooses the r_i's uniformly at random from $\{0, \ldots, 2^{\lambda\epsilon} - 1\}$ and the secret key randomly from $\{0, \ldots, 2^\lambda - 1\}$ according to any distribution, we have

$$P_{S_i}(s) \begin{cases} = 0 & \text{for } s < -(2^\lambda - 1) \\ \leq 2^{-\epsilon\lambda} & \text{for } -(2^\lambda - 1) \leq s < 0 \\ = 2^{-\epsilon\lambda} & \text{for } 0 \leq s \leq 2^{\epsilon\lambda} - 2^\lambda \\ \leq 2^{-\epsilon\lambda} & \text{for } 2^{\epsilon\lambda} - 2^\lambda < s \leq (2^{\epsilon\lambda} - 1) \\ = 0 & \text{for } (2^{\epsilon\lambda} - 1) < s. \end{cases}$$

This holds for any distribution of $c[i]$ over $\{0, 1\}$. Similarly, the probability $P_{S_1,\ldots,S_k}(s_1, \ldots, s_k)$ is 0 if any s_i is smaller than $-(2^\lambda - 1)$ or larger than $(2^{\epsilon\lambda} - 1)$, equals $2^{-k\epsilon\lambda}$ if all s_i are in the range $\{0, \ldots, 2^{\lambda\epsilon} - 2^\lambda\}$, and

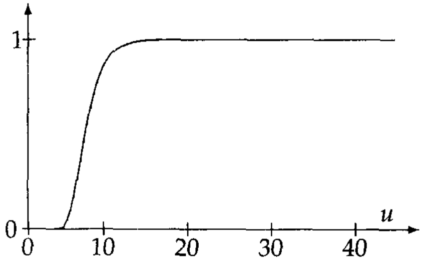

Figure 5.1: The function $(1-\frac{2}{2^u})^k$ for $k=80$

is smaller or equal than $2^{-k\epsilon\lambda}$ in the other cases. Thus we have

$$\begin{aligned}\sum_{\alpha\in(\mathbb{Z})^k} |P_{S_1,\dots,S_k}(\alpha) - P_{R_1,\dots,R_k}(\alpha)| &\le \frac{2(2^{k\lambda\epsilon} - (2^{\lambda\epsilon} - 2^{\lambda} + 1)^k)}{2^{k\lambda\epsilon}} \\ &= \frac{2\left(2^{k\lambda\epsilon} - 2^{k\lambda\epsilon}\left(1 + \frac{1-2^{\lambda}}{2^{\lambda\epsilon}}\right)^k\right)}{2^{k\lambda\epsilon}} \le 2\left(1 - \left(1 + k\frac{1-2^{\lambda}}{2^{\lambda\epsilon}}\right)\right) \\ &= 2k\frac{2^{\lambda}-1}{2^{\lambda\epsilon}} < \frac{2k}{(2^{\lambda})^{\epsilon-1}}.\end{aligned}$$

For λ as stated in the theorem, the last term can be expressed as one over a polynomial in the input length, and therefore the two distributions are statistical indistinguishable. □

To clarify what statistical (honest-verifier) zero-knowledgeness means to the prover and to judge how ϵ should be chosen in practice, let us consider the information an s_i gives about the secret key x. An s_i only gives information about x if $c[i]=1$ and if $s_i<0$ or $s_i>2^{\lambda\epsilon}-2^{\lambda}$. The smaller (respectively, the larger) it is, the more information it gives (e.g., if $s_i=1-2^{\lambda}$ then $x=2^{\lambda}-1$ is completely determined). The probability that a signature SPK_{12} (or the related proof system) gives no additional information about x is thus at least

$$\left(\frac{(2^{\lambda\epsilon}-2^{\lambda}-2^{\lambda}+1)}{2^{\lambda\epsilon}}\right)^k > \left(1-\frac{2}{2^{\lambda(\epsilon-1)}}\right)^k.$$

Figure 5.1 depicts the function $(1-\frac{2}{2^u})^k$ for $k=80$. One can see that already for $u\approx 15$, the probability that a signature SPK_{12} does not give information on the secret key is almost 1. In applications $u = \lambda(\epsilon - 1)$ would be at least 50 (e.g., $\epsilon = 4/3$ and $\lambda = 170$) and k around 80.

5.3.4 Proofs of Knowledge of Roots of Discrete Logarithms

Another building block for our group signature schemes are proofs of knowledge of an e-th root of a discrete logarithm. Of course, such proofs only make sense if the factorization of the order of the base of the discrete logarithms is not known. The proof system presented here is not very efficient but works for all exponents e. For small exponents, however, one can construct more efficient systems which are presented later on.

Definition 5.2. *A* $(k+1)$ *tuple* $(c, s_1, \ldots, s_k) \in \{0,1\}^\ell \times (\mathbb{Z}_n^*)^k$ *satisfying*

$$c = \mathcal{H}(S\|V\|m) \quad \textit{with } S = g\|e\|y \textit{ and } V = t_1\|\ldots\|t_k$$

where

$$t_i = \begin{cases} g^{(s_i^e)} & \textit{if } c[i] = 0 \\ y^{(s_i^e)} & \textit{otherwise} \end{cases}$$

is a signature of the message $m \in \{0,1\}^*$ *based on a proof of knowledge of an e-th root of the discrete logarithm of* y *to the base* g*, and is denoted*

$$SPK_{13}\{(\alpha) : \; y = g^{\alpha^e}\}(m).$$

Note that the integers $s_1, \ldots, s_k$ are elements of $\mathbb{Z}_n^*$ and therefore are not zero.

Such a signature can be computed if the e-th root x of the discrete logarithm of y to the base g is known, i.e., $y = g^{x^e}$. One first computes the values

$$\bar{t}_i := g^{(r_i^e)}$$

for $i = 1, \ldots, k$ with randomly chosen $r_i \in \mathbb{Z}_n^*$. Then, c is set to $\mathcal{H}(m \,\|\, y \,\|\, g \,\|\, e \,\|\, \bar{t}_1 \,\|\, \ldots \,\|\, \bar{t}_k)$, and finally,

$$s_i := \begin{cases} r_i & & \text{if } c[i] = 0, \\ r_i/x & \pmod{n} & \text{otherwise.} \end{cases}$$

for $i = 1, \ldots, k$. It can easily be seen that the resulting tuple $(c, s_1, \ldots, s_k)$ satisfies the verification equation.

Theorem 5.2. *The identification protocol corresponding to the SPK_{13} is* honest-verifier zero-knowledge *and a* proof of knowledge *of an e-th root of the discrete logarithm of y with respect to the base g.*

Proof (sketch). Proof of knowledge: We only show how an $x \in \mathbb{Z}_n^*$ such that $y = g^{x^e}$ can be computed from two different views having the same commitments. Without loss of generality we assume that j-th bits of c and $\tilde{c}$ differ and that $c[j] = 0$. Then we have

$$t_j = g^{s_j^e} = y^{\tilde{s}_j^e} = g^{x^e \tilde{s}_j^e}$$

and thus

$$x \equiv \frac{s_j}{\tilde{s}_j} \pmod{n}$$

since all s_j's and $\tilde{s}_j$'s are elements of $\mathbb{Z}_n^*$.

Honest-verifier zero-knowledge: The simulator can be constructed as usual.

□

For small exponents e, one can construct a more efficient system for proving the knowledge of an e-th root of a discrete logarithm. The idea is the same as for proving that polynomial relations hold among the secret keys as described in Section 3.5: the prover provides an additional element of G and shows that the discrete logarithm of this element is the e-th root of the discrete logarithm of the original element.

We assume that an element $h \in G$ is available, the discrete logarithm to the base g of which is unknown (for instance, h could be computed according to a suitable pseudo-random process with g as seed). The element h is used to randomize the additional elements the prover provides. This randomization assures that these additional elements do not leak information (cf. Section 3.5).

Definition 5.3. *An $(e-1)$-tuple $(\tilde{z}_1, \ldots, \tilde{z}_{e-1}) \in G^{e-1}$ and a signature*

$$U := SPK_7\Big\{(\gamma_1, \ldots, \gamma_e, \delta, \varepsilon) : \ \tilde{z}_1 = h^{\gamma_1} g^{\delta} \ \wedge \ \tilde{z}_2 = h^{\gamma_2} \tilde{z}_1^{\delta} \ \wedge \ldots$$

$$\ldots \wedge \ \tilde{z}_{e-1} = h^{\gamma_{e-1}} \tilde{z}_{e-2}^{\delta} \ \wedge \ z = h^{\gamma_e} \tilde{z}_{e-1}^{\delta} \ \wedge \ z = g^{\varepsilon}\Big\}(m)$$

is a signature of the message $m \in \{0,1\}^*$ based on a proof of knowledge of the e-th root of the discrete logarithm of z to the base g. It is denoted

$$SPK_{14}\Big\{(\alpha):\ z = g^{\alpha^e}\Big\}(m)\,.$$

From the properties of the SPK_7 it follows that the interactive protocol corresponding to the SPK_{14} is a proof of knowledge of the values $\gamma_1, \dots, \gamma_e, \delta, \varepsilon$. This is equivalent to the knowledge of a value α which is the e-th root of the discrete logarithm of y with respect to the base g as follows from

$$\begin{aligned} z &= h^{\gamma_e}\Big(h^{\gamma_{e-1}}\big(\dots h^{\gamma_2}(h^{\gamma_1}g^{\delta})^{\delta}\dots\big)^{\delta}\Big)^{\delta} = \\ &= h^{\gamma_e+\gamma_{e-1}\delta+\dots+\gamma_2\delta^{e-2}+\gamma_1\delta^{e-1}}g^{\delta^e} = h^{\zeta}g^{\delta^e} = g^{\varepsilon} \end{aligned}$$

for $\zeta \equiv \gamma_e + \gamma_{e-1}\delta + \dots + \gamma_2\delta^{e-2} + \gamma_1\delta^{e-1} \pmod{n}$. Since the discrete logarithm of g to the base h is not known, one can know only one representation of z to the bases h and g. It follows that $\zeta \equiv 0 \pmod{n}$ and $\varepsilon \equiv \delta^e \equiv \alpha^e \pmod{n}$ and that the prover knows the e-th root of the discrete logarithm of z to the base g. Furthermore, the interactive protocol corresponding to SPK_{14} is also honest-verifier zero-knowledge, since the simulator can just choose the $\tilde{z}_1, \dots, \tilde{z}_{e-1}$ from G and then run the simulator corresponding to the SPK_7.

The system SPK_{14} could be made more efficient for some e's by using addition chains for the integer e (cf. Section 3.5). For larger e's it depends on the actual choice of the system parameter whether the SPK_{13} or the SPK_{14} is more efficient.

An $SPK_{14}\{(\alpha) : z = g^{\alpha^e}\}(m)$ can be computed if a value x in $\mathbb{Z}_n^*$ is known such that $z = g^{x^e}$: one first computes the values $\tilde{z}_i{:=}h^{r_i}g^{x^i}$ for $i = 1, \dots, e-1$ with randomly chosen $r_i \in \mathbb{Z}_n$. Then the signature of knowledge U is computed. Note that the elements $\tilde{z}_i$ are truly random group elements and therefore do not leak any information.

The last building block we need for our signature scheme is very similar to the SPK_{14}: a system for proving the knowledge of an e-th root of the g-part of the representation of a public key with respect to g and h. Such a system can be derived from the previous one just by omitting the part that an ε is known such that $z = g^{\varepsilon}$.

Definition 5.4. *An $(e-1)$-tuple $(\tilde{v}_1, \ldots, \tilde{v}_{e-1}) \in G^{e-1}$ and a signature*

$$U := SPK_7\Big\{(\gamma_1, \ldots, \gamma_e, \delta) : \tilde{v}_1 = h^{\gamma_1} g^{\delta} \wedge \tilde{v}_2 = h^{\gamma_2} \tilde{v}_1^{\delta} \wedge \ldots$$
$$\ldots \wedge \tilde{v}_{e-1} = h^{\gamma_{e-1}} \tilde{v}_{e-2}^{\delta} \wedge v = h^{\gamma_e} \tilde{v}_{e-1}^{\delta}\Big\}(m)$$

is a signature of the message $m \in \{0,1\}^*$ based on a proof of knowledge of an e-th root of the g-part of a representation of v to the bases h and g. It is denoted

$$SPK_{15}\Big\{(\alpha, \beta) : v = h^{\alpha} g^{\beta^e}\Big\}(m) .$$

The security properties of this scheme follow from those of the SPK_{14}.

5.4 The Basic Group Signature Scheme

To realize a group signature scheme along the lines described in Section 5.2, we need a suitable signature scheme for the group (membership) manager to issue certificates. It must allow group members to prove the possession of a signature on their membership keys, i.e., certificates, issued by the group manager. Furthermore, the scheme must avoid the possibility of existential forgery of membership certificates even when other certificates are already known. A good candidate is the RSA signature scheme (cf. Section 2.7), since it is possible to prove the knowledge of an e-th root modulo a composite. It remains to find a format for the membership keys. Due to the multiplicative properties of the RSA signature scheme (cf. Section 2.8), we must avoid formats that are maintained under modular multiplication. Furthermore, the format must allow group members to prove their knowledge of the secret key without showing their membership key. We propose the following format

$$1 + a^x \pmod{n},$$

where x is a group member's secret key and a an element of large multiplicative order modulo n. This choice leads to the following assumption.

Assumption 5.1. *Let n be an RSA-modulus, e an element of $\mathbb{Z}_{\varphi(n)}^*$, and $a \in \mathbb{Z}_n^*$ an element of large multiplicative order such that computing discrete* *logarithms to the base a is infeasible. We assume that it is hard to compute a pair (x, v) of integers such that*

$$v^e \equiv 1 + a^x \pmod{n}$$

holds, if the factorization of n is not known. Furthermore, we assume that this is true even when other pairs (v', x') are known that satisfy the above equation.

It seems infeasible to construct such values x and v without the help of the group manager: on one hand, when first choosing x, it is infeasible to compute the e-th root of $1 + a^x$ because the factorization of n is unknown. On the other hand, when first choosing v, it is infeasible to compute the discrete logarithm of $v^e - 1 \pmod{n}$ to the base a (it might even not exist). Furthermore, given several pairs (x, v), there seems to be no method for computing an additional pair.

In the remainder of this section, a first realization of a group signature scheme based on the above assumption is presented. In this scheme, the opening of signatures can even be realized in a simpler way than presented in Section 5.2.

5.4.1 System Setup

The group manager chooses the following values:

- an RSA public key (n, e), where $n = pq$, $p = 2p' + 1$, and $p = 2p' + 1$ with p, q, p', and q' all prime. The primes p and q are the group manager's secret key.
- a cyclic group $G = \langle g \rangle$ of order n in which computing discrete logarithms is infeasible (e.g., G could be a subgroup of $\mathbb{Z}_p^*$ with a prime p such that $n|(p-1)$ holds),
- an element $a \in \mathbb{Z}_n^*$ (a must be of large multiplicative order modulo both prime factors of n), and
- an upper bound λ on the length of group members' secret keys and a constant $\epsilon > 1$.

The group's public key is $\mathcal{Y} := (n, e, G, g, a, \lambda, \epsilon)$. It is published together with the other system parameters such as the description of a hash function $\mathcal{H}$, or the security parameter k.

To prevent framing attacks[6] by the group manager, she must provide proofs that she has chosen all parameters correctly. For proving that g has indeed order n there exists an efficient zero-knowledge proof system by Boyar el al. [BFL91]. For proving that n is of the right form, there is no efficient proof system to the best of our knowledge. Thus one has to use general zero-knowledge proof techniques (e.g., [BCC88, BP96, CD97]) and a circuit that takes as input integers p, q, p', and q' and outputs 1 if and only if they are primes and if $n = pq$, $p = 2p' + 1$, and $p = 2p' + 1$ holds. The size of p and q can be checked by the number of input bits for them (they should have at most $\lceil 0.5 \log n \rceil$ bits). This is not very efficient but must done only once. To verify that a has a large order in $\mathbb{Z}_n^*$, one needs only to test whether $a \notin \{-1, 1\}$ and that $\gcd(a^2 - 1, n) \nmid n$. This guarantees that the order of a is at least $p'q'$ since if $\mathrm{ord}(a) < p'q'$ then $a^2 \equiv 1$ modulo p or q as can easily be seen using the Chinese Remainder Theorem (cf. [GKR97]). We remark that for the group signature scheme presented in the next section, some of the above proofs can be omitted.

5.4.2 Generating Membership Keys and Certificates

To join the group, a prospective member Arto

- chooses his secret key x randomly from $\{0, \ldots, 2^\lambda - 1\}$,
- computes the value $y := a^x \pmod{n}$ and
- his membership key $z := g^y$, and
- commits himself to z (for instance by signing it).

To obtain a membership certificate, Arto sends to the group manager Maude the values y, z, his commitment to z, and

$$U := SPK_1\{(\alpha) : y = a^\alpha \pmod{n}\}(z).$$

[6] The different attacks by the group manager are discussed in Section 5.4.5.

The latter convinces Maude that y (and thus also z) has the required form. Maude verifies the values she obtains from Arto, and returns the membership certificate

$$v := (y+1)^{1/e} \pmod{n}$$

to him. Arto stores x, y, and v securely. The membership key z may get known to other group members or even to non-members.

5.4.3 Signing Messages

To sign a message $m \in \{0,1\}^*$, Arto computes the following values:

- $\tilde{g} := g^r$ for $r \in_R \mathbb{Z}_n^*$
- $d := \tilde{g}^y (= z^r)$
- $V_1 := SPK_{14}\{(\beta) : d\tilde{g} = \tilde{g}^{\beta^e}\}(m)$
- $V_2 := SPK_{12}\{(\alpha) : d = \tilde{g}^{a^\alpha}\}(V_1)$

The resulting signature of the message m consists of ($\tilde{g}$, d, V_1, V_2) and can be verified by checking the correctness of V_1 and V_2. Note that, rather than encrypting the message m in the value d as described in Section 5.2, m is included in the argument to the hash function of V_1. The value d can be seen as a kind of "Diffie-Hellman encryption" of z.

If e is large, it might be more efficient to use an SPK_{13} instead of SPK_{14} for V_1.

Proposition 5.3. *In the random oracle model the following holds. If one can compute a tuple (signature) ($\tilde{g}$, d, V_1, V_2) for a message $m \in \{0,1\}^*$ such that*

$$V_1 = SPK_{14}\{(\beta) : d\tilde{g} = \tilde{g}^{\beta^e}\}(m)$$
$$V_2 = SPK_{12}\{(\alpha) : d = \tilde{g}^{a^\alpha}\}(V_2)$$

holds, then one can also compute a valid membership certificate. Therefore, under Assumption 5.1, only group members (or the group manager manager) can compute such a tuple.

Proof. We already know that signatures of the type SPK_{14} and SPK_{12} are secure in the random oracle model. Hence, if one can compute a signature V_2, then one can compute a value α such that

$$d = \tilde{g}^{a^{\alpha}} \text{ and therefore } d\tilde{g} = \tilde{g}^{a^{\alpha}+1}$$

holds. If one can further generate a signature V_1, then one can also compute e-th root of $a^{\alpha}+1$ modulo n. This is a valid membership certificate of a membership key. □

5.4.4 Opening Signatures

Linking two signatures $(\tilde{g}, d, V_1, V_2)$ and $(\tilde{g}', d', V_1', V_2')$, i.e., deciding whether these signatures have been issued by the same group member or not, is only possible by deciding whether the equality

$$\log_{\tilde{g}} d \equiv \log_{\tilde{g}'} d' \pmod{n}$$

holds. If one could decide this, then one could also solve the Decision Diffie-Hellman problem. Solving the latter is widely believed to be infeasible. Thus, we conclude that signatures are anonymous and unlinkable. However, the group manager has an advantage: she knows the relatively few possible values of $\log_{\tilde{g}} d$, namely the y's of the group members, and can therefore perform this test. More precisely, given a signature $(\tilde{g}, d, V_1, V_2)$ of a message m, the group manager can find out which one of the group members issued this signature by testing

$$\tilde{g}^{y_P} \stackrel{?}{=} d$$

for all group members P (here y_P denotes discrete logarithm of P's membership key z_P to the base g). A proof of this fact consists of the signer's membership key z_P, his commitment to this key, and of

$$SPK_6\{(\alpha) : z_P = g^{\alpha} \wedge d = \tilde{g}^{\alpha}\}(z_P).$$

Unfortunately, this method is impractical for very large groups. In the next section we will present an extension that enables Maude to identify signers directly.

5.4.5 Security Properties

Let us discuss the security properties of the scheme.

Signatures are unlinkable and anonymous: This property holds due to the hardness of the Decision Diffie-Hellman problem (DDHP), as is explained in previous paragraph. Furthermore, note that the signatures V_1 and V_2 do not leak useful information since their interactive counterparts are honest-verifier zero-knowledge.

Non-members cannot sign: This is Proposition 5.3.

Group members cannot sign on behalf of other group members: Besides the knowledge of a certificate on their own membership key, group members do not have more information than non-members. Thus, signing on behalf of another group member would require a computation of the double discrete logarithm x' of z and then a computation of the e-th root of $1 + a^x \pmod{n}$. Both is assumed to be infeasible.

The group manager cannot falsely accuse members: Since the integer a has large order in $\mathbb{Z}_n^*$ and since $\varphi(n)$ is not smooth, the group manager cannot compute a group member's secret key and thus is not able to impersonate group members. Finally, because of the proof the group manager must provide as evidence in the procedure open, she cannot falsely accuse members.

5.4.6 Efficiency Considerations

With the following choices of the system parameters

$$\ell = 160, k = 64, \lambda = 170, \epsilon = 4/3, |n| = 600, \text{and } e = 3,$$

a signature is about 2.5 Kbyte long and signing of messages or for verifying signatures require the computation of approximately 67'000 modular multiplications with a 600 bit modulus (this corresponds to about 75 exponentiations with full 600 bit exponents). If an SPK_{13} is used for V_1 instead of an SPK_{14}, i.e., if e is large, a signature is less than 7 Kbyte long and requires about 140'000 multiplications.

5.5 An Advanced Scheme

In this section we extend the group signature scheme of the previous section such that it is possible to split the group manager into a membership manager and a revocation manager. As an immediate consequence, the membership manager must be prevented from learning the value y and the membership certificate. This can be solved by sending the group manager only g^y and by using a blind signature scheme for issuing the certificate. Furthermore, the method for opening signatures as described in Section 5.2 must be applied. This also makes the opening of signatures much more efficient.

5.5.1 System Setup

We now distinguish the two roles of the group manager and assign them to the membership manager Maude and the revocation manager Rose.

The membership manager Maude chooses

- an RSA public key (n, e), where $n = pq$ (the primes p and q are her secret key),
- an element $a \in \mathbb{Z}_n^*$ (a should be of large multiplicative order modulo both prime factors of n), and
- a cyclic group $G = \langle g \rangle$ of order n in which computing discrete logarithms is infeasible (e.g., G could be a subgroup of $\mathbb{Z}_p^*$, for a prime p with $n|(p-1)$),
- an element $h \in G$ the logarithm to the base g of which must not be known, and
- an upper bound λ on the length of group members' secret keys and a constant $\epsilon > 1$.

The revocation manager chooses a secret key $\rho \in_R \mathbb{Z}_n^*$ and computes her public key $y_R := h^\rho$.

The group's public key is $\mathcal{Y} := (n, e, G, g, h, a, \lambda, \epsilon, y_R)$. It is published together with the other system parameters.

Different from the previous scheme, the membership manager is only required to prove that $n = pq$ holds for two large primes p and q, that g has order n, and that h and a are chosen pseudo-randomly. In this scheme, this is enough to assure the group members that she cannot compute their secret keys. For the other parameters, it is in the interest of the membership manager to choose them such that computing certificates is hard without her help (she would be held responsible if a membership key appears for which she cannot provide a commitment of a group member). Finally, the revocation manager must prove that she knows $\log_h y_R$. This can be achieved with a simple SPK_1-type signature.

5.5.2 Generating Membership Keys and Certificates

To become a group member, Arto

- chooses his secret key $x \in_R \{0, \ldots, 2^\lambda - 1\}$,
- computes the value $y := a^x \pmod{n}$ and
- his membership key $z := g^y$,
- and commits himself to z.

Since the membership manager should no longer learn y, certificates must be issued with the blind RSA-signature scheme of Chaum [Cha84] (cf. Section 2.8). This can be done as follows. Arto sends the group manager a randomized y, namely

$$\tilde{y} := r^e(y+1) \pmod{n} \quad \text{for } r \in_R \mathbb{Z}_n^*,$$

together with z and his commitment to z. To convince Maude that z is of the right form and that the value $1 + \log_g z \pmod{n}$ is contained in the randomized (blinded) value $\tilde{y}$, Arto has to provide her the two signatures

- $U_1 := SPK_{13}\{(\beta) : g^{\tilde{y}} = (zg)^{\beta^e}\}(z \| \tilde{y})$
- $U_2 := SPK_{12}\{(\alpha) : z = g^{a^\alpha}\}(U_1)$

The second signature U_2 assures Maude that z has the format of a membership key and that Arto knows the corresponding secret key. The first signature U_1 proves that $\tilde{y}$ is correctly randomized. After successfully verifying all values obtained from Arto, Maude sends him the e-th root $\tilde{v} := \tilde{y}^{1/e} \pmod{n}$ of $\tilde{y}$. From this Arto can compute his membership certificate:

$$v := \frac{\tilde{v}}{r} \equiv \frac{\tilde{y}^{1/e}}{r} \equiv \frac{(r^e(y+1))^{1/e}}{r} \equiv (y+1)^{1/e} \pmod{n}.$$

Arto stores x,y, and v securely.

5.5.3 Signing Messages

To sign a message m, Arto computes the values

- $\tilde{g} := g^r$ and $\tilde{z} := \tilde{g}^y$ for $r \in_R \mathbb{Z}_n^*$,
- $d_1 := y_R^u g^y$ and $d_2 := h^u$ for $u \in_R \mathbb{Z}_n^*$,
- $V_1 := SPK_7\{(\gamma, \delta) : \tilde{z} = \tilde{g}^\gamma \wedge d_2 = h^\delta \wedge d_1 = y_R^\delta g^\gamma\}(m)$,
- $V_2 := SPK_{14}\{(\beta) : \ \tilde{z}\tilde{g} = \tilde{g}^{\beta^e}\}(V_1)$, and
- $V_3 := SPK_{12}\{(\alpha) : \ \tilde{z} = \tilde{g}^{a^\alpha}\}(V_2)$,

The resulting signature of the message m consists of the tuple ($\tilde{z}$, $\tilde{g}$, d_1, d_2, V_1, V_2, V_3) and can be verified by checking the correctness of the SPK's V_1, V_2, and V_3.

Proposition 5.4. *In the random oracle model the following holds. If one can compute a tuple (signature) ($\tilde{z}$, $\tilde{g}$, d_1, d_2, V_1, V_2, V_3) for a message m such that*

$$\begin{aligned} V_1 &= SPK_7\{(\gamma, \delta) : \tilde{z} = \tilde{g}^\gamma \wedge d_2 = h^\delta \wedge d_1 = y_R^\delta g^\gamma\}(m) \\ V_2 &= SPK_{14}\{(\beta) : \ \tilde{z}\tilde{g} = \tilde{g}^{\beta^e}\}(V_1) \\ V_3 &= SPK_{12}\{(\alpha) : \ \tilde{z} = \tilde{g}^{a^\alpha}\}(V_2) \end{aligned}$$

holds, then one can also compute a valid membership certificate. Therefore, under Assumption 5.1, only a group member (or the membership manager) can compute such a tuple. Furthermore, the membership key of the group member who originated the signature is ElGamal-encrypted in (d_1, d_2) under the revocation manager's public key.

Proof. Signatures of the type SPK_{14}, SPK_{12}, and SPK_7 are secure in the random oracle model. Hence, if one can compute V_3 then one can also compute an integer α such that

$$\tilde{z} = \tilde{g}^{a^\alpha}$$

holds and therefore we have

$$\tilde{z}\tilde{g} = \tilde{g}^{a^\alpha + 1} .$$

If one can further generate V_2, then one can also compute an e-th root of $a^\alpha + 1$ modulo n, i.e., a membership certificate. This proves the first part.

From the properties of the SPK_7-type signature V_1 it follows that (d_1, d_2) encrypts $g^{\log_{\tilde{g}} \tilde{z}}$. Because of V_2 and V_3, and under Assumption 5.1, this is the membership key of the originator of the signature.

□

5.5.4 Opening Signatures

If the revocation manager wants to open a signature ($\tilde{z}$, $\tilde{g}$, d_1, d_2, V_1, V_2, V_3) of the message m, she decrypts z by computing d_1/d_2^ρ. From Proposition 5.4 we know that z must be the membership key of the originator. To prove that z is indeed encrypted in d_1 and d_2, she computes

$$SPK_6\{(\alpha) : \; d_1 z^{-1} = d_2^\alpha \;\wedge\; y_R = h^\alpha\}(z).$$

5.5.5 Security Properties

The security properties of this scheme are as follows.

Signatures are unlinkable and anonymous: Linking two signatures ($\tilde{z}$, $\tilde{g}$, d_1, d_2, V_1, V_2, V_3) and ($\tilde{z}'$, $\tilde{g}'$, d_1', d_2', V_1', V_2', V_3') would require deciding whether

$$\log_{\tilde{g}} \tilde{z} \equiv \log_{\tilde{g}'} \tilde{z}' \pmod{n} \quad \text{or} \quad \log_{y_R} \frac{d_1}{d_1'} \equiv \log_h \frac{d_2}{d_2'} \pmod{n}$$

holds. If this could be decided efficiently, then the DDHP could be solved efficiently. Hence the two signatures are unlinkable.

Deciding whether a given signature $(\tilde{z}, \tilde{g}, d_1, d_2, V_1, V_2, V_3)$ originated from a member having the membership key $z = g^y$ requires the decision whether

$$\log_g z \equiv \log_{\tilde{g}'} \pmod{n}$$

holds. Again, solving this efficiently implies solving the DDHP efficiently. Thus signatures are also anonymous.

Non-members cannot sign: This is Proposition 5.4.

Group members cannot sign on behalf of other group members: This holds due to the same reasons as for the previous scheme.

The group manager cannot falsely accuse members: Since g has order n and n is the product of two large primes, the membership manager cannot compute secret keys of members and hence cannot sign on their behalf. The revocation manager cannot falsely accuse members since she could not provide such a proof as evidence in the procedure open.

5.5.6 Efficiency Considerations

With the same system parameters as for the previous scheme, a signature is less than 3 Kbyte long and the operations for signing messages or for verifying signatures require the computation of approximately 72'000 modular multiplications with a 600 bit modulus (this corresponds to about 80 exponentiations with full 600 bit exponents). If an SPK_{13} is used for V_1 instead of an SPK_{14}, i.e., if e is large, a signature is less than 7 Kbyte and requires 147'000 multiplications. Compared to the previous scheme, all of the numbers are slightly larger.

5.6 A More Efficient Variant

When analyzing the efficiency of the schemes described in the previous two sections, one finds that the main part of the computations that are needed for signature generation and verification as well as most parts of a group-signature are due to the SPK_{12}-type signatures. To construct a more efficient group signature scheme, it seems thus most promising to replace the format for membership keys by one that does not use exponentiation with the secret key of a group member. We therefore propose the following format

$$f_1 x^{e_1} + f_2 \pmod{n}$$

where x is a group member's secret and the values e_1, f_1, and f_2 are system parameters. As before, a certificate is an RSA-signature of the membership manager of this "message". This format leads to the following assumption.

Assumption 5.2. *Let n be an RSA-modulus with unknown factorization. We assume that there exist two integers f_1 and f_2 in $\mathbb{Z}_n^*$ and two small integers e_1 and e_2 in $\mathbb{Z}_{\varphi(n)}^*$ such that it is hard to compute values x and v such that*

$$v^{e_2} \equiv f_1 x^{e_1} + f_2 \pmod{n}$$

holds. Furthermore, we also assume it to be hard to compute such values when already knowing other pairs (x', v') satisfying the above equation.

Clearly, the difficulty of computing such a pair (x, v) relies on the hardness of the RSA-problem. Moreover, it depends also on the choices of the values e_1, e_2, f_1, and f_2. For instance, for the choice $e_1 = 2, e_2 = 2$, and $f_1 = 1$, which is related to the Ong-Schnorr-Shamir signature scheme [OSS84], it is not difficult to compute such pairs for any value of f_2 as is shown in [AEM87, PS87]. Generally, it is regarded as an open problem to determine which types of polynomial congruences with composite moduli are hard to solve [McC90b].

Furthermore, for some choices of e_1, e_2, f_1, and f_2, it is easy to compute new pairs (v, x) from other pairs. For instance, if the equation $v^{e_2} \equiv f_1 x^{e_1} + f_2 \pmod{n}$ defines an elliptic curve, the addition-operation of the algebraic group defined by the curve can be used to compute a new pair from known pairs.

As a challenge, we propose the following parameters:

- $e_1 = 5, e_2 = 3$,
- $f_1 = 1$, and f_2 such that its 3rd root is hard to compute.

A possibility to make it harder to forge membership certificates is to modify the group signature scheme in such a way that solutions of the polynomial equation are only accepted if they meet additional requirements. For instance, by modifying the SPK_{14}-type signature V_1, one can efficiently prove that the secret key x is smaller than $\sqrt{n}$ (the techniques for this are similar to those used for the SPK_{12} signatures).

Finally, let us remark that one could also use several polynomials simultaneously, e.g.,

$$\begin{aligned} v_1^{e_{10}} + f_{11}v_2^{e_{11}} + f_{12}x^{e_{12}} &\equiv 0 \pmod{n} \\ v_1^{e_{20}} + f_{21}v_2^{e_{21}} + f_{22}x^{e_{22}} &\equiv 0 \pmod{n} \end{aligned}$$

instead of a single modular polynomial. Such a certification scheme would still yield an efficient group signature scheme.

5.6.1 System Setup

Again, we distinguish the two roles of the group manager. The membership manager Maude chooses the following values:

- an RSA modulus n and two public exponents $e_1, e_2 > 1$, such that e_2 is relatively prime to $\varphi(n)$ (the factorization of n is the membership manager's secret key),
- two integers $f_1, f_2 > 1$ whose e_1-th roots and e_2-th roots are not known,
- a cyclic group $G = \langle g \rangle$ of order n in which computing discrete logarithms is infeasible,
- an element $h \in G$ whose discrete logarithm to the base g is not known.

The revocation manager Rose chooses her secret key ρ randomly from $\mathbb{Z}_n$ and computes her public key $y_R := h^\rho$.

The group's public key consists of $\mathcal{Y} := (n, e_1, e_2, f_1, f_2, G, g, h, y_R)$ and is published together with the other system parameters.

To assure that all parameters were correctly chosen, the membership manager and the revocation manager must provide exactly the same proofs as in the system-setup of the previous section's scheme.

5.6.2 Generating Membership Keys and Certificates

To become a group member, Arto

- chooses $x \in_R \mathbb{Z}_n^*$ and
- computes $y := x^{e_1} \pmod{n}$ and
- $z := g^y$.

As was the case in the scheme of the previous section, the membership manager is not allowed to learn y and thus certificates must be issued using the blind RSA-signature scheme. To do so, Arto computes

- $\tilde{y} := r^{e_2}(f_1 y + f_2) \pmod{n}$ for $r \in_R \mathbb{Z}_n^*$,
- $U_1 := SPK_{15}\{(\beta) : \ g^{\tilde{y}} = (z^{f_1} g^{f_2})^{\beta^{e_2}} \}(z)$, and
- $U_2 := SPK_{15}\{(\alpha) : \ z = g^{\alpha^{e_1}} \}(U_1)$

and sends $\tilde{y}, z, U_1$, and U_2 to the membership manager. If U_1 and U_2 are correct, she sends Arto the "randomized" certificate

$$\tilde{v} := \tilde{y}^{1/e_2} \pmod{n}$$

back. Arto de-randomizes this value and obtains his membership certificate

$$v := \tilde{v}/r \equiv (f_1 y + f_2)^{1/e_2} \pmod{n}.$$

Let us now explain what U_1 and U_2 actually mean to Maude. The signature U_2 shows that the element z is of the form $g^{\alpha^{e_1}}$ for some α Arto knows. The signature U_1 assures that $\tilde{y} \equiv \beta^{e_2}(f_1 \alpha^{e_1} + f_2) \pmod{n}$ holds for some β Arto knows, and therefore she can conclude that $\tilde{y}$ is correctly randomized.

5.6.3 Signing Messages

To sign a message m on behalf of the group, Arto performs the following computations:

- $d_1 := y_R^r g^y$ for $r \in_R \mathbb{Z}_n^*$,
- $d_2 := h^r$,
- $V_1 := SPK_7\{(\varepsilon, \zeta) : d_2 = h^\varepsilon \ \wedge \ d_1 = y_R^\varepsilon g^\zeta\}(m)$,
- $V_2 := SPK_{15}\{(\gamma, \delta) : d_1^{f_1} g^{f_2} = y_R^\gamma g^{\delta^{e_2}}\}(V_1)$, and
- $V_3 := SPK_{15}\{(\alpha, \beta) : d_1 = y_R^\alpha g^{\beta^{e_1}}\}(V_2)$.

The resulting signature of the message m consists of $(d_1, d_2, V_1, V_2, V_3)$ and is valid if the three SKP's V_1, V_2, and V_3 are valid.

Remark 5.1. The signatures V_1 and V_3 could be replaced by the single signature

$$V_3' := SPK_{15^*}\{(\alpha, \beta) : d_2 = y_R^\alpha g^{\beta^{e_1}} \ \wedge \ d_1 = h^\alpha\}(m)$$

by slightly extending the definition of the SPK_{15}-type signatures.

Proposition 5.5. *In the random oracle model the following holds. If one can compute a tuple $(d_1, d_2, V_1, V_2, V_3)$ of a message m such that*

$$\begin{aligned}
V_1 &= SPK_7\{(\varepsilon, \zeta) : d_2 = h^\varepsilon \ \wedge \ d_1 = y_R^\varepsilon g^\zeta\}(m) \\
V_2 &= SPK_{15}\{(\gamma, \delta) : d_1^{f_1} g^{f_2} = y_R^\gamma g^{\delta^{e_2}}\}(V_1) \\
V_3 &= SPK_{15}\{(\alpha, \beta) : d_1 = y_R^\alpha g^{\beta^{e_1}}\}(V_2)
\end{aligned}$$

holds, then one can also compute a valid certificate. Therefore, under Assumption 5.2, only a group member can compute such a tuple. Furthermore, the membership key of the group member who originated the signature is ElGamal-encrypted in (d_1, d_2) under the revocation manager's public key.

Proof. Signatures of the type SPK_{14}, SPK_{12}, and SPK_7 are secure in the random oracle model. Thus, if one can compute V_3, one can also compute a pair (α, β) such that

$$d_1 = y_R^\alpha g^{\beta^{e_1}}$$

holds. Moreover, if one can compute the signature V_2, one can also compute a pair (γ, δ) such that

$$d_1^{f_1} g^{f_2} = y_R^\gamma g^{\delta^{e_2}} \quad \text{and thus} \quad y_R^\gamma g^{\delta^{e_2}} = (y_R^\alpha g^{\beta^{e_1}})^{f_1} g^{f_2} = y_R^{f_1 \alpha} g^{f_1 \beta^{e_1} + f_2}$$

holds. Then we have either

$$f_1\alpha \equiv \gamma \pmod{n} \quad \text{and} \quad \delta^{e_2} \equiv f_1\beta^{e_1} + f_2 \pmod{n}.$$

or, otherwise, one could compute the discrete logarithm of y_R to the base g. Since the latter assumed to be infeasible, the above must hold, and therefore δ is a valid certificate and β is the secret key of a membership.

Finally, V_1 and V_3 together guarantee that the pair (d_1, d_2) is an ElGamal encryption of $g^{\beta^{e_1}}$ (or, if it's not, one could compute the discrete logarithm of y_R to the base g), which is the membership key under Assumption 5.2. □

5.6.4 Opening Signatures

If the revocation manager Rose wants to open a signature ($\tilde{z}$, d, V_1, V_2, V_3) of the message m, she decrypts z by computing d_1/d_2^{ρ} which corresponds to the membership key of the signer. To prove that z is indeed encrypted in d_1 and d_2, Rose provides

$$SPK_6\{(\alpha): \; d_1 z^{-1} = d_2^{\alpha} \;\wedge\; y_R = h^{\alpha}\}(z),$$

which she can do because α corresponds to her secret key.

5.6.5 Security Properties

The security properties of the last scheme of this chapter are as follows.

Signatures are unlinkable and anonymous: This is due to the same argument as for the previous scheme, with the difference that we are only given d_1 and d_2.

Non-members cannot sign: This is Proposition 5.5.

Group members cannot sign on other group members' behalf: Signing in the name of a group member would require a computation of the discrete logarithm y' of z, the e_1-th root of this y', and the e_2-th root of $f_1 y' + f_2 \pmod{n}$. This is assumed to be infeasible.

The group manager cannot falsely accuse members: This is guaranteed in the same way as for the previous scheme.

5.6.6 Efficiency Considerations

For the choice

$$e_1 = 5, e_2 = 3, k = 160, \text{ and } |n| = 600,$$

a signature is about 1.4 Kbyte long and the operations for signing for verifying signatures require the computation of approximately 18'000 modular multiplications with a 600 bit modulus (this corresponds to about 20 exponentiations with full 600 bit exponents). Signature generation and verification are about four times faster, and a signature is about half the size, compared to the two preceding schemes.

5.7 Extensions

In this section we consider how the functionality of the membership and revocation managers can be shared among several entities such that only defined subsets of these can issue certificates (or revoke group members' anonymity) together. This reduces the risk of fraudulent anonymity revocation and of issuing certificates to persons not entitled to become group members. We also consider how the schemes can be extended to generalized group signature schemes.

5.7.1 Sharing the Functionality of the Group Manager

For the first scheme, more efficient solutions for sharing the functionalities of the group manager than general multi-party computation do not seem achievable. We discuss the two other schemes.

The role of the revocation manager can easily be distributed amongst several entities by sharing the revocation manager's secret key ρ among them. For this, any secret sharing scheme can be used that allows the share holders to jointly and securely compute b^ρ for any element $b \in G$. This is discussed in Chapter 7.

The membership manager has two different but related tasks. Namely, generating most of the parameters in the setup phase and issuing membership certificates. For issuing certificates, one could use the robust RSA threshold signature schemes proposed by Gennaro et al. [GJKR96a, Gen96]. For the generation of the system parameters, either a general multi-party protocol [GMW87a, Fra93, Can95], the scheme of Boneh and Franklin [BF97], or a trusted party can be used. The latter has the disadvantage that the party choosing the RSA-modulus n could also issue certificates since he/she knows the factorization of n. The first solution is not very efficient, but acceptable, since the system parameters only need to be generated once.

5.7.2 Generalized Group Signature Schemes

One way to extend the group signature schemes presented in this chapter to generalized group signature schemes is to generate a separate membership key for each coalition and then to share the corresponding secret key among the group members belonging to that coalition. This of course has the drawback that a group member has to store a (share of a) secret key and a membership certificate for each coalition he belongs to.

For the first certification scheme, where we have $y \equiv a^x \pmod{n}$, the secret key x can be shared among the group members using a suitable secret sharing scheme (cf. Chapter 7). Then all computations involving x must be carried out by all these group members together. For the second scheme, this would be the signature U_2 and V_3. The other computations could be carried out by any group member of the coalition.

For the second certification scheme, we have $y \equiv x^{e_2} \pmod{n}$. Here, x can be chosen as the product of the shares, while y and the certificate v can be known to all members of the coalition. Computing g^x and $g^{x^{e_1}}$ can then be achieved simply by repeatedly powering g with x_i or $x_i^{e_1}$. This does not leak additional information about the shares.

A generalized group scheme with a k-threshold authority-structure can be also constructed in such a way that each member no longer has to store information for each coalition he belongs to. However, this is possible only at the cost that the length of a signature depends on k. The idea is that a signature of the threshold group signature scheme is defined to consist of at least k ordinary group signatures from different group members. However, checking that the k signatures are from *different* group members is not possible directly, since signatures are anonymous. Thus, the signature of the threshold group signature scheme must also contain a signature based on a proof that the randomized membership keys are indeed different. This is indeed possible for instance by providing the signature

$$SPK_2\left\{(\alpha,\beta) : g = y_R^{\alpha}\left(\frac{\tilde{d}_1}{d_1}\right)^{\beta}\right\} \qquad \text{for } 1 \le i < k,\ i < j \le k,$$

where $\tilde{d}_1$ and d_1 are encryptions of membership keys provided as a part of the group signatures as defined in Section 5.5 and 5.6. Since $k(k-1)/2$ such signatures are necessary to prove that all membership keys are different, this is rather inefficient.

If the threshold group signature scheme is based on the simple group signature scheme in Section 5.5, there is a simpler way to verify that the randomized membership keys are indeed different[7]: requiring the signing group members to use the same $\tilde{g}$. Then the $\tilde{z}_i = \tilde{g}^{\tilde{y}_i}$'s are equal if the y_i's are equal. This has the advantage over the solution above that the extra SPK's can be omitted and thus signatures are shorter, but the disadvantage that using the same $\tilde{g}$ for all $\tilde{z}_i$'s might leak information about the membership keys.

5.8 Comparison of the Different Group Signature Schemes

As mentioned before, the main difference between the group signature scheme presented in this chapter and schemes previously proposed is that all parameters of schemes in this chapter are independent of the number of group members. In particular, the size of the group's public key and the length of signatures, as well as all computations for any operation do not depend on the size of the group. Furthermore, new group members can be added without changing the group's public key. The schemes even conceal the size or any other structure of the group. However, removing members from the group is not possible without changing the group's public key and therefore issuing new certificates to all remaining group members. This is an advantage of the "traditional" schemes such as those presented in the previous chapter. There,

[7] The scheme of Section 5.6 could be modified such that this approach would also work.

adding *and* removing a group member can be achieved by modifying only the group's public key; current members not involved can keep their secret keys.

When considering generalized group signature schemes, the scheme presented in the previous chapter has the advantage that each group member needs to store only a single secret key, regardless of how many coalitions he belongs to. However, the scheme completely reveals the structure of the group.

5.9 Open Problems

The main open problems to study are the two Assumptions 5.1 and 5.2, i.e., the difficulty of computing membership certificates without the help of the membership manager. A further direction of new research is to find other certification schemes, especially schemes the security of which can be proven related to some well studied cryptographic problems. Finally, it would be nice to find a method for excluding members without modifying the group's public key and/or requiring that the remaining members get new certificates.

Chapter 6

Payment Systems with Passive Trustees

Anonymous digital payment systems are an important ingredient to electronic commerce over computer networks such as the internet. They allow customers to instantaneously pay goods. Furthermore, by enabling anonymous payments, they protect the customer's privacy. Unfortunately, anonymity also leaves the door open for the misuse by criminals. Therefore, in order to investigate crimes, it must be possible to revoke anonymity. In this chapter we present the an efficient anonymous digital payment system satisfying this requirement. It can be used either as on-line payment system or, with extensions, as an off-line payment system. The system builds on the primitives introduced in Chapter 3. These primitives are used as signatures as well as different interactive protocols, i.e., as blind signature schemes, and protocols whereby two entities share a secret key and then jointly compute an SPK.

The work presented in this chapter appeared in [CMS97] and in parts in [CMS96].

6.1 Introduction

Anonymity of the participants is an important requirement for some applications in electronic commerce, in particular for payment systems. In most presently used payment systems the protection of the user's privacy relies exclusively on administrative and legal measures. Using cryptographic tools, in particular blind signature schemes [Cha84], it is possible to design electronic payment systems that allow the customers to remain anonymous (e.g., [Bra93, Cha85, CFN90, CPS94b, OO92]), without affecting the other security requirements. However, while protecting the honest customers' privacy, the anonymity also opens the door for misuse by criminals, for instance for perfect blackmailing [vSN92] or for money laundering.

Therefore, in order to make anonymous payment systems acceptable to governments and banks, they must provide mechanisms for revoking a participant's anonymity under certain well-defined conditions. Such anonymity revocation must be possible only for an authorized trusted third party or for a set of such parties. In this chapter we refer to trusted third parties as *trustees*. In a concrete scenario a trustee would be a judge or a law enforcement agency.

The concept of *anonymity-revocable payment systems*, sometimes called fair payment systems, was introduced independently in [BGK95] and [SPC95]. The customer's privacy cannot be compromised neither by the bank nor by the payee, even if they collaborate, but the trustee or a specified set of trustees can (in cooperation with the bank) revoke a customer's anonymity. It is understood that the trustee(s) answer a request only if there exists sufficient evidence that a transaction is not lawful.

All previously proposed anonymity-revocable systems are either inefficient because they are based on the cut-and-choose paradigm [BGK95, SPC95], or they require the trustee's participation in the opening of accounts or even in the withdrawal transactions [CPS95, CPS96, JY96, SPC95]. From an operational point of view, it is an important requirement that a trustee can be passive, i.e., that he need not be involved in regular transactions or when a customer opens a new account. This chapter is reports on the first efficient anonymous digital payment systems satisfying this requirement.

After these results were been presented at ESORICS'96 (see [CMS96]) Frankel, Tsiounis, and Yung proposed a different, somewhat less efficient solution to this problem in [FTY96].

6.2 Digital Payment Systems

An electronic payment system consists of a set of protocols between three interacting parties: a bank, a customer (the payer), and a shop (the payee). The customer and the shop have accounts with the bank. The goal of the system is to transfer money in a secure way from the customer's account to the shop's account. It is possible to identify three different phases: a *withdrawal phase* involving the bank and the customer, a *payment phase* involving the customer and the shop, and a *deposit phase* involving the shop and the bank. In an *off-line* system, each phase occurs in a separate transaction, whereas in an *on-line* system, payment and deposit take place in a single transaction involving all three parties.

The bank, the shop and the customer have different security requirements. The bank wants a guarantee that money can be deposited only if it has been previously withdrawn. In particular, double-spending of digital money must be impossible. The shop, upon receiving a payment in an off-line system, must be sure that the bank will accept the payment. Finally, the customer must be sure that the withdrawn money will later be accepted for a payment and that the bank cannot claim that the money has already been spent (called a framing attack), i.e., falsely accuse him of double-spending. Furthermore, the customer may require that his privacy be protected, i.e., that payments are anonymous and unlinkable. A payment is called *anonymous*, if the bank can by no means find out which customer withdrew the money that a shop[1] wants to deposit. Payments are called *unlinkable* if the bank cannot tell whether two payments were made by the same customer or not. Unlinkability is a stronger requirement than anonymity. Payment systems providing at least anonymity of payments, are called anonymous payment systems. However, most anonymous payment systems provide both, and, to grasp the difference, are sometimes said to provide *perfect anonymity*.

[1] Throughout, we assume that shops can not identify customers

An anonymous digital payment system can be realized with blind signature schemes. Upon charging the customers account, the bank blindly signs a message (often a random number) for the customer. The bank's signature makes this message a valid coin having an agreed value. Different denominations can be realized by having the bank to use a different public key for every denomination. The customer can use the coin in a payment to a shop. Due to the properties of blind signature schemes, the bank has no information on the coin (apart from its denomination). Therefore, payments are anonymous and unlinkable.

An obvious problem with such a scheme is that money can in principle be spent more than once. In an on-line system, double-spending can be prevented by the bank by checking the record of previous deposits. This requires that all deposit transactions (at least within the validity period of the bank's public key) are stored by the bank. In an off-line system, double-spending cannot be prevented, but it is possible to design systems that allow the revocation of a customer's anonymity if the money is spent more than once. This can be achieved by assuring that the customer's identity is properly encoded in the signed message and by having the customer answer a challenge message during the payment in such a way that the customer's identity can be computed from the answers to two different challenges. Alternatively, the anonymity revoking mechanism we present herein could be used, but this is not the main purpose of presenting the mechanism.

6.3 Anonymity Revocation by a Trustee

Anonymity revocation by a trustee means that, when the need arises, the trustee can link a withdrawal transaction with the corresponding deposit transaction. There are two types of anonymity revocation, depending on which kind of information is available to the trustee:

Withdrawal-based anonymity revocation: Based on the bank's view of a withdrawal transaction, the trustee can compute a piece of information that can be used (by the bank or a payee) to recognize the money when it is spent later. This type of anonymity revocation can for instance be used in case of blackmailing. When the owner of an account is forced to withdraw money and to transfer it to an anonymous criminal, the account owner could secretly inform the bank and the trustee could be asked to compute a value that can be put on a black list and linked with the money when it is deposited. This corresponds to putting the serial-number of a conventional bank-note on a blacklist.

Payment-based anonymity revocation: Based on the bank's view of a deposit transaction, the trustee determines the identity of the person who withdrew the money. This may be needed, for instance, when there is a suspicion of money laundering.

One of the security requirements of such a payment system is that the trustee must be capable only of anonymity revocation, and cannot play a different role in the system. In particular, the trustee must be unable to forge money.

It it possible to distinguish three different approaches achieving the above goals according to the type of the trustee's involvement.

1. The trustee is involved in every withdrawal. In such systems [CPS95, JY96] the trustee plays the role of an intermediary during the withdrawal protocol and performs the blinding operation on behalf of the customer. The trustee can then trivially revoke the anonymity if needed.

2. The trustee is involved in the opening of accounts, but not in transactions (e.g., [CPS96]). Such systems are potentially more efficient because normally an account is used for many transactions.

3. The trustee is not involved in any of the protocols of the payment system but is needed only for anonymity-revocation. In such systems the customer proves to the bank in the withdrawal protocol that the coin and the exchanged messages contain information, encrypted under the trustee's public key, that allows the revocation of anonymity. This can in principle be achieved by application of the well-known cut-and-choose paradigm, as described independently in [BGK95] and [SPC95]. However, such a system is quite inefficient as explained in Section 6.7

The goal of this chapter is to present an efficient anonymity-revocable payment system that allows both types of anonymity revocation and in which, in contrast to the previously proposed efficient systems, the trustee is completely passive unless he is asked to revoke the anonymity of a person. In particular, after initially publishing a public key, the trustee need neither be involved in the opening of an account nor in any withdrawal, payment, or deposit transaction.

6.4 An Efficient Anonymous Payment System with a Passive Anonymity-Revoking Trustee

In this section we describe the on-line payment scheme with a single denomination of coins. An extension to multiple denominations is straight-forward. Extensions to off-line payment schemes are discussed in Section 6.5.

Let us explain the underlying ideas of our scheme. The main components of a coin are a pair (h_p, z_p) satisfying $z_p = h_p^x$, where x is the bank's secret key, a signature (denoted W) based on a proof of this fact, and a further signature (denoted V) needed to guarantee that anonymity revocation is possible. The signature W is given by the bank and the signature V can be computed by the customer on his own. To achieve anonymity, the signature W must be issued using a blind signature scheme: during withdrawal the customer sends the bank a randomized (or blinded) pair (h_w, z_w), the bank computes a signature corresponding to W based on this pair, and the customer transforms this signature into the signature W. This is achieved by a subprotocol of the withdrawal protocol and is explained in Section 6.4.2. The anonymity of a coin can be revoked by the trustee if he can link pairs (h_p, z_p) and (h_w, z_w). This is guaranteed by a mechanism explained in Section 6.4.5.

6.4.1 System Setup

To set up the payment system the bank chooses a finite group G of prime order q such that computing discrete logarithms in G is infeasible. Such a group is cyclic and thus every element (except the neutral element) is a generator. Today the choice $q \approx 2^{170}$ appears to be secure unless the group has a special structure. Three elements g, g_1 and g_2

are chosen by a publicly verifiable pseudo-random mechanism to assure that the discrete logarithms of none of these elements with respect to one another are known. Finally, the bank chooses a secret key $x \in_R \mathbb{Z}_q$ and computes the public key $y{:=}g^x$. The bank publishes G, g, g_1, g_2, and y.

The trustee randomly chooses his secret key $\tau \in \mathbb{Z}_q^*$ and computes and publishes the corresponding public key $y_T{:=}g_2^{\tau}$.

6.4.2 A Subprotocol: A Modified Blind Schnorr Signature Scheme

As mentioned before, the pair (h_p, z_p) is obtained blindly by the customer in a subprotocol, referred to as protocol **P** (see Figure 6.1), during which the bank sees only the pair (h_w, z_w). This protocol is an extension of the blind issuing protocol for Schnorr signatures originated by Brands [Bra93]. It is discussed here as an independent protocol (with its own players, input and output parameters) because it is of independent interest and because it will be reused later.

Protocol **P** takes place between two players A and B, substituted in the withdrawal protocol by the customer and the bank, respectively. A's input consists of the pair (g, y), where $y = g^x$ is B's public key, a secret message m, a group element h_w also known to B, and a blinding exponent a. If both players are honest, A's output of the protocol consists of the pair (h_p, z_p) and the signature

$$W = (c, s) = SPK_6\{(\alpha) : y = g^{\alpha} \wedge z_p = h_p^{\alpha}\}(m)$$

which serves two purposes. On one hand, it is a (blind) signature by player B of the message m. On the other hand, W also proves that the pair (h_p, z_p) satisfies $z_p = h_p^x$, where $x(= \log_g y)$ is player B's secret key.

From B's point of view, the protocol **P** is the interactive protocol corresponding to the SPK

$$W' = SPK_6\{(\alpha) : y = g^{\alpha} \wedge z_w = h_w^{\alpha}\}(m)$$

and is thus honest-verifier zero-knowledge. This protocol is turned into a blind signature scheme by Player A by randomizing $z_w, \tilde{t}_g, \tilde{t}_h, \tilde{c}, \tilde{s}$ with γ, δ and a. The exponent a that transforms (h_w, z_w) into (h_p, z_p), i.e.,

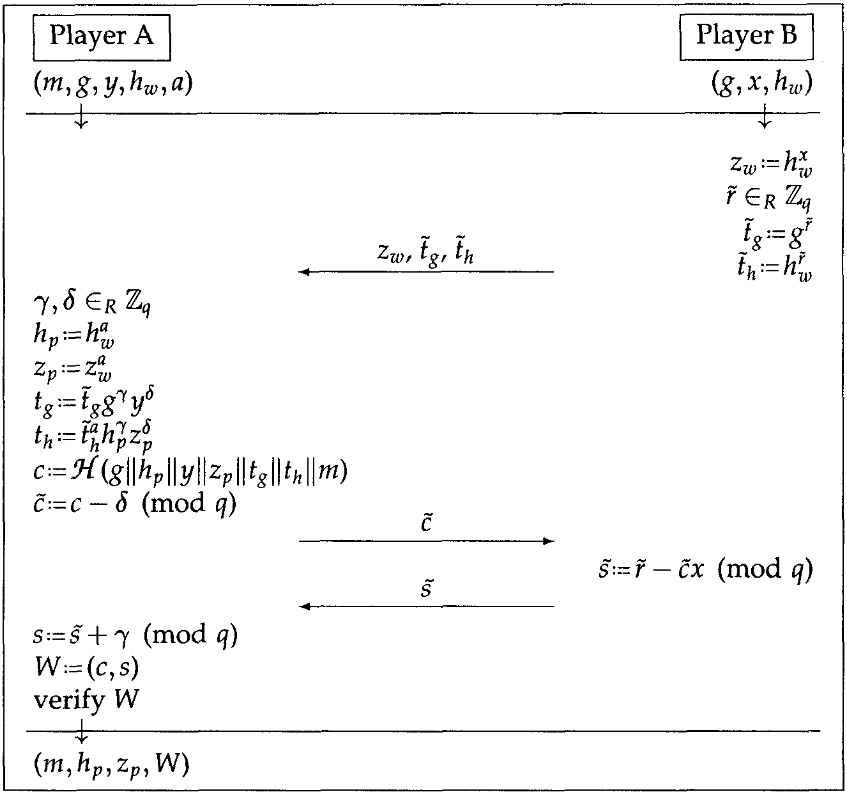
Player A
Player B
(m,g,y,h_w,a)
(g,x,h_w)
$z_w := h_w^x$
$\tilde{r} \in_R \mathbb{Z}_q$
$\tilde{t}_g := g^{\tilde{r}}$
$\tilde{t}_h := h_w^{\tilde{r}}$
$z_w, \tilde{t}_g, \tilde{t}_h$
$\gamma, \delta \in_R \mathbb{Z}_q$
$h_p := h_w^a$
$z_p := z_w^a$
$t_g := \tilde{t}_g g^\gamma y^\delta$
$t_h := \tilde{t}_h^a h_p^\gamma z_p^\delta$
$c := \mathcal{H}(g\|h_p\|y\|z_p\|t_g\|t_h\|m)$
$\tilde{c} := c - \delta \pmod q$
$\tilde{c}$
$\tilde{s} := \tilde{r} - \tilde{c}x \pmod q$
$\tilde{s}$
$s := \tilde{s} + \gamma \pmod q$
$W := (c,s)$
verify W
(m,h_p,z_p,W)

Figure 6.1: The protocol **P**. It is in a blind signature scheme that allows the customer to obtain $W = (c,s) = SPK_6\{(\alpha) : y = g^\alpha \wedge z_p = h_p^\alpha\}(m)$. It is also the core of the withdrawal-protocol.

randomizes (h_p, z_p), is chosen by player A before engaging in protocol **P**.

The proof that the protocol **P** is a blind signature scheme, i.e., that B's view $\langle B(x), A(m,a,y)\rangle(g,h_w)$ is statistically independent of A's output $[A(m,a,y), B(x)](g,h_w)$ is similar to the proof of blindness for the blind Schnorr signature protocol (cf. Section 2.8.2). Given any B's view consisting of $(z_w, h_w, \tilde{t}_g, \tilde{t}_h, \tilde{r}, \tilde{c}, \tilde{s})$, and any signature $(c,s) = SPK_6\{(\alpha) : y = g^\alpha \wedge z_p = h_p^\alpha\}(m)$ of a message m, let

$$\begin{array}{rcll} \gamma & := & s - \tilde{s} & \pmod q, \\ \delta & := & c - \tilde{c} & \pmod q, \end{array}$$

$$\begin{array}{lcl} a & := & \log_{h_w} h_p \equiv \log_{z_w} z_p \pmod{q}, \\ t_g^* & := & \tilde{t}_g g^{\gamma} y^{\delta}, \quad \text{and} \\ t_h^* & := & \tilde{t}_h^a h_p^{\gamma} z_p^{\delta}. \end{array}$$

It remains to show that $t_g^* = t_g = g^s y^c$ and $t_h^* = t_h = h_p^s z_p^c$ is satisfied:

$$t_g^* = \tilde{t}_g g^{\gamma} y^{\delta} = g^{\tilde{r}+\gamma+x\delta} = g^{\tilde{r}+s-\tilde{s}+x(c-\tilde{c})} = g^{s+xc} g^{\tilde{r}-\tilde{s}-x\tilde{c}} = g^s y^c = t$$

and

$$t_h^* = \tilde{t}_h^a h_p^{\gamma} z_p^{\delta} = h_w^{\tilde{r}a+\gamma a+x\delta a} = h_p^{\tilde{r}+s-\tilde{s}+x(c-\tilde{c})} = h_p^s z_p^c = t_h.$$

For both t_g and t_h the last two equalities hold because $\tilde{s} \equiv \tilde{r} - x\tilde{c} \pmod{q}$ and because (c, s) is a valid signature.

6.4.3 The Withdrawal Protocol

The actual withdrawal protocol, which uses protocol **P** as a subprotocol, is shown in Figure 6.2. It is based on a fair blind signature scheme devised by Stadler [Sta96a]. The customer chooses a random exponent a which plays two different roles in the protocol. On one hand, it serves as the blinding exponent (within protocol **P**) to transform the pair (h_w, z_w) into the pair (h_p, z_p). On the other hand, it is used to compute h_w as $h_w := g_1^{a^{-1}} g_2$ and d as $d := y_T^a$. The SPK denoted as U proves that the two exponents used in the computation of h_w and d are indeed the same. The value d can be interpreted as a Diffie-Hellman-type encryption of h_p for the trustee and is stored by the bank for possible later anonymity revocation.

For the computation of U we use the fact that by exchanging base and input element of a discrete logarithm computation, the resulting discrete logarithm is inverted modulo the group order:

$$\log_{g_1} h_w/g_2 \equiv (\log_{(h_w/g_2)} g_1)^{-1} \pmod{q}.$$

The correctness of the withdrawal protocol follows from the correctness of protocol **P**. The coin consists of the coin number *c#*, the values h_p, z_p, W, and a signature

$$V := SPK_1\left\{(\alpha) : \frac{h_p}{g_1} = g_2^{\alpha}\right\}(W)$$

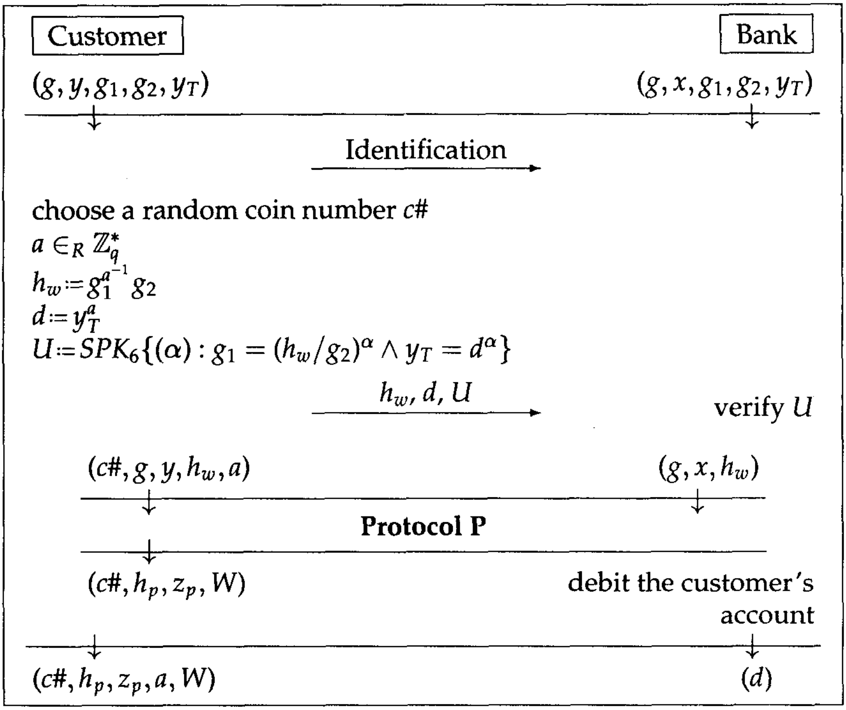

Figure 6.2: The withdrawal protocol in the on-line scheme. It uses the protocol **P** as subprotocol.

that the customer computes before spending the coin. The pair V is a signature based on a proof that h_p equals $g_1 g_2^a$ for some a known to the customer. This prevents the potential attack that in protocol **P**, seen as an independent protocol, player A could successfully choose h_p as $h_p := h_w^a g^b$ and z_p as $z_p := z_w^a y^b$ for some $b \neq 0$. Such an attack would allow a cheating customer to avoid later anonymity revocation. However, the customer can generate the signature V only if he chooses $b = 0$ in the described attack, which is therefore not successful.

6.4.4 The On-line Payment Protocol

A coin can be spent by sending it to a shop which verifies the coin and, if it is valid, passes the coin on to the bank. The bank checks the database of all previously spent coins. (By including an expiration date in the coin one can limit the size of this database.) If the coin is new it is accepted and entered into the database, and the shop's account is credited. The protocol is shown in Figure 6.3.

6.4.5 Anonymity Revocation

As already mentioned, there are two kinds of anonymity revocation, namely withdrawal-based and payment-based revocation. The latter can be achieved by letting the trustee compute the value

$$(h_p/g_1)^{\tau} = (g_2^a)^{\tau} = d$$

from an h_p that is observed in a payment. Furthermore, the trustee computes

$$S_p := SPK_6\left\{(\alpha) : \; d = \left(\frac{h_p}{g_1}\right)^{\alpha} \wedge \; y_T = g_2^{\alpha}\right\}$$

to assure that she computed d correctly. The value d can then be searched for in the bank's revocation database containing the transcripts of the withdrawal transactions, including the values d.

Withdrawal-based anonymity revocation is achieved as follows. Given the value d observed in a withdrawal transaction, the trustee computes

$$g_1 d^{(\tau^{-1})} = g_1 g_2^a = h_p$$

and the signature

$$S_w := SPK_6\left\{(\alpha) : \frac{h_p}{g_1} = d^{\alpha} \wedge g_2 = y_T^{\alpha}\right\}(h_p)$$

to assure correctness. The value h_p can be put on a black list for recognizing the coin later when it is spent. The two types of anonymity revocation are possible because

$$a \equiv \left(\log_{g_1}\left(\frac{h_w}{g_2}\right)\right)^{-1} \equiv \log_{y_T} d \equiv \log_{g_2}\left(\frac{h_p}{g_1}\right) \pmod{q}$$

holds for every pair h_w and h_p generated during a legitimate withdrawal transaction.

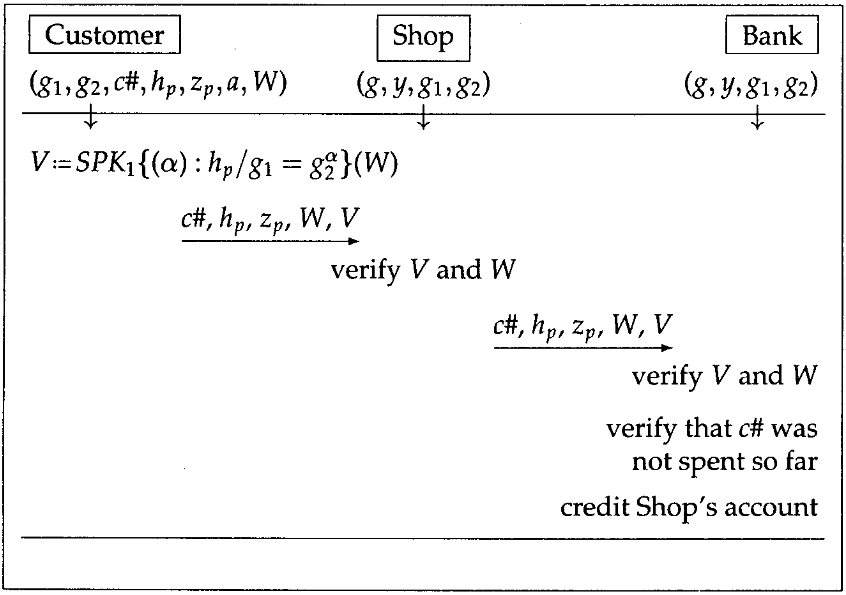

Figure 6.3: The payment protocol in the on-line scheme. The customer, the shop, and the bank are all on-line. It is assumed that if any verification fails the protocol is stopped and all three parties are informed.

6.4.6 Security Properties

We summarize how our on-line payment scheme achieves the required security properties.

Payments are unlinkable and anonymous: Withdrawal and payment are unlinkable because of the blindness of the subprotocol **P**. However, although the blindness of protocol **P** is unconditional, i.e., information-theoretical, the anonymity of the payment scheme is only computational because of the revocation parameter d. The bank could link withdrawal and payment by testing whether

$$\log_{y_T} d \equiv \log_{g_2}(h_p/g_1) \pmod{q},$$

holds. However, since the bank does not know $\log_{g_2} y_T$, this is computationally infeasible. In particular, if the bank could decide this, it could also solve the Decision-Diffie-Hellman problem.

Coins cannot be forged: In the random oracle model, forging coins without the help of the bank is as hard as computing discrete logarithms. Although it seems impossible that a customer cannot get more than k valid coins from k interactions with the bank in a withdrawal protocol, this cannot be proven to be as hard as computing discrete logarithms. This remains an open problem.

Double-spending is prevented: Since the payment is made on-line, the bank can easily detect whether a coin has already been spent by checking its database. Thus double-spending is not possible. Note that this implies that the customer can not be falsely accused of double-spending.

The anonymity can be revealed: Withdrawal-based and payment-based revocation of the anonymity is discussed in paragraph 6.4.5.

It remains to argue that a customer cannot circumvent the revocation. A customer can circumvent revocation if he succeeds in getting a signature $W = SPK_6\{(\alpha) : y = g^{\alpha} \wedge z'_p = {h'}_p^{\alpha}\}(m)$ for a $h'_p = g_1 g_2^{a'}$ the corresponding d does not appear in the bank's database. In the withdrawal protocol, the customer receives $\tilde{t}_g = g^{\tilde{r}}$ and $\tilde{t}_h = (g_1^{a^{-1}} g_2)^{\tilde{r}}$. From these two group elements the customer must compute the value $t'_h = (g_1 g_2^{a'})^{\tilde{r}}$ for including it into to argument to the hash function. If the customer had an efficient algorithm to do this, he could also solve the Diffie-Hellman problem (DHP) efficiently. We therefore conclude that this attack is not feasible.

Let us briefly describe how the customer could solve the DHP given such an algorithm. Let the algorithm output g_1^r on input g_1, g_2, a, and $\tilde{t}_h$, where $r \equiv \log_{g_1^{1/a} g_2} \tilde{t}_h \pmod{q}$. (The group element g_1^r allows the customer to compute the desired t'_h.) Let $\dot{g}$ be the base of the DHP and $\dot{y}_1$ and $\dot{y}_2$ the two public keys. For solving the DHP, the customer has to compute the element $\dot{z}$ such that

$$\log_{\dot{g}} \dot{z} \equiv \log_{\dot{g}} \dot{y}_1 \log_{\dot{g}} \dot{y}_2 \pmod{q}$$

holds. This he can do using the above algorithm with the inputs $g_1 = \dot{y}_1$, $g_2 = \dot{g}\dot{y}_1^{-1/a}$, a, and $\tilde{t}_h = \dot{y}_2$, where a is randomly chosen from $\mathbb{Z}_q^*$. The output of the algorithm will be the element $\dot{z}$ as can easily be seen.

6.4.7 Efficiency Considerations

A coin in the proposed scheme consists of two group elements, two hash values, and two numbers smaller than q. If the group G allows for a compact representation of its elements, the signatures can be quite short. For instance, elements of an elliptic curve with order q over a field of cardinality close to q can be represented by two field elements. Hence for $q \approx 2^{170}$, the total signature length is roughly $6 \log_2 q + 256 \approx 1300$ bits. This could be reduced further to about 1000 bits by a compressed representation of group-elements and by using the same challenge for the proofs V and W.

6.5 Extensions to Off-line Payments

In an off-line system, double-spending can only be detected after the fact, but it cannot be prevented. Detecting double-spending is trivial, but identifying the cheating customer requires an additional mechanism in the protocols. In the presented system, a natural solution appears to be to involve the trustee for exposing double-spenders. This solution is unsatisfactory from an operational point of view when many instances of double-spending occur. We describe a modified payment protocol in Section 6.5.1 that allows the bank to identify double-spenders without the trustee's help.

In Section 6.5.2 we discuss the use of tamper-resistant hardware (called observer in this context) for preventing double-spending in off-line systems. As of today there exist no absolutely tamper-proof components, so such devices must be combined with the techniques for identifying double-spenders discussed before. In contrast to off-line systems without observers, it is acceptable to involve the trustee for identifying cheaters because the breakdown of an observer can be considered a rare event. However, it is also possible to use the technique of Section 6.5.1 in an observer-based system.

6.5.1 Enabling the Bank to Identify Cheaters

The payment protocol of Figure 6.4 allows the bank to identify double-spenders, but the anonymity of honest customers is not compromised.

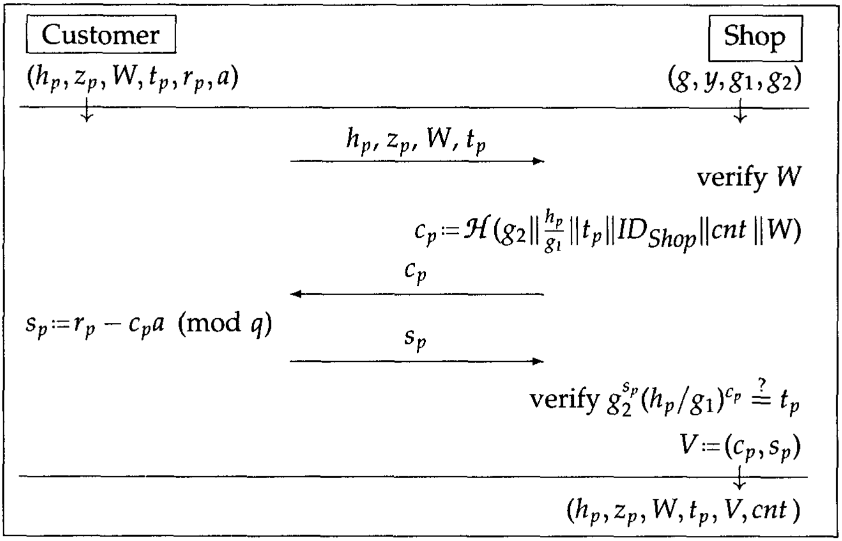
Customer
Shop
$(h_p, z_p, W, t_p, r_p, a)$
(g, y, g_1, g_2)
h_p, z_p, W, t_p
verify W
$c_p := \mathcal{H}(g_2 \| \frac{h_p}{g_1} \| t_p \| ID_{Shop} \| cnt \| W)$
c_p
$s_p := r_p - c_p a \pmod{q}$
s_p
verify $g_2^{s_p}(h_p/g_1)^{c_p} \stackrel{?}{=} t_p$
$V := (c_p, s_p)$
$(h_p, z_p, W, t_p, V, cnt)$

Figure 6.4: The payment protocol in the off-line scheme. The signature $V = SPK_1\{(\alpha) : h_p/g_1 = g_2^{\alpha}\}(ID_{Shop} \| cnt \| W)$ is now jointly computed by the shop and the customer.

The basic idea is to redefine the signature V and to make use of the following fact that was already used in a similar way in [Bra93] for the purpose of identifying double-spenders. If the value $t = g^r$ generated by the signer for issuing a Schnorr signature (or in a message-dependent SPK) is used for signing more than one message, it is easy to compute the secret key from the two signatures. Let (c_1, s_1) and (c_2, s_2) denote the two distinct signatures and let $y = g^x$. If

$$g^{s_1} y^{c_1} = t = g^{s_2} y^{c_2}$$

then we have $s_1 + c_1 x \equiv s_2 + c_2 x \pmod{q}$ and hence

$$x \equiv \frac{s_1 - s_2}{c_2 - c_1} \pmod{q}.$$

This fact can be used when forcing the customer to use the same value $t_p = g_2^{r_p}$ in every potential payment of a given coin. This is achieved by having the customer choose and store r_p during withdrawal and by including t_p instead of the coin number $c\#$ in the signature W. Furthermore, the argument to the hash function in V contains informa-

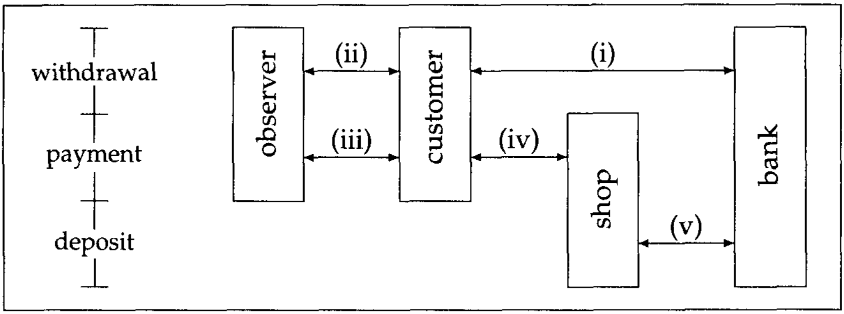

Figure 6.5: The customer needs the help of the observer for carrying out withdrawal and payment transactions. For each coin the observer only participates once in a payment transaction; hence multiple spending of a coin can be prevented.

tion about the shop and the withdrawal transaction, where the latter is achieved by including W which contains a hash-value of t_p, h_p, and z_p. The counter *cnt* is included in the hash value because otherwise the shop could deposit a coin twice and, in reply to bank's objection, blame the customer of having it spent twice (at the same shop).

When a coin is spent more than once, the bank can compute the value a that served as the customer's secret key in the payment protocol. Then the bank can identify the customer by computing $d = y_T^a$.

6.5.2 Observers Can Prevent Double-spending

It is quite unsatisfactory that in anonymous off-line payment schemes the multiple spending of coins can only be detected but not prevented. Chaum et al. [Cha92] proposed as a solution the use of so-called *observers*, which are small tamper-resistant hardware devices that are issued by the bank to every customer. Transactions can only be carried out in cooperation with the observer (see Figure 6.5). In particular, the observer keeps a list of active (withdrawn but not yet spent) coins and refuses to cooperate in spending a coin a second time, i.e., it cooperates only in spending coins contained in its list of active coins.

The following requirements guarantee the customers' privacy (for a more detailed discussion see [CP94]):

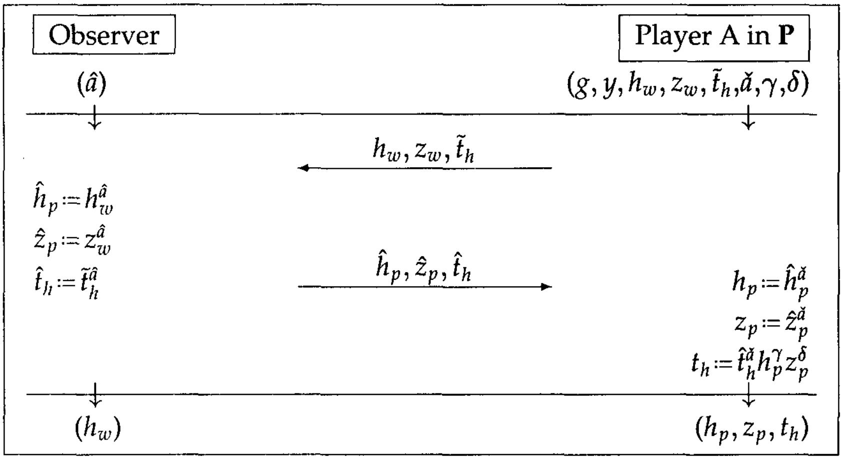
Observer
Player A in P
$(\hat{a})$
$(g, y, h_w, z_w, \tilde{t}_h, \check{a}, \gamma, \delta)$
$h_w, z_w, \tilde{t}_h$
$\hat{h}_p := h_w^{\hat{a}}$
$\hat{z}_p := z_w^{\hat{a}}$
$\hat{t}_h := \tilde{t}_h^{\hat{a}}$
$\hat{h}_p, \hat{z}_p, \hat{t}_h$
$h_p := \hat{h}_p^{\check{a}}$
$z_p := \hat{z}_p^{\check{a}}$
$t_h := \hat{t}_h^{\check{a}} h_p^{\gamma} z_p^{\delta}$
(h_w)
(h_p, z_p, t_h)

Figure 6.6: In the observer-based system, the computation of h_p, z_p, and t_h within protocol **P** must be performed jointly by customer and observer. The modified protocol **P** is referred to as protocol $\mathbf{P}_o$.

- The observer must not be able to communicate with anyone except the customer. In particular, the observer must not be able to establish a subliminal channel to the bank.

- Even if the bank can get hold of an observer and read all data stored in it, the bank must be unable to trace payments. Note that this implies that the observer must not have an internal clock.

In Figure 6.5 the communication between observer, customer, bank and shop is illustrated. Because each customer's observer is unique, the bank could link the communications (ii) and (iii) with (i). In order to satisfy the conditions stated above, the communication (iv) must be unlinkable with communications (i) to (iii).

We now describe how the on-line payment scheme presented in Section 6.4 can be turned into an observer-based off-line scheme. Let ω and $y_o := g_1^{\omega}$ denote the observer's secret key and public key, respectively. (Note that each observer has its own secret key/public key pair.) The basic idea is that the customer and the observer share the value a such that neither of them alone knows a. More precisely, a is replaced by the product $\hat{a}\check{a}$ modulo q, where $\hat{a}$ is chosen (and kept secret) by the observer, and $\check{a}$ is chosen by the customer. All operations involving a

in the on-line protocol now require the observer's cooperation. Hence the observer is able to prevent double-spending. Furthermore, the customer must prove to the bank during the withdrawal of a coin that the value a is indeed shared with the observer.

Figure 6.6 shows the modifications in the subprotocol **P**. The parameter a occurs in the computation of h_p, z_p, and t_h. The observer obtains only values (h_w, z_w, and $\tilde{t}_h$) which the bank already knows; hence no relevant information is leaked to the observer. The resulting subprotocol is called $\mathbf{P}_o$ in which the parameter $\breve{a}$ replaces player A's input a in protocol **P**.

Figure 6.7 describes the withdrawal protocol. After the identification, the customer and the observer jointly compute the values h_w and d. Then they jointly construct the two SPK's

$$U_1 := SPK_1\left\{(\alpha) : \frac{h_w}{g_2} = y_o^\alpha\right\}$$

and

$$U_2 := SPK_6\left\{(\alpha) : \frac{h_w}{g_2} = g_1^\alpha \wedge y_T = d^\alpha\right\}.$$

The protocol that the customer and the observer carry out to compute the signatures U_1 and U_2 is the interactive protocol corresponding to the two signatures

$$U_1' := SPK_1\{(\alpha) : \hat{h}_w = y_o^\alpha\}$$

and

$$U_2' := SPK_6\{(\alpha) : \hat{h}_w = g_1^\alpha \wedge y_T = \hat{d}^\alpha\}$$

and is honest-verifier zero-knowledge. Using this protocol the customer can compute the signatures U_1 and U_2. The signature U_2 is identical to the signature U in the on-line scheme and convinces the bank that d is formed correctly. The signature U_1 convinces the bank that the observer is indeed engaged in the protocol, because the knowledge of both $\log_{(h_w/g_2)} y_o$ (because of U_1) and $\log_{g_1} h_w/g_2$ (because of U_2) implies the knowledge of $\log_{g_1} y_o$ ($= \omega$), the observer's secret key. Since the bank knows that the observer does not deviate from the protocol it knows that the customer alone cannot know the value $\log_{h_w/g_2} g_1$.

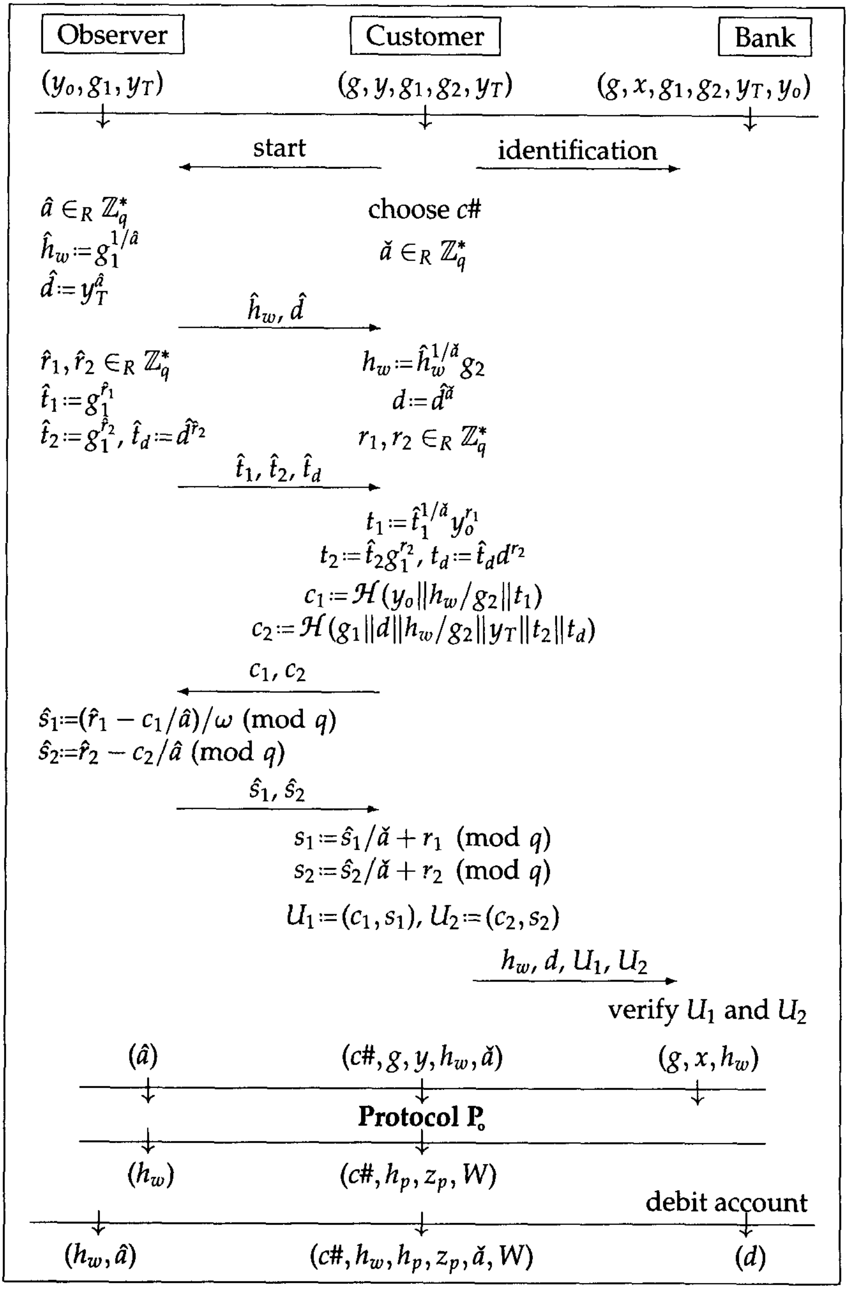
Observer
Customer
Bank
(y_o, g_1, y_T)
(g, y, g_1, g_2, y_T)
$(g, x, g_1, g_2, y_T, y_o)$
start
identification
$\hat{a} \in_R \mathbb{Z}_q^*$
$\hat{h}_w := g_1^{1/\hat{a}}$
$\hat{d} := y_T^{\hat{a}}$
choose $c\#$
$\check{a} \in_R \mathbb{Z}_q^*$
$\hat{h}_w, \hat{d}$
$\hat{r}_1, \hat{r}_2 \in_R \mathbb{Z}_q^*$
$\hat{t}_1 := g_1^{\hat{r}_1}$
$\hat{t}_2 := g_1^{\hat{r}_2}, \hat{t}_d := \hat{d}^{\hat{r}_2}$
$h_w := \hat{h}_w^{1/\check{a}} g_2$
$d := \hat{d}^{\check{a}}$
$r_1, r_2 \in_R \mathbb{Z}_q^*$
$\hat{t}_1, \hat{t}_2, \hat{t}_d$
$t_1 := \hat{t}_1^{1/\check{a}} y_o^{r_1}$
$t_2 := \hat{t}_2 g_1^{r_2}, t_d := \hat{t}_d d^{r_2}$
$c_1 := \mathcal{H}(y_o \| h_w/g_2 \| t_1)$
$c_2 := \mathcal{H}(g_1 \| d \| h_w/g_2 \| y_T \| t_2 \| t_d)$
c_1, c_2
$\hat{s}_1 := (\hat{r}_1 - c_1/\hat{a})/\omega \pmod q$
$\hat{s}_2 := \hat{r}_2 - c_2/\hat{a} \pmod q$
$\hat{s}_1, \hat{s}_2$
$s_1 := \hat{s}_1/\check{a} + r_1 \pmod q$
$s_2 := \hat{s}_2/\check{a} + r_2 \pmod q$
$U_1 := (c_1, s_1), U_2 := (c_2, s_2)$
h_w, d, U_1, U_2
verify U_1 and U_2
$(\hat{a})$
$(c\#, g, y, h_w, \check{a})$
(g, x, h_w)
Protocol P_o
(h_w)
$(c\#, h_p, z_p, W)$
debit account
$(h_w, \hat{a})$
$(c\#, h_w, h_p, z_p, \check{a}, W)$
(d)

Figure 6.7: The withdrawal protocol in the off-line system with observer.

The following equation shows that the signatures U_1 and U_2 computed in the withdrawal protocol of Figure 6.7 are indeed valid. We have

$$\begin{aligned} y_o^{s_1}\left(\frac{h_w}{g_2}\right)^{c_1} &= y_o^{\hat{s}_1/\breve{a}+r_1}\hat{h}_w^{c_1/\breve{a}} = y_o^{(\hat{r}_1-c_1/\hat{a})/(\omega\breve{a})+r_1}\hat{h}_w^{c_1/\breve{a}} = \\ &= y_o^{r_1}g_1^{(\hat{r}_1-c_1/\hat{a})/\breve{a}}\hat{h}_w^{c_1/\breve{a}} = y_o^{r_1}g_1^{\hat{r}_1/\breve{a}-c_1/(\hat{a}\breve{a})}g_1^{c_1/(\breve{a}\hat{a})} = \\ &= y_o^{r_1}g_1^{\hat{r}_1/\breve{a}} = y_o^{r_1}\hat{t}_1^{1/\breve{a}} = t_1 \end{aligned}$$

and hence the verification equation for U_1 holds. For U_2 we show only that $d^{s_2}y_T^{c_2} = t_d$ is true. Showing that $g_1^{s_2}(h_w/g_2)^{c_2} = t_2$ holds is analogous. We have

$$\begin{aligned} d^{s_2}y_T^{c_2} &= d^{\hat{s}_2/\breve{a}+r_2}y_T^{c_2} = d^{r_2}d^{(\hat{r}_2-c_2/\hat{a})/\breve{a}}y_T^{c_2} = \\ &= d^{r_2}\hat{d}^{(\hat{r}_2-c_2/\hat{a})}y_T^{c_2} = d^{r_2}\hat{d}^{\hat{r}_2}y_T^{-c_2}y_T^{c_2} = \\ &= d^{r_2}\hat{t}_d = t_d \end{aligned}$$

and thus also the verification equation of U_2 holds.

An important point in the construction of these two signatures is the additional randomization (blinding) performed by the customer using the random values r_1 and r_2. This prevents the bank from computing a using s_1 and $\hat{s}_1$ (or s_2 and $\hat{s}_2$) in case the bank learns the values $\hat{s}_1$ and $\hat{s}_2$. This could happen for instance when the observer is returned to the bank or when the bank knows the seed of the pseudo-random number generator used by the observer.

At the end of the withdrawal protocol the observer and the customer store h_w for the purpose of identifying the coin later in the payment protocol (shown in Figure 6.8).

The payment protocol is very similar to the protocol in Figure 6.4, except that the observer and the customer jointly compute the signature V. It is essential that the communication between the customer and the observer is unlinkable with the communication between the shop and the customer, so that neither the bank nor the observer obtains useful information about the correspondence of withdrawal and payment transactions. This is achieved by randomizing (blinding) the values t_p, c_p, and s_p. The protocol between the customer and the observer is the interactive protocol corresponding to the SPK

$$V' := SPK_1\left\{(\alpha) : \left(\frac{h_p}{g_1}\right)^{1/\breve{a}} = g_2^{\alpha}\right\}.$$

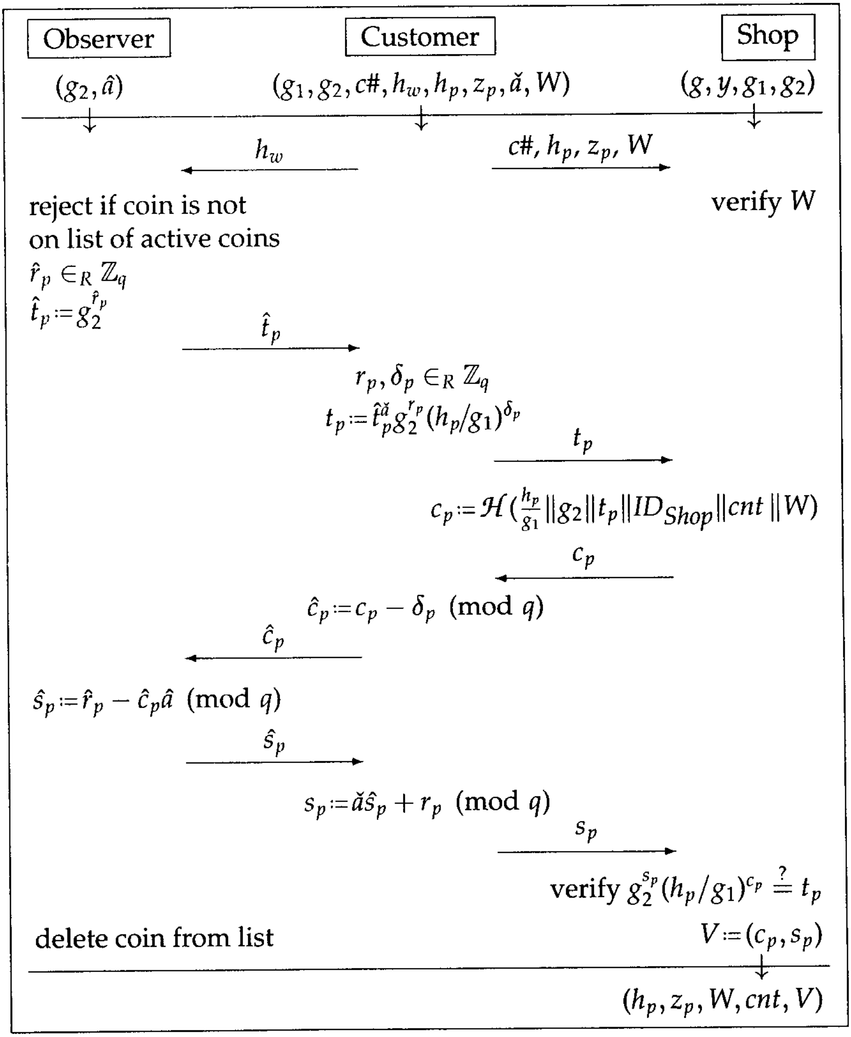
Observer
Customer
Shop
$(g_2, \hat{a})$
$(g_1, g_2, c\#, h_w, h_p, z_p, \check{a}, W)$
(g, y, g_1, g_2)
h_w
$c\#, h_p, z_p, W$
reject if coin is not on list of active coins
verify W
$\hat{r}_p \in_R \mathbb{Z}_q$
$\hat{t}_p := g_2^{\hat{r}_p}$
$\hat{t}_p$
$r_p, \delta_p \in_R \mathbb{Z}_q$
$t_p := \hat{t}_p^{\check{a}} g_2^{r_p} (h_p/g_1)^{\delta_p}$
t_p
$c_p := \mathcal{H}(\frac{h_p}{g_1} \| g_2 \| t_p \| ID_{Shop} \| cnt \| W)$
c_p
$\hat{c}_p := c_p - \delta_p \pmod q$
$\hat{c}_p$
$\hat{s}_p := \hat{r}_p - \hat{c}_p \hat{a} \pmod q$
$\hat{s}_p$
$s_p := \check{a} \hat{s}_p + r_p \pmod q$
s_p
verify $g_2^{s_p} (h_p/g_1)^{c_p} \overset{?}{=} t_p$
delete coin from list
$V := (c_p, s_p)$
(h_p, z_p, W, cnt, V)

Figure 6.8: The payment protocol in the observer-based off-line system. At the end of the protocol the shop possesses the signature $V = SPK_1\{(\alpha) : h_p/g_1 = g_2^{\alpha}\}(ID_{Shop} \| cnt \| W)$

and is therefore also honest-verifier zero-knowledge. Using this protocol the customer can compute the signature V. Since the observer will carry out this protocol only once for a given h_w and therefore only once for a given $\hat{a}$, the customer cannot double-spend a coin.

The correctness of the protocol in Figure 6.8 can be seen as follows:

$$\begin{aligned} g_2^{s_p}\left(\frac{h_p}{g_1}\right)^{c_p} &= g_2^{(\check{a}\hat{r}_p-\hat{c}_p\hat{a}\check{a}+r_p)}\left(\frac{h_p}{g_1}\right)^{c_p} = \\ &= \hat{t}_p^{\check{a}}g_2^{(r_p+\delta_p\hat{a}\check{a})} = \hat{t}_p^{\check{a}}g_2^{r_p}\left(\frac{h_p}{g_1}\right)^{\delta_p} = \\ &= t_p \end{aligned}$$

and therefore the verification equation of the signature V holds. Here we used the fact that $h_p = g_1 g_2^{\hat{a}\check{a}}$.

6.5.3 Security properties

In this paragraph we discuss the security properties of both the observer-based and the non-observer-based off-line schemes.

Payments are unlinkable and anonymous: This holds for the same reasons as for the on-line system.

Coins cannot be forged: For both off-line system we have the same situation as for the on-line system.

Double-spending is detected: In both off-line systems the customer and the shop jointly compute the signature V of a message that includes W. Different payments yield different such signatures. Thus the bank can detect a double spending it finds different signatures V of messages containing the same W. In the non-observer-based system the bank can identify a double-spender as described in Section 6.5.1.

In the observer-based system double-spending is not possbile because of the observer. However, if a malicious customer succeeded to break the observer and thus to double-spend, the bank can still detect it. To identify a double-spender the bank requires the help of the trustee. Since breaking the observer is assumed to be hard, this can be considered to be a rare event.

The anonymity can be revealed: This is true for same reasons as for the on-line system.

The customer can not be falsely accused of double-spending: To accuse a customer of double sending, the bank must show two different signatures V. The ability of the bank to do this when the customer spent a coin only once, implies the ability of the bank to forge such signatures which is assumed to be infeasible.

The shop is assured of the validity of coins: This is by definition of the validity of coins, i.e., if the sigatures W and V are valid, the bank must accept the coin.

Finally, for the observer-based system it remains to show that the observer (even when the bank gets hold on it later) cannot infringe the customer's privacy. This is argued in Section 6.5.2.

6.6 Sharing the Revocation Capability Among Several Trustees

To achieve higher security against fraudulent anonymity revocation, the revocation capability can be shared among several trustees such that only predefined subsets of the trustees are able to cooperatively revoke a customer's anonymity. In Chapter 7 it is shown how the revocation capability can be shared using a secret sharing scheme with this access structure and τ as the shared secret. Furthermore, procedures are discussed that enable the trustees to jointly revoke the customers' anonymity.

6.7 Comparison with Other Schemes with Passive Trustees

In this section we compare our on-line scheme with the cut-and-choose based approaches and the recent proposal of [FTY96].

We sketch the scheme of [SPC95] (which from an conceptual and efficiency point of view is similar to the scheme of [BGK95]). In order to obtain a blind signature of a message m, the customer prepares $2K$ blinded messages, each of which contains m encrypted with the trustee's public key as well as a session identifier encrypted with the trustee's public key. K is a security parameter. These encryptions are probabilistic (i.e., the text is padded with a random string before encryption) in order to prevent decryption by an exhaustive search over a small set of possible values. To check that these messages are properly formed, the bank chooses a random subset of K blinded messages and asks the customer to open all of them, where "open" means presenting the random padding used for encrypting the session identifier. For the purpose of possible later anonymity revocation, the bank stores the corresponding K encryptions of m. Then it blindly signs the remaining K messages that were not opened. Such a coin (a blind signature for the message m) is valid if the bank's signature is valid and if it can be verified that m was correctly encrypted for the trustee.

In such a system, withdrawal-based revocation can be achieved by asking the trustee to open the encryptions of m which the bank obtained and stored during the withdrawal protocol. Payment-based anonymity revocation can be achieved by asking the trustee to decrypt the encrypted session ID contained in each of the K components of the signature. The probability that a dishonest customer manages to escape payment-based or withdrawal-based anonymity revocation is $1/\binom{2K}{K} \approx 2^{-2K}\sqrt{\pi K}$. To achieve reasonable security, K should be at least 20. Each of the K components consists of a random padding string and a public-key encrypted value. In order to achieve the same security level as in our scheme, the lengths of these two values must be at least 64 and 768 bits, respectively. This results in a total signature length of close to $17,000$ bits, which is about 13 times longer than coins in our scheme (see Section 6.4.7).

The scheme presented in [FTY96] is based on Brands' payment system [Bra93] and on so-called "indirect discourse proofs." Two such proofs are used to convince the bank and, independently, the shop that the trustee can revoke the anonymity of a coin. Technically, an indirect discourse proof consists of an ElGamal encryption [ElG85a] of either the customer's identity (for payment-based revocation) or a unique part of the coin (for withdrawal-based revocation), and a proof that the correct value is encrypted. Conceptually, this technique is similar to that described in this chapter, but the system of [FTY96] is less efficient. For instance, coins in [FTY96] are approximately twice as long.

Chapter 7

Sharing and Diverting the Capability of Anonymity Revocation

The schemes presented in the previous three chapters have in common that a trusted third party is able to revoke the anonymity of other parties. In the group signature schemes, it is the group (or revocation) manager who can find out which group member signed a message. In the payment systems, it is the trustee who can reveal a payer's identity. In both scenarios, customers/group members have to trust this third party not to reveal their identity at will. This risk can, on one hand, be reduced by sharing this revocation-ability among several third parties such that only designated subsets of them can *jointly* reveal identities. On the other hand, the risk of fraudulent anonymity-revocation can also be weakened if a customer/group member can choose a third party he trusts such that only "his" third party is able to reveal his identity. Of course, it is possible to combine both approaches. This chapter describes realizations of them.

Subsequently, by customers and trustees we always also mean group members and revocation managers, respectively.

7.1 Provable Encryption

The way to share and/or divert the capability of anonymity revocation depends strongly on how the original single trustee obtains this capability and what operations she must perform for revocation. Therefore, we review how our schemes achieves the trustee's ability to revoke the anonymity of customers (or group members).

Since the trustee should be passive in all our schemes, the customer has to provide information that is encrypted for the trustee and which allows her to (later) reveal his identity by a simple decryption. Furthermore, the customer must provide a proof that he indeed encrypted this information[1]. In the group signature schemes, this is done exactly as we describe it here. In the payment schemes, however, a slightly different method is used.

We assume a group G of prime order q. Let h be a generator of G, $x \in_R \mathbb{Z}_q^*$ be the secret key of the trustee, $y := h^x$ be her public key, and let $m \in G$ decode the revocation information that is encrypted for her.

We distinguish two cases according to whether the customer uses the original ElGamal encryption scheme or its variation[2].

I: 1. choose $r \in_R \mathbb{Z}_q^*$ and compute $A := h^r$ and $B := y^r m$.

2. provide a proof that m is encrypted in the pair (A, B):

$$V_{\mathrm{I}} := SPK_6\{(\alpha) : A = h^\alpha \ \wedge \ B/m = y^\alpha\}.$$

II: 1. choose $r \in_R \mathbb{Z}_q^*$ and compute $A := y^r$ and $B := h^r m$.

2. provide a proof that m is encrypted in the pair (A, B):

$$V_{\mathrm{II}} := SPK_6\{(\alpha) : A = y^\alpha \ \wedge \ B/m = h^\alpha\}.$$

To revoke a customer's anonymity, the trustee simply needs to decrypt the pair (A, B). This requires her to compute $A^{1/x}$ in case I, and A^x in case II. Furthermore, she often ought to prove that she decrypted correctly, i.e., that she did not deliver a wrong identity. This proof can be given with the signature

$$U_{\mathrm{I}} := SPK_6\{(\alpha) : y = h^\alpha \ \wedge \ B/m = A^\alpha\}(m)$$

[1] Frankel et al. [FTY96] call this an "indirect discourse proof."

[2] The variation is obtained by interchanging the roles of the base h and the trustee's public key y (cf. Section 2.5.3).

in case I, and with

$$U_{\text{II}} := SPK_6\{(\alpha) : y = h^{\alpha} \ \wedge \ A = (B/m)^{\alpha}\}(m)$$

in case II. Finally, we remark that providing U_{I} or U_{II} does not require computations with x^{-1}.

7.2 Sharing the Capability of Anonymity Revocation

In this section, we consider a method of sharing the revocation-capability among several trustees, say $T_1, \ldots, T_n$, such that only defined subsets of them can revoke the anonymity of a customer. The set of all such subsets is denoted by $\Gamma \subseteq 2^{\{T_1,\ldots,T_n\}}$ and called access structure, borrowing terminology from secret sharing.

The basic idea is to share the original trustee's secret key x among the n trustees with a secret sharing scheme for the access structure Γ. Thus $y = h^x$ remains the public key that is used for encryption by the customers and with respect to which the signature U_{I} or U_{II} is provided.

As we saw in the previous section, decryption of the revocation information requires only an exponentiation of a group element with either x or x^{-1} and a single division. Furthermore, the signature U_{I} (or U_{II}) must be computed. Hence, the secret sharing scheme must allow these operations to be performed efficiently and without the trustees getting to know the shares of other trustees of the secret x. Solutions for threshold access structures are described in the following subsection. For general access structure, however, there seem to exist no solutions other than using general multi-party computation.

One thing to consider in multi-party computations are adversaries. It is assumed that all adversaries cooperate to be stronger. One distinguishes them according to their assumed power:

- *Eavesdropping adversaries* are honest during execution of the protocol, but try to get as much information as possible. In particular they try to get hold on the secret key x.

- *Halting adversaries* are eavesdropping adversaries that may stop participating during protocol execution.

- *Malicious adversaries* are eavesdropping adversaries that may arbitrarily deviate from the protocol.

7.2.1 Threshold Access Structures

For access structures with threshold t and n trustees, a realization can for instance be based on Shamir's secret sharing scheme [Sha79] over $\mathbb{Z}_q^*$. The secret key x can be shared using and Feldman's or Pedersen's verifiable secret sharing schemes [Fel87, Ped92]. If powering with x^{-1} is necessary, the trustees have also to compute shares of x^{-1} during the setup of the system. This can for instance be done using results described in [CDM97]. Efficient solutions for powering a group element with x (or x^{-1}) are described in [GJKR96b, Gen96]. Furthermore, to devise a protocol for distributed computation of the signatures U_{I} and U_{II} the techniques presented in [GJKR96b, Gen96] for DSA can be applied . In both cases I and II one can tolerate $e_1 < t \leq n$ eavesdropping adversaries, $e_2 < t \leq (n+1)/2$ halting adversaries, and $e_3 < t \leq (n+1)/2$ malicious adversaries.

7.2.2 Using Publicly Verifiable Secret Sharing

An alternative way to share the revocation capability is to use publicly verifiable secret sharing schemes [Ped92, Sta96a, Gen96]. Roughly speaking, such a scheme works as follows. First, the secret is shared with a suitable ordinary secret sharing scheme. The shares obtained in this manner are encrypted for the designated participants such that it is still possible to verify that the encrypted information is indeed a valid share.

Using such a secret sharing scheme, the customer simply shares the revocation-information m among the trustees according to some access structure Γ. This has the disadvantage that, instead of a single encryption, we have one encryption per trustee. (In our applications, this would lead to larger coins and larger group signatures, respectively.) The advantage is that each customer could choose his own access structure (possibly within some policy-defined constraints).

In this scenario, one has to care only about halting and malicious adversaries among the trustees: The trustee needs only to correctly decrypt the shares, provide them and prove that they were indeed the shares encrypted for them. Thus, to revoke a customer's anonymity, one only needs enough honest trustees to form a qualified set.

7.3 Diverting the Capability of Anonymity Revocation

In this section, we consider the second possibility to reduce the customer's risk of fraudulent anonymity revocation. We assume that there are n trustees $T_1, \ldots, T_n$, where $y_i = h^{\omega_i}$ $(i = 1, \ldots, n)$ are their public keys and the ω_i's their individual secret keys. The customer can now choose which trustees he trusts and then enable only them to revoke his anonymity. If the customer does not mind that others get to know who "his" trustees are, he can just announce them and then prove that he encrypted the revocation-information m for them. However, the customer may not want to divulge whom he trusts. It is this latter case we will consider in the remainder of this section.

The customer sets up some *trust structure* $\Gamma \subseteq 2^{\{1,\cdots,n\}}$, such that Γ contains at least the set $\mathcal{S}$ of "his" trustees. This structure has a rather different meaning than an access structure of secret sharing. We require Γ to be monotone. In applications, the structure Γ might be fixed, or chosen within some policy-defined constraints.

Now the customer has to prove that he encrypted the information $m \in G$ for all trustees corresponding to some set in Γ without stating which set. Thus, the more sets Γ contains, the less information is revealed about which set is the set of trustees that are trusted by the customer. Using a similar idea as for the generalized group signature scheme presented in Chapter 4, the customer can provide this proof as follows. (We only consider encryption according to case I, although the procedure can easily be adapted for case II.)

- for $i \in \mathcal{S}$ choose $r_i \in_R \mathbb{Z}_q^*$ and compute $A_i{:=}h^{r_i}$ and $B_i{:=}y_i^{r_i}m$

- for $j \notin \mathcal{S}$ choose $r_j \in_R \mathbb{Z}_q^*$ and $\tilde{m}_j \in_R G$ and compute $A_j{:=}h^{r_i}$ and $B_j{:=}y_j^{r_j}\tilde{m}_j$

- compute the signature

$$V_{\mathrm{I}}{:=}SPK_{11}\Big\{(\alpha_1,\ldots,\alpha_n) : \bigvee_{S\in\Gamma}\big(\bigwedge_{T_i\in S} A_i = h^{\alpha_i} \wedge B_i/m = y_i^{\alpha_i}\big)\Big\}(m)$$

The properties of the signatures of the type SPK_{11}, ensure that it is not possible to find out which set of trustees the customer trusts. We refer to Chapter 4 for technical details and how the customer can compute V_{I}.

To enable the trustees not trusted by the customer to recognize this fact, the string $\widetilde{m}_j$ could be set to a fixed default string.

Of course the two methods of reducing the risk of fraudulent anonymity revocation can be combined. One way of doing so is to share the secret keys ω_i of this section among several trustees as described in the previous section. This would lead to a trust-structure the elements of which are sets of access-structures, since each ω_i is shared according an access-structure.

To combine the other method of the previous section that uses a verifiable secret sharing scheme, one would choose n public-secret key pairs, share the secret keys using a publicly verifiable secret sharing scheme (with different access structures each time), and then prove that m is encrypted with respect to some of these chosen public keys.

Chapter 8

Concluding Remarks

The group signature schemes and payment systems described in this thesis all build on a common framework. It consists of different methods to prove knowledge and properties of secret discrete logarithms of publicly known values. This framework allows a comprehensive and compact description of our schemes and it is hoped that it will be useful for designing other cryptographic systems as well.

In the area of group signature schemes the contribution of this thesis is twofold. First, the model of group signature schemes is extended to generalized group signature schemes and a realization thereof is presented. This new model allows the definition of sets of group members such that only these sets can collectively sign messages. This is in contrast to previously proposed models in which every group member is able to sign a message on the group's behalf.

Second, the first group signature schemes are proposed in which the length of signatures and the size of the group's public key do *not* depend on the group's size. This has not been achieved by the previously known schemes and is a crucial property in cases the number of group members is large.

The security of the new schemes is based on some new cryptographic assumptions which are a subject of further study. Furthermore, the search for other assumptions that allow the construction of more efficient schemes or can be well founded is an interesting challenge. With respect to generalized group signature schemes suited for large groups, we were not able to find a fully satisfactory solution.

In the area of payment systems the first schemes are presented in which a customer's anonymity can be revoked by a *passive* entity (called trustee). More precisely, this entity needs not to be involved in any protocol other than those for revocation and the initial setup of the system. An on-line scheme and two off-line schemes are described, one of which uses so-called observers to prevent double-spending.

Bibliography

[AC97] Masayuki Abe and Jan Camenisch. Partially blind signatures. In *The 1997 Symposium on Cryptograpy and Information Security*, Fukuoka, Japan, January 1997. The Institute of Electronics, Information and Communcation Engineers. SCSI97-33D.

[AEM87] Leonard M. Adleman, Dennis R. Estes, and Kevin S. McCurley. Solving bivariate quadratic congruences in random polynomial time. *Mathematics of Computation*, 43(177):17–28, January 1987.

[AF96] Masayuki Abe and Eiichiro Fujisaki. How to date blind signatures. In Kwangjo Kim and Tsutomu Matsumoto, editors, *Advances in Cryptology — ASIACRYPT '96*, volume 1163 of *Lecture Notes in Computer Science*, pages 244–251. Springer Verlag, 1996.

[And95] Ross Anderson. The classification of hash functions. In P. G. Farrell, editor, *Codes and Cyphers — Cryptography and Coding IV*, pages 83–94. The Institute of Mathematics and its Applications, 1995.

[AV96] Ross Anderson and Serge Vaudenay. Minding your p's and q's. In Kwangjo Kim and Tsutomu Matsumoto, editors, *Advances in Cryptology — ASIACRYPT '96*, volume 1163 of *Lecture Notes in Computer Science*, pages 26–35. Springer Verlag, 1996.

[Bab85] László Babai. Trading group theory for randomness. In *Proceedings of the Seventeenth Annual ACM Symposium on*

Theory of Computing, pages 421–429, Providence, Rhode Island, 6–8 May 1985.

[BC86] Gilles Brassard and Claude Crépeau. Non-transitive transfer of confidence: a perfect zero-knowledge interactive protocol for SAT and beyond. In *Proc. 27th IEEE Symp. Found. Comp. Sc.*, pages 188–195, 1986.

[BC89] Gilles Brassard and Claude Crépeau. Sorting out zero-knowledge. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology — EUROCRYPT '89*, volume 434 of *Lecture Notes in Computer Science*, pages 150–154. Springer Verlag, 1989.

[BCC88] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37(2):156–189, Oct. 1988.

[BDP97] Antoon Bosselaers, Hans Dobbertin, and Bart Preneel. The RIPEMD-160 cryptographic hash function. *Dr. Dobb's Journal*, pages 24–28, January 1997.

[BF97] Dan Boneh and Matthew Franklin. Efficient generation of shared RSA keys. In Burt Kaliski, editor, *Advances in Cryptology — CRYPTO '97*, volume 1296 of *Lecture Notes in Computer Science*, pages 425–439. Springer Verlag, 1997.

[BFL91] Joan Boyar, Katalin Friedl, and Carsten Lund. Practical zero-knowledge proofs: Giving hints and using deficiencies. *Journal of Cryptology*, 4(3):185–206, 1991.

[BG92] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In Ernest F. Brickell, editor, *Advances in Cryptology — CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*, pages 390–420. Springer-Verlag, 1992.

[BGK95] Ernie Brickell, Peter Gemmel, and David Kravitz. Trustee-based tracing extensions to anonymous cash and the making of anonymous change. In *Proceedings of the Sixth Annual ACM-SIAMs*, pages 457–466. Association for Computing Machinery, January 1995.

[BL90] Josh Benaloh and Jerry Leichter. Generalized secret sharing and monotone functions. In Shafi Goldwasser, editor, *Advances in Cryptology — CRYPTO '88*, volume 403 of *Lecture Notes in Computer Science*, pages 27–35. Springer-Verlag, 1990.

[Bla79] George Robert Blakley. Safeguarding cryptographic keys. In *Proceedings of the National Computer Conference 1979*, volume 48 of *American Federation of Information Processing Societies Proceedings*, pages 313–317, 1979.

[Bla83] Richard E. Blahut. *Theory and Practice of Error Control Codes*. Addison-Wesley, Reading, 1983.

[Ble96] Daniel Bleichenbacher. Generating ElGamal signatures without knowing the secret key. In Ueli Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 10–18. Springer Verlag, 1996.

[Blo85] Rolf Blom. An optimal class of symmetric key generation systems. In Thomas Beth, Norbert Cot, and Ingemar Ingemarsson, editors, *Advances in Cryptology — Proceedings of EUROCRYPT 84*, volume 209 of *Lecture Notes in Computer Science*, pages 335–338. Springer-Verlag, 1985.

[BM88] László Babai and Shlomo Moran. Arthur - Merlin games: A randomized proof system, and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36, 1988.

[Boy89] Colin Boyd. Digital multisignatures. In Henry J. Beker and Fred C. Piper, editors, *Cryptography and Coding*, pages 241–246. The Institute of Mathematics and its Applications Conference Series, Oxford University Press, 1989.

[BP96] Joan Boyar and René Peralta. Short discreet proofs. In Ueli Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 131–142. Springer Verlag, 1996.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In

First ACM Conference on Computer and Communication Security, pages 62–73. Association for Computing Machinery, 1993.

[BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signature - how to sign with RSA and Rabin. In Ueli Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 399–416. Springer Verlag, 1996.

[BR97] Mihir Bellare and Phillip Rogaway. Collision-resistant hashing: Towards making UOWHFs practical. In Burton S. Kaliski Jr. editor, *Advances in Cryptology — CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 470–484. Springer Verlag, 1997.

[Bra93] Stefan Brands. An efficient off-line electronic cash system based on the representation problem. Technical Report CS-R9323, CWI, April 1993.

[Bra97] Stefan Brands. Rapid demonstration of linear relations connected by boolean operators. In Walter Fumy, editor, *Advances in Cryptology — EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 318–333. Springer Verlag, 1997.

[Cam] Jan Camenisch. The group signature scheme of Park, Lee, and Won is not secure. Unpublished manuscript.

[Cam97] Jan Camenisch. Efficient and generalized group signatures. In Walter Fumy, editor, *Advances in Cryptology — EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 465–479. Springer Verlag, 1997.

[Can95] Ran Canetti. *Studies in Secure Multiparty Computation and Applications*. PhD thesis, Weizmann Institute of Science, Rehovot 76100, Israel, June 1995.

[CD97] Ronald Cramer and Ivan Damgård. Linear zero-knowledge: A note on efficient zero-knowledge proofs and arguments. In *Proceedings of ACM STOC '97*, pages 436–445. ACM press, 1997.

[CDEH$^+$96] James Cowie, Bruce Dodson, R. Marije Elkenbracht-Huizing, Arjen K. Lenstra, Peter L. Montgomery, and Jörg Zayer. A world wide number field sieve factoring record: On to 512 bits. In Kwangjo Kim and Tsutomu Matsumoto, editors, *Advances in Cryptology — ASIACRYPT '96*, volume 1163 of *Lecture Notes in Computer Science*, pages 382–394. Springer Verlag, 1996.

[CDM97] Ronald Cramer, Ivan Damgård, and Ueli Maurer. Span programs and general secure multi-party computation. Technical Report RS-97-28, Basic Research in Computer Science, University of Aarhus, November 1997.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo G. Desmedt, editor, *Advances in Cryptology — CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer Verlag, 1994.

[CEvdG88] David Chaum, Jan-Hendrik Evertse, and Jeroen van de Graaf. An improved protocol for demonstrating possession of discrete logarithms and some generalizations. In David Chaum and Wyn L. Price, editors, *Advances in Cryptology — EUROCRYPT '87*, volume 304 of *Lecture Notes in Computer Science*, pages 127–141. Springer-Verlag, 1988.

[CFN90] David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In Shafi Goldwasser, editor, *Advances in Cryptology — CRYPTO '88*, volume 403 of *Lecture Notes in Computer Science*, pages 319–327. Springer Verlag, 1990.

[CFPR96] Don Coppersmith, Matthew Franklin, Jacques Patarin, and Michael Reiter. Low-exponent RSA with related messages. In Ueli Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 1–9. Springer Verlag, 1996.

[CGMA85] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *Proc. 26th IEEE Symp. Found. Comp. Sc.*, pages 383–395, 1985.

[CH89] R.A. Croft and S.P. Harris. Public key cryptography and re-usable shared secrets. In Henry J. Beker and F.C. Piper, editors, *Cryptography and Coding*, pages 189–201. The Institute of Mathematics and its Applications Conference Series, Oxford Science Publications, 1989.

[Cha83] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *Advances in Cryptology — Proceedings of CRYPTO '82*, pages 199–203. Plenum Press, 1983.

[Cha84] David Chaum. Blind signature systems. In David Chaum, editor, *Advances in Cryptology — CRYPTO '83*, page 153. Plenum Press, 1984.

[Cha85] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, October 1985.

[Cha87] David Chaum. Demonstrating that a public predicate can be satisfied without revealing any information about how. In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 195–199. Springer-Verlag, 1987.

[Cha91] David Chaum. Zero-knowledge undeniable signatures. In Ivan Bjerre Damgård, editor, *Advances in Cryptology — EUROCRYPT '90*, volume 473 of *Lecture Notes in Computer Science*, pages 458–464. Springer-Verlag, 1991.

[Cha92] David Chaum. Achieving electronic privacy. *Scientific American*, pages 96–101, August 1992.

[Che94] Lidong Chen. *Witness Hiding Proofs and Applications*. Ph.D. Thesis, DAIMI PB – 477, Computer Science Departement, Aarhus University, August 1994.

[CLR92] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, 1992.

[CMS96] Jan Camenisch, Ueli Maurer, and Markus Stadler. Digital payment systems with passive anonymity-revoking trustees. In Elisa Bertino, Helmut Kurth, Giancarlo Martella, and Emilio Montolivo, editors, *Computer Security — ESORICS 96*, volume 1146 of *Lecture Notes in Computer Science*, pages 33–43. Springer Verlag, 1996.

[CMS97] Jan Camenisch, Ueli Maurer, and Markus Stadler. Digital payment systems with passive anonymity-revoking trustees. *Journal of Computer Security*, 5(1):69–89, 1997.

[Coh93] Henri Cohen. *A Course in Computational Algebraic Number Theory*. Number 138 in Graduate Texts in Mathematics. Springer-Verlag, Berlin, 1993.

[COS86] Don Coppersmith, Andrew Odlyzko, and Richard Schroeppel. Discrete logarithms in $GF(p)$. *Algorithmica*, 1:1–15, 1986.

[CP93] David Chaum and Torben Pryds Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *Advances in Cryptology — CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105. Springer-Verlag, 1993.

[CP94] Ronald J. F. Cramer and Torben P. Pedersen. Improved privacy in wallets with observers. In Tor Helleseth, editor, *Advances in Cryptology — EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*, pages 329–343. Springer-Verlag, 1994.

[CP95] Lidong Chen and Torben Pryds Pedersen. New group signature schemes. In Alfredo De Santis, editor, *Advances in Cryptology — EUROCRYPT '94*, volume 950 of *Lecture Notes in Computer Science*, pages 171–181. Springer-Verlag, 1995.

[CPS94a] Jan L. Camenisch, Jean-Marc Piveteau, and Markus A. Stadler. Blind signatures based on the discrete logaritm problem. In Alfredo De Santis, editor, *Advances in Cryptology — EUROCRYPT '94*, volume 950 of *Lecture Notes in Computer Science*, pages 428–432. Springer Verlag Berlin, 1994.

[CPS94b] Jan L. Camenisch, Jean-Marc Piveteau, and Markus A. Stadler. An efficient payment system protecting privacy. In Dieter Gollmann, editor, *Computer Security — ESORICS 94*, volume 875 of *Lecture Notes in Computer Science*, pages 207–215. Springer Verlag, 1994.

[CPS95] Jan Camenisch, Jean-Marc Piveteau, and Markus Stadler. Faire Anonyme Zahlungssysteme. In F. Huber-Wäschle, H. Schauer, and P. Widmayer, editors, *GISI 95*, Informatik aktuell, pages 254–265. Springer Verlag Berlin, September 1995.

[CPS96] Jan Camenisch, Jean-Marc Piveteau, and Markus Stadler. An efficient fair payment system. In *3rd ACM Conference on Computer and Communicatons Security*, pages 88–94, New Delhi, March 1996. Association for Computing Machinery.

[Cra97] Ronald Cramer. *Modular Design of Secure yet Practical Cryptographic Protocol.* PhD thesis, University of Amsterdam, 1997.

[CS97a] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In Burt Kaliski, editor, *Advances in Cryptology — CRYPTO '97*, volume 1296 of *Lecture Notes in Computer Science*, pages 410–424. Springer Verlag, 1997.

[CS97b] Jan Camenisch and Markus Stadler. Proof systems for general statements about discrete logarithms. Technical Report TR 260, Institute for Theoretical Computer Science, ETH Zürich, March 1997.

[Cv90] David Chaum and Hans van Antwerpen. Undeniable signatures. In *Advances in Cryptology — CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 212–216. Springer-Verlag, 1990.

[CvH91] David Chaum and Eugène van Heyst. Group signatures. In Donald W. Davies, editor, *Advances in Cryptology — EUROCRYPT '91*, volume 547 of *Lecture Notes in Computer Science*, pages 257–265. Springer-Verlag, 1991.

[Dam88] Ivan Bjerre Damgård. Collision free hash functions and public key signature schemes. In David Chaum and Wyn L. Price, editors, *Advances in Cryptology — EUROCRYPT '87*, volume 304 of *Lecture Notes in Computer Science*, pages 203–216. Springer-Verlag, 1988.

[Dam94] Ivan Bjerre Damgård. Practical and provable secure release of a secret and exchange of signatures. In Tor Helleseth, editor, *Advances in Cryptology — EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*, pages 200–217. Springer-Verlag, 1994.

[Dam95] Ivan Bjerre Damgård. Practical and provable secure release of a secret and exchange of signatures. *Journal of Cryptology*, 8(4):201–22, 1995.

[dB90] Bert den Boer. Diffie-Hellman is as strong as discrete log for certain primes. In Shafi Goldwasser, editor, *Advances in Cryptology — CRYPTO '88*, volume 403 of *Lecture Notes in Computer Science*, pages 520–539. Springer-Verlag, 1990.

[DBP96] Hans Dobbertin, Antoon Bosselaers, and Bart Preneel. RIPEMD-160: A strengthened version of RIPEMD. In Dieter Gollmann, editor, *Fast Software Encryption*, volume 1039 of *Lecture Notes in Computer Science*, pages 71–82. Springer Verlag, 1996.

[DDFY94] Alfredo De Santis, Yvo Desmedt, Yair Frankel, and Moti Yung. How to share a function securely. In *Proc. 26th ACM Symposium on Theory of Computing (STOC)*, pages 522–533, 1994.

[DH76] Whitfield Diffie and Martin E Hellman. New directions in cryptography. *IEEE Trans. on Information Theory*, IT-22(6):644–654, Nov. 1976.

[Dob96] Hans Dobbertin. Cryptanalysis of MD5 compress. Presented at the Rump-session of EUROCRYPT '96, may 1996.

[ElG85a] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In

George Robert Blakley and David Chaum, editors, *Advances in Cryptology — CRYPTO '84*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer Verlag, 1985.

[ElG85b] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. on Information Theory*, IT-31(4):469–472, July 1985.

[Fel87] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proc. 28th IEEE Symp. Found. Comp. Sc.*, pages 427–437, 1987.

[FFS88] Uriel Feige, Amos Fiat, and Adi Shamir. Zero-knowledge proofs of identity. *Journal of Cryptology*, 1:77–94, 1988.

[FO97] Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In Burton S. Kaliski Jr. editor, *Advances in Cryptology — CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 16–30. Springer Verlag, 1997.

[Fra93] Matthew K. Franklin. *Complexity and Security of Distributed Protocols*. PhD thesis, Columbia University, 1993.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solution to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer Verlag, 1987.

[FS90] Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing*, pages 416–426, May 1990.

[FTY96] Yair Frankel, Yiannis Tsiounis, and Moti Yung. "Indirect discourse proofs:" Achieving efficient fair off-line e-cash. In Kwangjo Kim and Tsutomu Matsumoto, editors, *Advances in Cryptology — ASIACRYPT '96*, volume 1163 of *Lecture Notes in Computer Science*, pages 286–300. Springer Verlag, 1996.

[Gen96] Rosario Gennaro. *Theory and Practice of Verifiable Secret Sharing*. PhD thesis, Massachusetts Institute of Technology, May 1996.

[GJKR96a] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Robust and efficient sharing of RSA functions. In Neal Koblitz, editor, *Advances in Cryptology — CRYPT0 '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 157–172, Berlin, 1996. IACR, Springer Verlag.

[GJKR96b] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold DSS signatures. In Ueli Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 354–371. Springer Verlag, 1996.

[GKR97] Rosario Gennaro, Hugo Krawczyk, and Tal Rabin. RSA-based undeniable signatures. In Burt Kaliski, editor, *Advances in Cryptology — CRYPTO '97*, volume 1296 of *Lecture Notes in Computer Science*, pages 132–149. Springer Verlag, 1997.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, April 1984.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. In *Proc. 27th Annual Symposium on Foundations of Computer Science*, pages 291–304, 1985.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[GMW87a] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proc. 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.

[GMW87b] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to prove all NP statements in zero-knowledge and a methodology of cryptographic protocol design. In Andrew M. Odlyzko, editor, *Advances in Cryptclogy — CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 171–185. Springer-Verlag, 1987.

[GO94] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7(1):1–32, 1994.

[Gol95] Oded Goldreich. Foundations of Cryptography (Fragments of a Book). Available via the internet, Departement of Computer Science and Applied Mathematics, Weizmann Intitute of Science, Rehovot, Israel, February 1995.

[Hås88] Johan Håstad. Solving simultaneous modular equations of low degree. *SIAM Journal on Computing*, 17(2):336–341, April 1988.

[HMP95] Patrick Horster, Markus Michels, and Holger Petersen. Meta-message recovery and meta-blind signature schemes based on the discrete logarithm problem and their applications. In Josef Pieprzyk and Reihanah Safavi-Naini, editors, *Advances in Cryptology — ASIACRYPT '94*, volume 917 of *Lecture Notes in Computer Science*, pages 224–237. Springer Verlag Berlin, 1995.

[JY96] Markus Jakobsson and Moti Yung. Revokable and versatile electronic money. In *3rd ACM Conference on Computer and Communicatons Security*, pages 76–87, New Delhi, March 1996. Association for Computing Machinery.

[Knu81] Donald Ervin Knuth. *The Art of Computer Programming*, volume 2 — Seminumerical algorithms. Addison-Wesley, second edition, 1981.

[Kob94] Neal Koblitz. *A course in number theory and cryptography.* Springer-Verlag, second edition, 1994.

[KP97] Joe Kilian and Erez Petrank. Identity escrow. Theory of Cryptography Library, Record Nr. 97-11, `http://theory.lcs.mit.edu/~tcryptol`, August 1997.

[KPW96] Seung Joo Kim, Sung Jun Park, and Dong Ho Won. Convertible group signatures. In Kwangjo Kim and Tsutomu Matsumoto, editors, *Advances in Cryptology — ASIACRYPT '96*, volume 1163 of *Lecture Notes in Computer Science*, pages 311–321. Springer Verlag, 1996.

[Kra86] Evangelos Kranakis. *Primality and Cryptography.* Wiley-Teubner Series in Computer Science, 1986.

[Kra94] Hugo Krawczyk. Secret sharing made short. In Douglas R. Stinson, editor, *Advances in Cryptology — CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 136–146. Springer-Verlag, 1994.

[Lai92] Xuejia Lai. *On the Design and Security of Block Ciphers*, volume 1 of *ETH Series in Information Processing.* Hartung-Gorre Verlag Konstanz, 1992.

[LL97] Chae Hoon Lim and Pil Joong Lee. A key recovery attack on discrete log-based schemes using a prime order subgroup. In Burt Kaliski, editor, *Advances in Cryptology — CRYPTO '97*, volume 1296 of *Lecture Notes in Computer Science*, pages 249–263. Springer Verlag, 1997.

[LLJ93] A.K. Lenstra and H.W. Lenstra Jr. *The Development of the Number Field Sieve*, volume 1554 of *Lecture Notes in Mathematics.* Springer Verlag, 1993.

[LM91] Xuejia Lai and James L. Massey. A proposal for a new block encryption standard. In Ivan Bjerre Damgård, editor, *Advances in Cryptology — EUROCRYPT '90*, volume 473 of *Lecture Notes in Computer Science*, pages 389–404. Springer-Verlag, 1991.

[LM93] Xuejia Lai and James L. Massey. Hash functions based on block ciphers. In Rainer A. Rueppel, editor, *Advances in Cryptology — EUROCRYPT '92*, volume 658 of *Lecture Notes in Computer Science*, pages 55–70. Springer-Verlag, 1993.

[Mau94] Ueli Maurer. Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete logarithms. In *Advances in Cryptology — CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 271–281. Springer Verlag, 1994.

[McC90a] Kevin McCurley. The discrete logarithm problem. In Carl Pomerance, editor, *Cryptology and computational number theory*, volume 42 of *Proceedings of Symposia in Applied Mathematics*, pages 49–74. American Mathematical Society, 1990.

[McC90b] Kevin McCurley. Odds and ends from cryptology and computational number theory. In Carl Pomerance, editor, *Cryptology and computational number theory*, volume 42 of *Proceedings of Symposia in Applied Mathematics*, pages 145–166. American Mathematical Society, 1990.

[Men93] Alfred Menezes. *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, 1993.

[Mic96] Markus Michels. Comments on some group signature schemes. Technical Report TR-96-3-D, Departement of Computer Science, University of Technology, Chemnitz-Zwickau, November 1996.

[MUO96] Masahiro Mambo, Keisuke Usuda, and Eiji Okamoto. Proxy signatures for delegating signing operation. In *3rd ACM Conference on Computer and Communicatons Security*, pages 48–57, New Delhi, March 1996. Association for Computing Machinery.

[MvOV97] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, FL, 1997.

[MW] Ueli Maurer and Stefan Wolf. Lower bounds on generic algorithms in groups. Manuscript.

[MW96] Ueli Maurer and Stefan Wolf. On the complexity of breaking the diffie-hellman protocol. Technical report, Institute for Theoretical Computer Science, ETH Zürich, April 1996.

[Nat93] National Institute of Standards and Technology. NIST FIPS PUB 180: Secure hash standard, May 1993.

[Nat94] National Institute of Standards and Technology. NIST FIPS PPU 186: Digital signature standard, May 1994.

[NR93] Kaisa Nyberg and Rainer A. Rueppel. A new signature scheme based on the DSA giving message recovery. In *First ACM Conference on Computer and Communication Security*. Association for Computing Machinery, 1993.

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, pages 33–43, Seattle, Washington, 15–17 May 1989. ACM.

[Odl94] Andrew M. Odlyzko. Discrete logarithm and smooth polynomials. In Gary L. Mullen and Peter Jau-Shyong Shiue, editors, *Finite Fields: Theory, Applications and Algorithms*, volume 168 of *Contemporary Mathematics*, pages 269–278. American Mathematical Society, 1994.

[Oka93] Tatsuaki Okamoto. Provable secure and practical identification schemes and corresponding signature schemes. In Ernest F. Brickell, editor, *Advances in Cryptology — CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*, pages 31–53. Springer-Verlag, 1993.

[Oka97] Tatsuaki Okamoto. Threshold key-recovery systems for RSA. In Mark Lomas and Serge Vaudenay, editors, *Security Protocols Workshop*, Paris, 1997.

[OO92] Tatsuaki Okamoto and Kazuo Ohta. Universal electronic cash. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 324–337. Springer-Verlag, 1992.

[OO93] Kazuo Ohta and Tatsuaki Okamoto. A digital multisignature scheme based on the Fiat-Shamir scheme. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *Advances in Cryptology — ASIACRYPT '91*, volume 739 of

Lecture Notes in Computer Science, pages 139–148. Springer-Verlag, 1993.

[OSS84] H. Ong, Claus P. Schnorr, and Adi Shamir. Efficient signature schemes based on polymonial equations. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology — CRYPTO '84*, volume 196 of *Lecture Notes in Computer Science*, pages 37–46. Springer Verlag, 1984.

[Pap94] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, 1994.

[Ped92] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer Verlag, 1992.

[Pet96] Holger Petersen. *Digitale Signaturverfahren auf der Basis des diskreten Logarithmusproblems und ihre Anwendungen*. PhD thesis, Technische Universiät Chemnitz-Zwickau, 1996.

[Pet97] Holger Petersen. How to convert any digital signature scheme into a group signature scheme. In Mark Lomas and Serge Vaudenay, editors, *Security Protocols Workshop*, Paris, 1997.

[PH78] Stephen C. Pohlig and Martin E. Hellman. An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance. *IEEE Trans. Inform. Theory*, IT-24:106–110, January 1978.

[PLW95] Sung Jun Park, In Sook Lee, and Dong Ho Won. A practical group signature. In *Proceedings of the 1995 Japan-Korea Workshop on Information Security and Cryptography*, pages 127–133, January 1995.

[Poi96] David Pointcheval. *Les Preuves de Connaissance et leurs Preuves de Sécurité*. PhD thesis, Université de Caen, 1996.

[Pol75] John M. Pollard. A monte carlo method for factorization. *bit*, 15:331–334, 1975.

[Pol78] John M. Pollard. Monte Carlo methods for index computation (mod p). *Mathematics of Computation*, 32(143):918–924, July 1978.

[Pom85] Carl Pomerance. The quadratic sieve factoring algorithm. In Thomas Beth, Norbert Cot, and Ingemar Ingemarsson, editors, *Advances in Cryptology — Proceedings of EUROCRYPT 84*, volume 209 of *Lecture Notes in Computer Science*, pages 169–182. Springer-Verlag, 1985.

[Pre93] Bart Preneel. *Analsyis and Design of Cryptographic Hash Functions*. PhD thesis, Katholieke Universiteit Leuven, 1993.

[PS87] John M. Pollard and Claus P. Schnorr. An efficient solution of the congruence $x^2 + ky^2 = m \pmod{n}$. *IEEE Transactions on Information Theory*, 33(5):702–709, September 1987.

[PS96] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 387–398. Springer Verlag, 1996.

[Riv92] Ron Rivest. The MD5 message-digest algorithm. RFC 1321, April 1992.

[RSA78] Ronld Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[Sch91] Claus P. Schnorr. Efficient signature generation for smart cards. *Journal of Cryptology*, 4(3):239–252, 1991.

[Sch96] Bruce Schneier. *Applied Cryptography*. Wiley, New York, second edition, 1996.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, November 1979.

[Sha90] Adi Shamir. IP = PSPACE. In *31st Annual Symposium on Foundations of Computer Science*, volume I, pages 11–15, St. Louis, Missouri, 22–24 October 1990. IEEE.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Advances in Cryptology — EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266. Springer Verlag, 1997.

[Sim91] Gustavus J. Simmons. An introduction to shared secret and/or shared control schemes and their applications. In Gustavus J. Simmons, editor, *Contemporary Cryptology: The Science of Information Integrity*, pages 441–497. IEEE Press, 1991.

[SPC95] Markus Stadler, Jean-Marc Piveteau, and Jan Camenisch. Fair blind signatures. In Louis C. Guillou and Jean-Jacques Quisquater, editors, *Advances in Cryptology — EUROCRYPT '95*, volume 921 of *Lecture Notes in Computer Science*, pages 209–219. Springer Verlag, 1995.

[Sta96a] Markus Stadler. *Cryptographic Protocols for Revocable Privacy.* Ph.D. Thesis, ETH Zürich, 1996. Diss. ETH No. 11651.

[Sta96b] Markus Stadler. Publicly verifiable secret sharing. In Ueli Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 191–199. Springer Verlag, 1996.

[Sti92] D. R. Stinson. An explication of secret sharing schemes. *Designs, Codes and Cryptography*, 2:357–390, 1992.

[Sti95] Douglas R. Stinson. *Cryptography: Theory and Practice.* CRC Press, 1995.

[TW87] Martin Tompa and Heather Woll. Random self-reducibility and zero knowledge interactive proofs of possession of information. In *28th Annual Symposium on Foundations of Computer Science*, pages 472–482, Los Angeles, California, 12–14 October 1987. IEEE.

[vHP93] Eugène van Heyst and Torben Pryds Pedersen. How to make efficient fail-stop signatures. In Rainer A. Rueppel, editor, *Advances in Cryptology — EUROCRYPT '92*, volume

658 of *Lecture Notes in Computer Science,* pages 366–377. Springer-Verlag, 1993.

[vSN92] Sebastiaan von Solms and David Naccache. On blind signatures and perfect crimes. *Computer & Security,* 11(6):581–583, 1992.

[WP90] Michael Waidner and Birgit Pfitzmann. The dining cryptographers in the disco: Unconditional sender and recipient untraceability with computationally secure serviceability. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology — EUROCRYPT '89,* volume 434 of *Lecture Notes in Computer Science,* page 690. Springer Verlag, 1990.

Leer - Vide - Empty

Index

Curriculum Vitae

1968 Born on April 25, 1968 in Chur, Switzerland.

1974–82 Primarschule at Langwies, Sekundarschule at Arosa.

1982–87 Kantonsschule Graubünden (Gymnasium) at Chur: Matura Typus C.

1988–93 ETH Zurich, Dept. of Electrical Engineering: Dipl. El.-Ing. ETH.

1989 Internship at Schmid Fernmeldetechnik, Zurich, Switzerland.

1990–91 Full-time and later part-time position as El.-Ing. ETH at Martignoni Electronics AG, Zurich, Switzerland.

1993–98 Teaching and research assistant in Prof. U. Maurer's "Information Security and Cryptography" group at ETH Zurich.