NTNU
Innovation and Creativity

Security analysis of blind signatures and group signatures

Børge Nordli

Master of Science in Physics and Mathematics
Submission date: July 2007
Supervisor: Aslak Bakke Buan, MATH

Norwegian University of Science and Technology
Department of Mathematical Sciences

Problem Description

Blind signatures and group signatures are useful as subprotocols in systems for digital cash. A rigorous security analysis of such protocols is hence of interest. Crucial steps in such an analysis have been carried out by Juels et. al. in the blind signatures case, and by Bellare et. al. in the case of group signatures. The object of this master-thesis is to give an overview of the recent advances to this theory, and if possible, contribute to further developments.

Assignment given: 08. February 2007
Supervisor: Aslak Bakke Buan, MATH

Abstract

We present the latest formal security definitions for blind signature schemes and for group signature schemes. We start by introducing theory about algorithms, probability distributions, distinguishers, protocol attacks and experiments, which is needed to understand the definitions. In the case of blind signatures, we define blindness and non-forgeability, and we present the blind Schnorr and Okamoto-Schnorr signature schemes and prove that the latter is secure. For group signatures, we define full-anonymity and full-non-forgeability (full-traceability). In the end, we sketch a secure general group signature scheme presented by Bellare, Micciancio and Warinschi.

Preface

This Master's thesis is written for the Algebra group in the Department of Mathematical Sciences of the Norwegian University of Science and Technology (NTNU). This thesis concludes five years of mathematical studies, of which the last two years have been devoted to algebra, specifically cryptography. It is a full semester's worth of work, and it is a continuation of my last semester's literature study project concerning digital cash systems [18]. I have enjoyed to continue studying blind signatures in more formal detail, as well as looking at the more recent concept of group signatures.

It is nice to see that there exists a formal framework around digital signatures, which I was not aware of earlier, and I hope that the theory around various digital signature schemes in the future will come to practical use.

I would like to thank my supervisor Aslak Bakke Buan for guidance and comments, and for providing me with relevant literature. I would also thank Kristian Gjøsteen for his insight on cryptographic protocols, and Lillian Kråkmo for assistance on the Okamoto-Schnorr scheme. Finally, I would like to thank my fellow students Halvor Sakshaug and Pål Løvhaugen for helpful proofreading.

Børge Nordli
6 July 2007
Trondheim, Norway

Contents

0 Introduction **1**

1 Theory **3**
1.1 Algorithms . . . 3
1.2 Distributions and distinguishers . . . 4
1.3 Protocol attacks . . . 6
1.3.1 Attacks on digital signature schemes . . . 6
1.3.2 Forgeries on digital signature schemes . . . 8
1.4 Provable security . . . 8
1.5 Experiments . . . 10

2 Blind signatures **13**
2.1 Definition . . . 14
2.2 Security requirements . . . 15
2.2.1 Blindness . . . 15
2.2.2 Non-forgeability . . . 17
2.2.3 Summary . . . 18
2.3 Two blind signature schemes . . . 19

3 Group signatures **25**
3.1 Definition . . . 25
3.2 Security requirements . . . 27
3.2.1 Full-anonymity . . . 27
3.2.2 Full-non-forgeability . . . 29
3.2.3 Other security requirements . . . 30
3.2.4 Summary . . . 33
3.3 A secure general group signature scheme . . . 34

4 Conclusion **37**

Bibliography **39**

Chapter 0

Introduction

David Chaum was interested in creating a digital version of regular cash, and invented in 1982 blind signatures in order to maintain the anonymity of the users of digital cash [10]. The concept was kept alive and used in several cash systems, for example by Brands [7]. However, as with many cryptographic primitives, it took a long time before a strict mathematical definition appeared. It was Juels, Luby and Ostrovsky [16] who first formally defined a blind digital signature scheme. They also identified two main security requirements (blindness and non-forgeability) for such schemes. The definitions have later been studied and revised, for instance by Pointcheval and Stern [22], Okamoto [20] and Hazay et al. [15]. Today, there seems to be a strong agreement on the definition of blind signature schemes and their security requirements, and efficient schemes have been presented and proved correct with respect to these two security properties.

In 1991, Chaum and van Heyst invented another type of digital signature, namely the group signature scheme [11]. Group signatures were originally constructed for generalising identification primitives, but has later been seen useful both in digital payment and voting systems [1]. Ateniese and Tsudik [2] revived this concept in 1999 and tried to summarise the security requirements that seemed to apply. Later, Bellare et al. [3, 5] have studied group signatures extensively, and they have made formal definitions, both of group signature schemes and of their security requirements. They found that all earlier mentioned requirements can be united into two general requirements, namely full-anonymity and full-traceability.

The purpose of this thesis is to give an overview of the recent advances within blind and group signature schemes, and to present formal definitions of the schemes and their security requirements.

We assume the reader has basic knowledge of algebra and cryptography, as taught in standard university courses. This includes finite groups, discrete logarithms, hash functions, and basic encryption and signature schemes.

Since algorithms and adversaries in these courses often are rather superficially treated, Chapter 1 will commence with a discussion of algorithms, giving weight on the probabilistic type. We will then proceed to probability distributions and to distinguishers. In order to discuss security requirements, we begin with presenting attacks and forgeries on digital signature schemes, and we then discuss provable security, especially complexity-based proofs. We end the theory with introducing the concept of experiments, which is a common tool to formalise adversaries.

Chapter 2 is devoted to blind signature schemes, and we provide a formal definition of such schemes and their security requirements. We then present the blind Schnorr and Okamoto-

Schnorr signature schemes. We stress the similarities between the two by presenting them together, in a format we have not seen elsewhere. We prove that they both are blind, and sketch the proof of Pointcheval and Stern [22] of the non-forgeability of the Okamoto-Schnorr scheme.

In Chapter 3, we treat group signature schemes in the same way, so we begin with defining them and their security requirements. Then we mention the many other requirements that existed earlier [2] and discuss why all of them can be incorporated into the two new requirements. We end this thesis by sketching the general group signature scheme of Bellare, Micciancio and Warinschi [3], which is proven to be secure under the assumption that trapdoor permutations exist.

Chapter 1

Theory

This chapter introduces notation and concepts needed in order to formalise signatures schemes and their security requirements.

A function $f\colon \mathbb{N} \to \mathbb{R}$ is *negligible* if for each $c > 0$, there exists an integer N, such that for all integers $n > N$, $f(n) < n^{-c}$. Otherwise, it is *non-negligible*. Likewise a function g is *overwhelming* if $1 - g(n)$ is negligible. When dealing with security requirements of group signatures, we need to formalise non-negligibility of two-argument functions [3]: A two-argument function $f\colon \mathbb{N} \times \mathbb{N} \to \mathbb{R}$ is *negligible* if $f_n\colon \mathbb{N} \to \mathbb{R}$, $f_n(k) = f(k, n(k))$ is negligible for every polynomially bounded function $n\colon \mathbb{N} \to \mathbb{N}$.

To make the distinction clear between comparisons and assignments in the description of algorithms and experiments, we use '$\leftarrow$' for assignments and '$=$' for comparison. We write $x \leftarrow y$ when x is assigned the value of y, and $x \leftarrow \mathcal{A}(y)$ when x is assigned the output of the algorithm $\mathcal{A}$ run with the input y.

Also, we write $x \xleftarrow{r} X$ when x is assigned with an element from the set X, uniformly chosen, and $x \xleftarrow{\mu} X$ when x is sampled according to a distribution μ on X. When dealing with algorithms, we write PPT as a short hand for probabilistic polynomial time, and DPT for deterministic polynomial time.

1.1 Algorithms

In cryptographic models, we view all participants of the model as algorithms of some kind. Algorithms are most often treated informally as a sequence of elementary steps that can be performed by a machine [13].

In a more formal framework, an algorithm is a sequence of instructions that can be carried out by a Turing machine, or more general, any Turing-complete system. The algorithm receives input (possibly empty), and should after some time terminate with some output. In the language of Turing machines, the machine receives the encoding of the input on its working tape, and terminates with the encoding of the output on the same tape [17].

A *probabilistic algorithm* has, in addition to its input/working/output tape, access to a *random tape*, which is a tape containing a random bit string, and the output of the algorithm is in general not determined from the input alone, but is also dependent of the contents of the random tape. On the contrary, a *deterministic algorithm* has no such inherent randomness, and such an algorithm can be considered as a computable function. The running of the algorithm with a given input determines the value of the function at this point.

An *interactive algorithm* has, in addition to its main tape (and possibly its random tape), two communication tapes, which it can use to send and receive messages to and from another interactive algorithm. We refer to [14], which has a thorough treatment of interactive algorithms.

The running time of an algorithm is of main interest, since the distinction between efficient and inefficient algorithms is very decisive. An algorithm is called a *polynomial time* (or *efficient*) algorithm if there exists a polynomial p such that for any input x, the algorithm stops within $p(|x|)$ steps [17], where $|x|$ is the size, or the length, of x. Problems that are solvable by a polynomial time algorithm are called *feasible*, the rest of them are called *infeasible*.

1.2 Distributions and distinguishers

Since most cryptographic algorithms are probabilistic, we will introduce the concept of probability distributions and discover that a probabilistic algorithm can be thought of as a probability distribution. The definition of a distribution is standard, but we will also introduce the concepts of induced distributions and the distance between two distributions. Most of the contents of this section is gathered from Gjøsteen [13] and Pointcheval and Stern [22].

Definition 1.1 (Distribution, induced distribution)**.** A *distribution* on a finite set S is a function $\mu\colon S \to [0,1]$ such that $\sum_{s\in S}\mu(s) = 1$. Given a distribution μ_0 and a function $f\colon S_0 \to S_1$, the distribution μ_1 on S_1 *induced by* μ_0 *and the function* f is defined by

$$\mu_1(s) = \sum_{t\in f^{-1}(s)} \mu_0(t).$$

A deterministic algorithm is equivalent to a function, and thus a deterministic algorithm can induce a distribution in the same way. We also generalise this definition to probabilistic algorithms: Let $\mathcal{A}\colon S_0 \to S_1$ be a probabilistic algorithm, and μ_0 a distribution on the input set S_0, then the distribution $\mathcal{A}_{\mu_0}$ on S_1 *induced by* μ_0 *and the algorithm* $\mathcal{A}$ is defined by

$$\mathcal{A}_{\mu_0}(s) = \Pr[\mathcal{A}(t) = s \mid t \xleftarrow{\mu_0} S_0] = \sum_{t\in S_0} \Pr[\mathcal{A}(t) = s]\mu_0(t).$$

We see that the two definitions coincide, because a deterministic algorithm has only one possible output for any single input, and $\Pr[\mathcal{A}(t) = s] = 1$ exactly if t is in the preimage of the equivalent function, and 0 otherwise. If we fix the input t of a probabilistic algorithm $\mathcal{A}$, then we can identify the algorithm with its output distribution on this input. We call this distribution $\mathcal{A}_{(t)}$, and it can for example be defined by considering the induced distribution of μ and $\mathcal{A}$, where $\mu(t) = 1$.

Definition 1.2 (Statistically indistinguishable)**.** The *statistical distance* between two distributions on the same set S is

$$\mathrm{d}(\mu_0, \mu_1) = \frac{1}{2}\sum_{s\in S} |\mu_0(s) - \mu_1(s)|.$$

If $\mathrm{d}(\mu_0, \mu_1)$ is negligible in the size of S, then the two distributions are *statistically indistinguishable*.

Often, when breaching a cryptographic protocol, the adversary is successful in distinguishing two probability distribution, and therefore, we have another useful definition of the distance of distributions:

Definition 1.3 (Distinguisher, polynomially indistinguishable)**.** A *distinguisher* $\mathcal{D}$ is a PPT algorithm which given a single input s answers 0 or 1. The *advantage* of $\mathcal{D}$ with respect to two probability distributions μ_0 and μ_1 on S is

$$\mathbf{Adv}^{\text{dist}}_{(\mu_0,\mu_1),\mathcal{D}} = \left|\Pr[\mathcal{D}(s) = 1 \mid s \xleftarrow{\mu_1} S] - \Pr[\mathcal{D}(s) = 1 \mid s \xleftarrow{\mu_0} S]\right|.$$

Two distributions μ_0 and μ_1 are *polynomially indistinguishable* if all PPT distinguishers $\mathcal{D}$ has a negligible advantage (in the size of S).

The distinguisher tries to guess which distribution the random variable s came from. The advantage is 0 if $\mathcal{D}$ does not guess better than an algorithm guessing at random, and the advantage is 1 if $\mathcal{D}$ always guesses correctly.

Intuitively, it does not help having a good distinguisher if the distributions are very similar, and Theorem 1.5 guarantees that the statistical distance is the best any distinguisher can get. To prove this, we need Lemma 1.4, with proof taken from [13].

Lemma 1.4. Let $\mathcal{A}\colon S_0 \to S_1$ be an algorithm (not necessarily polynomially bounded), and let μ_0 and μ_1 be two distributions on S_0, its input. Then

$$\mathrm{d}(\mathcal{A}_{\mu_0}, \mathcal{A}_{\mu_1}) \leq \mathrm{d}(\mu_0, \mu_1).$$

Proof.

$$\begin{aligned}
2\mathrm{d}(\mathcal{A}_{\mu_0}, \mathcal{A}_{\mu_1}) &= \sum_{s\in S_1} |\mathcal{A}_{\mu_0}(s) - \mathcal{A}_{\mu_1}(s)| \\
&= \sum_{s\in S_1} \left|\sum_{t\in S_0} \Pr[\mathcal{A}(t) = s]\mu_0(t) - \sum_{t\in S_0} \Pr[\mathcal{A}(t) = s]\mu_1(t)\right| \\
&\leq \sum_{s\in S_1}\sum_{t\in S_0} \Pr[\mathcal{A}(t) = s]|\mu_0(t) - \mu_1(t)| \\
&= \sum_{t\in S_0}\sum_{s\in S_1} \Pr[\mathcal{A}(t) = s]|\mu_0(t) - \mu_1(t)| \\
&= \sum_{t\in S_0} |\mu_0(t) - \mu_1(t)| \sum_{s\in S_1} \Pr[\mathcal{A}(t) = s] \\
&= 2\mathrm{d}(\mu_0, \mu_1),
\end{aligned}$$

by the generalised triangle inequality, and because s in the very last sum ranges over all possible outputs of $\mathcal{A}(t)$. □

Theorem 1.5. If two distributions μ_0 and μ_1 are statistically indistinguishable, then they are also polynomially indistinguishable.

Proof. We will show that $\mathbf{Adv}^{\text{dist}}_{(\mu_0,\mu_1),\mathcal{D}} = \mathrm{d}(\mathcal{D}_{\mu_0}, \mathcal{D}_{\mu_1})$, and from the lemma above, we see that if the statistical distance is negligible, so must also the advantage of $\mathcal{D}$ be.

First note that since a distinguisher outputs either 0 or 1, we can find another expression for the advantage:

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{dist}}_{(\mu_0,\mu_1),\mathcal{D}} &= \left|\Pr[\mathcal{D}(s)=1 \mid s \xleftarrow{\mu_1} S] - \Pr[\mathcal{D}(s)=1 \mid s \xleftarrow{\mu_0} S]\right| \\
&= \left|\Big(1-\Pr[\mathcal{D}(s)=0 \mid s \xleftarrow{\mu_1} S]\Big) - \Big(1-\Pr[\mathcal{D}(s)=0 \mid s \xleftarrow{\mu_0} S]\Big)\right| \\
&= \left|\Pr[\mathcal{D}(s)=0 \mid s \xleftarrow{\mu_0} S] - \Pr[\mathcal{D}(s)=0 \mid s \xleftarrow{\mu_1} S]\right| \\
&= \left|\Pr[\mathcal{D}(s)=0 \mid s \xleftarrow{\mu_1} S] - \Pr[\mathcal{D}(s)=0 \mid s \xleftarrow{\mu_0} S]\right|.
\end{aligned}$$

By combining the two expressions, and looking at the induced distributions, we see that

$$\begin{aligned}
2\mathbf{Adv}^{\mathrm{dist}}_{(\mu_0,\mu_1),\mathcal{D}} &= \left|\Pr[\mathcal{D}(s)=1 \mid s \xleftarrow{\mu_1} S] - \Pr[\mathcal{D}(s)=1 \mid s \xleftarrow{\mu_0} S]\right| + \\
&\quad \left|\Pr[\mathcal{D}(s)=0 \mid s \xleftarrow{\mu_1} S] - \Pr[\mathcal{D}(s)=0 \mid s \xleftarrow{\mu_0} S]\right| \\
&= |\mathcal{D}_{\mu_1}(1) - \mathcal{D}_{\mu_0}(1)| + |\mathcal{D}_{\mu_1}(0) - \mathcal{D}_{\mu_0}(0)| \\
&= \sum_{b=0,1} |\mathcal{D}_{\mu_0}(b) - \mathcal{D}_{\mu_1}(b)| \\
&= 2\mathrm{d}(\mathcal{D}_{\mu_0}, \mathcal{D}_{\mu_1}),
\end{aligned}$$

and using Lemma 1.4, $\mathrm{d}(\mathcal{D}_{\mu_0}, \mathcal{D}_{\mu_1}) \leq \mathrm{d}(\mu_1, \mu_0)$, so the theorem follows.

□

1.3 Protocol attacks

A main requirement for a cryptographic protocol is that it should be secure. But secure in what sense, and at which level? And secure against what type of attackers? When dealing with interactive protocols, we must look at one participant of the protocol, and see what can happen if there is an attacker or adversary that deliberately disobeys the protocol and tries to learn secrets (or parts thereof) held by the honest participant.

When constructing a protocol, one must balance between making it secure against the most powerful attacks, and making it simple and effective. By imposing too strict requirements on a protocol, there is a possibility that such protocols cannot exist. We discern between the tools an adversary has at its disposal in order to carry out the attack, and at which level the protocol is broken.

In this paper we shall deal with modified versions of digital signatures, which are assumed to be well-known. In order to introduce the concept of attacks and forgeries on signature schemes, we will therefore in this section discuss attacks and forgeries on regular digital signature schemes.

1.3.1 Attacks on digital signature schemes

There are many ways an adversary can try to break a regular digital signature scheme. The different attacks are modelled by the adversary having either a predetermined set of information available, or an *oracle* at his[1] disposal. The oracle mainly comes from a restricted subset of

[1]To avoid the awkward notation "(s)he", all participants in this thesis are referred to as male, but the reader may substitute any appearances of "he", "his", "him" and "himself" with "she", "her" and "herself", if he[1] prefers.

the algorithms that exist in the protocol, and can thus be both interactive and non-interactive. The adversary can query the oracle to gain knowledge, but we can monitor the queries, and he is sometimes restricted in his choice of inputs to the oracle. The oracle also acts as a black box to the adversary, and does not reveal any part of the computation it carries out.

Some well-known attack types are listed in Pointcheval and Stern [22] and from Stinson [25], and they are repeated here, in increasing order of power.

Plain no-message attack In this attack, the adversary cannot interfere in any way with the scheme, but is given a list of previously generated message-signature pairs, which he does not take part in choosing. This attack compares to an adversary listening to an unsecure communication channel, and can also be modelled by an oracle (running with no input) giving out signatures of random messages. This attack is also called known-message attack [25].

Chosen-message attack The adversary can in this attack ask the signer to sign a set of messages, chosen by the adversary. He will then see what the resulting signatures look like. This attack is in [22] further divided into the generic chosen-message attack (where the adversary chooses a list of messages before meeting the signer) and the oriented chosen-message attack (where the adversary first meets the signer and sees his public key, and then produces a list of messages to sign). The chosen-message attack is modelled with an oracle allowing one query with a list of messages the adversary wants to sign. The oracle then replies with the correct signatures of these messages.

Adaptive chosen-message attack This is the most powerful attack, where an adversary asks the oracle to sign messages that are adaptively chosen by the adversary. Adaptively means that the adversary chooses messages one after another, after seeing the result of the previous signing. This attack is modelled by an oracle allowing repeated queries of messages to sign.

If the adversary is allowed to perform several queries to an interactive oracle during one attack, it is also important to separate between how the oracles are to be queried, and Pointcheval and Stern [22] and Hazay et al. [15] list the following type of attacks.

Sequential attack In a sequential attack, the adversary must complete one interaction protocol with an oracle before trying to send another query. It could be advantageous for an adversary to be able to halt an oracle and wait for the result of another query before continuing, but in this attack model, this is prohibited.

Parallel attack In a parallel attack, the adversary is allowed to interact in parallel, with many copies of the oracle at the same time, but must interact with all of them simultaneously, and cannot run one oracle copy more than one step ahead of any other.

Concurrent attack In a concurrent attack, the adversary is allowed to interact with many oracles in an arbitrarily interleaved manner. He can decide to abort the interaction of one oracle, or halt one oracle to wait from the results of another one before continuing. This attack compares to a real life setting where the signer must be able to interact with many users concurrently, without taking into consideration that some of them may be controlled by the same user, or an adversary.

Replay attack This attack is a very powerful attack, in which the adversary is not only allowed to run oracles concurrently, but he is also allowed to "rewind" the oracles by tricking them to redo some computation, often involving random choices. This way, he can make the oracles do choices that are more favourable. Since this attack is mostly considered to be too powerful, being easy to remedy[2], it is often not taken into consideration.

Note that we do not care which attack strategies the adversary chooses, for example whether he uses brute force, utilises the birthday paradox [25], or employs a side channel attack, as an adversary is free to make any computation he sees fit. However, most adversaries are restricted to polynomially many computations and oracle interactions.

1.3.2 Forgeries on digital signature schemes

There are several adversarial goals of an attack on a digital signature scheme. The most common forgeries [22, 25] are classified here, in order from the most devastating to the less serious, therefore also from the weakest to the strongest.

Total break From the attack, the adversary manages to discover the secret key of the signer. In this way the adversary can in all circumstances act as the signer himself. This is the most serious attack, and of course the hardest to accomplish.

Universal forgery Here, the adversary does not directly learn the secret key, but can by other means construct an effective algorithm to create valid signatures on an overwhelming majority of messages. This is in practice equivalent to a total break of the scheme.

Selective forgery In this forgery, the adversary can with non-negligible probability create a signature on a randomly chosen message. The distinction between selective and universal forgery is the size of the set of forgeable messages.

Existential forgery In an existential forgery, the adversary can on his own produce single message-signature pair without querying any oracle with this particular message. The message is chosen by the adversary, but it might be that the actual message is of no use at all. This is the least serious attack, and by far the hardest to protect against, since it means that no forgeable message can exist.

As we see from the lists above, the best a scheme can do is to resist an adaptive chosen-message existential forgery, and it is well known that the plain versions of RSA and ElGamal signature schemes fail under adaptive chosen-message existential forgery [22].

1.4 Provable security

After identifying the the types of attacks of an imaginary adversary, and what he might accomplish, we will now proceed with discussing how to prove that a cryptographic protocol is secure. There are many proof techniques when inspecting the security of a protocol, and we will now present those that seem to be the most popular [15, 16, 26].

[2] Using random numbers (nonces) as session identifiers.

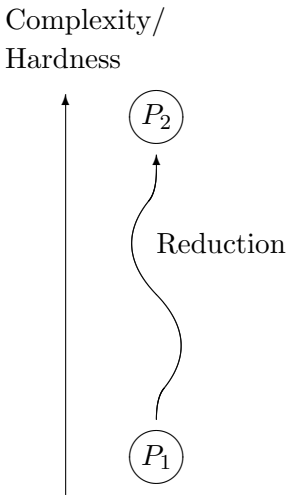

Figure 1.1: A guide for visualising reductions

Random oracle model Many cryptographic protocols [7, 10, 12, 22] use a hash function as part of the scheme. The random oracle model was introduced by Bellare and Rogaway [4], and it assumes that we are given a hash function that behaves like it is truly random [25]. This means that the oracle represents a hash function $H\colon X \to Y$, and if we call upon $H(x)$, the oracle samples $y \xleftarrow{r} Y$ and sets $H(x) = y$. The exception is of course if we have asked for the value of $H(x)$ before, because then the oracle must give us the same y as it gave us then. However, in implementations of cryptographic protocols, we must choose a real hash function, which realistically cannot behave exactly like a random oracle. The security of the protocol then hinges on the proposition that the implemented hash function cannot be distinguished from a real random oracle, which generally is very hard to prove.

Common reference string model There is also a model called the common reference string model, in which all participants has access to the same common reference string. This is a generalisation of the random oracle model, and is in the context of blind and group signature schemes treated further in respectively Fischlin [12] and Trolin [26].

Complexity-based proofs Introduced by Diffie and Hellman, complexity-based proofs give much more general and stronger results [16], since they aim to reduce the security of a given protocol to the underlying fundamental cryptographic problem. Thus only general complexity assumptions like the hardness of factoring or the discrete logarithm problem are building blocks of the proofs.

Since complexity-based proofs of security are preferable because they yield stronger results, we will in this section continue to study reductions, as they lie at the core of complexity-based proofs.

Definition 1.6 (Reduction, polynomial reduction)**.** A *reduction* from a problem P_1 to a problem P_2 is an algorithm that solves P_1 when given access to an oracle that solves P_2 [13, 17]. It is important to measure the reduction in terms of time complexity and number of oracle queries.

If both measurements are polynomial in size of the input (of problem P_1), we say that it is a *polynomial reduction*, and that P_1 *reduces* to P_2. This is written $P_1 \rightsquigarrow P_2$.

If there exists a polynomial reduction from P_1 to P_2, we learn that the problem P_2 is *at least as hard as* P_1 (see Figure 1.1). This is because if P_1 is solvable, then P_2 also is [17]. We cannot say how much harder P_2 is relative to P_1, and there might also exist a reduction from P_2 to P_1, in which case they are *equivalent*.

If we let P_1 be a famous infeasible problem, and P_2 be a security requirement of the protocol we are studying, then we can make a complexity-based proof of the security of the protocol by constructing a reduction from P_1 to P_2. We then see that an effective attack on the security requirement will lead to an effective attack on the main problem itself. But P_1 is assumed to be infeasible, so the conclusion is that P_2 is also infeasible, and that the security requirement must hold.

We are then left with formalising an attacker related to the security requirement, leading to a possible attack on the protocol. This is done by introducing *experiments*, or *games*, which we will introduce in the next section.

1.5 Experiments

Experiments are a common method to formalise security properties [24]. An experiment is a game which we play with the adversary. Throughout the game, the adversary is given information, and the experiment challenges the adversary to make use of this information to produce new information, or to distinguish between two probability distributions. If the adversary with non-negligible probability wins the game, then we say that the adversary can carry out a successful attack against the corresponding security property. We believe the notion of an experiment is best explained by providing an example, and we have chosen to do this by formalising the decision Diffie-Hellman assumption, inspired by Trolin [26].

Definition 1.7 (Decision Diffie-Hellman problem)**.** The *decision Diffie-Hellman problem* (ddh) in the cyclic group G_n (of order n) is to find out, given $g \in G_n$, if $k \in G_n$ is the Diffie-Hellman key g^{ab} of $g_1 = g^a$ and $g_2 = g^b$.

The decision Diffie-Hellman assumption says that this problem is infeasible, or equivalent, that it is infeasible to distinguish between the distributions μ_0 and μ_1 on $G_n \times G_n \times G_n \times G_n$, which are induced by the uniform distribution on $\mathbb{Z}_n \times \mathbb{Z}_n \times \mathbb{Z}_n$ and respectively the functions f_0 and f_1, where

$$\begin{aligned} f_0(a,b,c) &= (g, g^a, g^b, g^{ab}), \quad \text{and} \\ f_1(a,b,c) &= (g, g^a, g^b, g^c), \end{aligned}$$

and g is any fixed element of G_n.

In the setting of experiments we can make a formal definition:

Proposition 1.8 (Decision Diffie-Hellman assumption)**.** Consider the experiment $\mathbf{Exp}^{\text{ddh}}_{G_n,\hat{\mathcal{A}}}(n)$, listed in Experiment 1.1. We define the advantage of the adversary $\hat{\mathcal{A}}$ in this experiment as

$$\mathbf{Adv}^{\text{ddh}}_{G_n,\hat{\mathcal{A}}}(n) = \left|\Pr[d' = 1 \mid d = 1] - \Pr[d' = 1 \mid d = 0]\right|.$$

For all PPT algorithms $\hat{\mathcal{A}}$, this advantage is negligible in n.

Experiment 1.1 $\mathbf{Exp}^{\mathrm{ddh}}_{G_n,\hat{\mathcal{A}}}(n)$:

1. Let $d \xleftarrow{r} \{0,1\}$.
2. Let $g \xleftarrow{r} G_n$, and $a, b, c \xleftarrow{r} \mathbb{Z}_n$.
3. Define $(g_1, g_2) \leftarrow (g^a, g^b)$.
4. If $d = 0$, then $k \leftarrow g^{ab}$, else $k \leftarrow g^c$.
5. Let $d' \leftarrow \hat{\mathcal{A}}(\mathsf{guess}, g, g_1, g_2, k)$.

The notation $\mathbf{Exp}^{\mathrm{ddh}}_{G_n,\hat{\mathcal{A}}}(n)$ of Experiment 1.1 means that we are running an adversary $\hat{\mathcal{A}}$ against a security property or problem ddh concerning the object G_n, and similar notation will be used in the rest of this paper.

Experiment 1.1 is quite simple, because after some analysis, we see that before Step 5, (g, g_1, g_2, k) is sampled according to the distribution μ_d, where d is the *decision bit*. We then ask the adversary to distinguish between these two distributions by trying to guess d. Comparing the advantage $\mathbf{Adv}^{\mathrm{ddh}}_{G_n,\hat{\mathcal{A}}}$ to the advantage $\mathbf{Adv}^{\mathrm{dist}}_{(\mu_0,\mu_1),\hat{\mathcal{A}}}$ of $\hat{\mathcal{A}}$ appearing as a distinguisher in Section 1.2, we see that they both are equal to $\mathrm{d}(\hat{\mathcal{A}}_{\mu_0}, \hat{\mathcal{A}}_{\mu_1})$, so this experiment comes from a standard method of formalising a distinguisher.

Experiments can be made more complex by adding more communications with the adversary. If this is the case, we implicitly assume that the adversary is allowed to maintain state information from one call to the next.

We can also give the adversary access to more information via some oracle. For instance, let $\mathcal{DL}_{G_n}$ be an oracle solving the discrete log problem (dlog) in the group G_n, that is $\mathcal{DL}_{G_n}(g, g^a) = a$ for all $g \in G_n$, $a \in \mathbb{Z}_n$. We can modify the experiment and give $\hat{\mathcal{A}}$ access to this oracle. We specify this by replacing Step 5 with $d' \leftarrow \hat{\mathcal{A}}^{\mathcal{DL}_{G_n}(\cdot,\cdot)}(\mathsf{guess}, g, g_1, g_2, k)$. We use the notation that all oracles available to an adversary are listed as a superscript, and the dot notation shows which inputs can be chosen freely by the adversary.

With the aid of this oracle, an adversary can readily solve ddh, because we can construct an adversary $\hat{\mathcal{A}}$ presented in Algorithm 1.2, having advantage $\mathbf{Adv}^{\mathrm{ddh}}_{G_n,\hat{\mathcal{A}}}(n) = 1$, which is certainly not negligible. It uses the power of the oracle to compute the discrete logarithms of its inputs.

Algorithm 1.2 $\hat{\mathcal{A}}^{\mathcal{DL}_{G_n}(g,\cdot)}(\mathsf{guess}, g, g_1, g_2, k)$:

1. Let $a \leftarrow \mathcal{DL}_{G_n}(g, g_1)$,
   $b \leftarrow \mathcal{DL}_{G_n}(g, g_2)$, and
   $c \leftarrow \mathcal{DL}_{G_n}(g, k)$.
2. If $ab \equiv c \pmod{n-1}$, then return 0, else return 1.

Note that by this discussion, we have reconstructed the well-known polynomial reduction from the discrete log problem to the decision Diffie-Hellman problem, and have shown that $\mathsf{dlog} \rightsquigarrow \mathsf{ddh}$.

Chapter 2

Blind signatures

Blind signatures were introduced by Chaum [9] in 1982, and has been an invaluable tool in order to design digital cash systems. A blind signature is like a regular signature, in the way that a signer is asked to create a signature on a message or a document. The difference is that the actual message is to be kept secret from the signer. The blind signature is therefore often compared to the user putting a message in an envelope, letting the signer print his signature on the envelope, such that the signature is copied through to the message hidden inside.

The main properties of blind signatures is that a user can ask a signer to produce an unforgeable signature of a document (non-forgeability), in such a way that the signer cannot distinguish between signatures he has issued (blindness).

Because of the nature of the blindness property, the signer cannot be allowed to see the complete document he is signing. In the different schemes that have been constructed, there are thus different ways of making the user guarantee that the document has some specific structure, or includes some specific information. This is an important issue, since few signers would want to sign an arbitrary document. In the first construction by Chaum, where the blind signature is a vital part of his on-line digital cash system [10], the signature itself (as opposed to the document) is the valuable information, and the user can freely choose any message to use as a coin. In the off-line variant presented in the same paper, the user creates several coin candidates, and the signer asks him to reveal some, usually half, of them. This is called a *cut-and-choose* method [22], since if the signer can verify the correctness of the opened candidates, then he can with some given degree trust the rest of the candidates to be correct, and send his signature back to the user. In Brands' off-line cash system [7], a *restrictive blind signature* is used, where the setup of the system guarantees that if a user chooses to generate an invalid document, he cannot make use of the signature at all. The last method is superior to the others, because a dishonest user cannot cheat the signer at all, and the communication between the participants is greatly reduced. We also refer to O discussion regarding *partially blind signatures* [20].

We do not discuss the content of the messages further in this thesis, as many implementations of blind signatures have shown that they are of practical use, both in payment systems and in identification schemes [6, 7]. We will now concentrate on the formal definitions and security requirements of blind signature schemes.

2.1 Definition

It took a very long time from Chaum invented the concept of a blind digital signature until a strict mathematical definition appeared. The first mathematical definition of a blind signature scheme were proposed by Juels, Luby and Ostrovsky in 1997 [16], and in recent years, the definition of such a scheme has more or less been agreed upon. We take the latest definitions from Hazay et al. [15] and Buan, Gjøsteen and Kråkmo [8].

Definition 2.1 (Blind signature scheme)**.** A *blind signature scheme* $\mathcal{BS}$ consists of four polynomial time algorithms (Gen, $\mathcal{S}$, $\mathcal{U}$, Vrfy):

- The PPT *key generation algorithm* Gen takes 1^k as input, where k is the security parameter and 1^k is a string of length k. It outputs a key pair $(\mathsf{PK}, \mathsf{SK})$ of matching public and secret (or private) keys. The security parameter is implicit in both of these keys.

- The two interactive and PPT algorithms *signer* $\mathcal{S}$ and *user* $\mathcal{U}$ run a *signing protocol* to collaboratively create a blind signature σ on a message m, chosen by $\mathcal{U}$. Both are given PK as input, and the signer is given SK as additional input. The resulting signature is known to $\mathcal{U}$ only, and we write the execution of this protocol as $\sigma \leftarrow \langle \mathcal{S}, \mathcal{U}(m) \rangle$ (with the keys as implicit input). The two algorithms stop after a polynomial (in k) number of interactions, and $\mathcal{S}$ outputs either Accept or Reject, while $\mathcal{U}$ outputs either the signature σ or $\perp$, signalling a failure.

- The DPT *verification algorithm* Vrfy takes a message m, a purported signature σ and the public key PK as input, and outputs Accept if the signature is valid, else it outputs Reject.

Some papers (for example [15]) regard the joint execution of $\mathcal{S}$ and $\mathcal{U}$ as one *signing algorithm* $\sigma = \mathsf{Sign}(\mathsf{PK}, \mathsf{SK}, m)$, but we believe this does not clearly show the different objectives of the involved signer and user, and the possibility for an adversary to interfere with this algorithm.

A scheme is useless without a correctness statement, which should state what we want the scheme to achieve. In most cases, the goal of a scheme is obvious, but the correctness of a blind signature scheme is mathematically precise from the following definition.

Definition 2.2 (Correctness)**.** A blind signature scheme (Gen, $\mathcal{S}$, $\mathcal{U}$, Vrfy) is *correct* if for all $(\mathsf{PK}, \mathsf{SK})$ output by $\mathsf{Gen}(1^k)$, for all $\mathcal{S}$ and $\mathcal{U}$ following the protocol, and for all m, $\mathcal{S}$ outputs Accept, $\mathcal{U}(m)$ outputs a signature $\sigma \neq \perp$, and $\mathsf{Vrfy}(\mathsf{PK}, m, \langle \mathcal{S}, \mathcal{U}(m) \rangle) = \mathsf{Accept}$.

The correctness property says that if both parties follow the instructions of the signing protocol, then the produced signature must be valid. Note that this does not exclude trivial signature schemes, for example in which all signatures are equal, or all signatures are valid. But these schemes are clearly not useful, and as with all cryptographic protocols, we are more interested in that the scheme is secure, even if one of the parties is dishonest. This is treated in the next section.

The message, signature and key spaces are not very important. Since a message, signature or key always can be encoded in binary, the spaces are often assumed to be subspaces of the binary strings $\{0,1\}^*$. There is also a note in [15] concerning the lengths of a signature, which we will repeat and comment here.

Definition 2.3. (Length-regular) A blind signature scheme $(\mathsf{Gen}, \mathcal{S}, \mathcal{U}, \mathsf{Vrfy})$ is *length-regular* if there exists a polynomial p such that for all $(\mathsf{PK}, \mathsf{SK})$ output by $\mathsf{Gen}(1^k)$, for all $\mathcal{S}$ and $\mathcal{U}$ following the protocol, and for all m, if $\sigma \leftarrow \langle \mathcal{S}, \mathcal{U}(m) \rangle$, then $|\sigma| \leq p(|m|)$, and if $|\sigma| > p(|m|)$, then $\mathsf{Vrfy}(\mathsf{PK}, m, \sigma) = \mathsf{Reject}$.

Since both $\mathcal{S}$ and $\mathcal{U}$ are polynomially bounded, they cannot produce output that are more than polynomial in its input size, so the first property is automatically satisfied. The last property says that all signatures that are too large must be invalid.

From now on, we assume that all blind signature schemes are both correct and length-regular.

2.2 Security requirements

There are two parties in a blind digital signature scheme, and each of them have the opportunity to break the scheme. The user would of course want to produce valid signatures on his own, without interacting with the signer. The signer would on his side like to mark or put information in the signatures in such a way that he would later recognise the signature and identify the user.

The first property has been called unforgeability [15], non-forgeability [16] and one-more-forgery [22], while the last has been presented in different forms and called both blindness [15, 16] and untraceability [10]. It is not immediate what blindness should contain, since there are some subtle points that must be resolved. Because of this, there has also been the notion of unlinkability [7, 22], which says that the signer cannot link two signatures to the same (but still anonymous) user. However, the current definition of blindness also incorporates the notion of unlinkability, which therefore is obsolete.

The formal definitions of the security requirements of blind signature schemes were first presented by Juels, Luby and Ostrovsky [16], and they have since remained standard, only with slight variations.

2.2.1 Blindness

The blindness property says that the signer should not learn anything about the messages he signs (that he did not know a priori). But referring to the discussion above, he should also be unable to put information in the signature that would make him able to recognise it later. The ability to recognise a signature later can be reduced to discerning between two single signatures, so Juels, Luby and Ostrovsky [16] therefore formalised the chosen-key-and-messages attack, which is a very strong attack, and this has become the standard definition of blindness.

In this attack, the adversary chooses a public key representing his identity, and two messages that a user would want a signature on. The adversary then interacts with two copies of the user, without knowing which user gets which message. When given the two signatures after the two interactions has completed, the adversary should not have any non-negligible chance of guessing which signature belongs to which message.

Definition 2.4 (Blindness)**.** Consider the experiment $\mathbf{Exp}^{\text{blind}}_{\mathcal{BS},\hat{\mathcal{S}}}(k)$, listed in Experiment 2.1, where $\hat{\mathcal{S}}$ is any algorithm (the adversary). We define the advantage of $\hat{\mathcal{S}}$ in breaking $\mathcal{BS}$ with respect to *blindness* as

$$\mathbf{Adv}^{\text{blind}}_{\mathcal{BS},\hat{\mathcal{S}}}(k) = \left| \Pr[b' = 1 \mid b = 1] - \Pr[b' = 1 \mid b = 0] \right|,$$

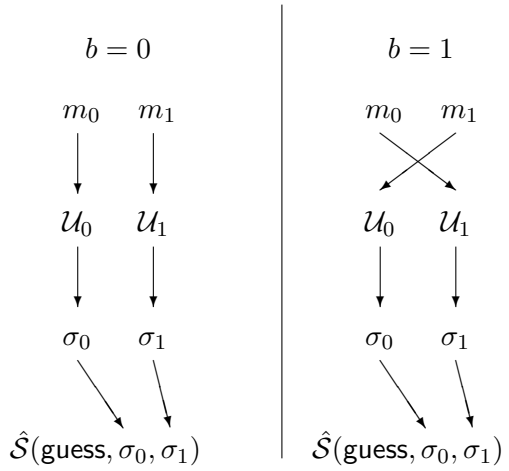

Figure 2.1: The two scenarios of Experiment 2.1

where the probability is taken over all random choices done by all algorithms in the experiment. The scheme $\mathcal{BS}$ is *secure with respect to blindness* if for all PPT algorithms $\hat{S}$, $\mathbf{Adv}^{\text{blind}}_{\mathcal{BS},\hat{\mathcal{S}}}(k)$ is negligible in k.

Experiment 2.1 $\mathbf{Exp}^{\text{blind}}_{\mathcal{BS},\hat{\mathcal{S}}}(k)$:

1. Let $(\mathsf{PK}, m_0, m_1) \leftarrow \hat{\mathcal{S}}(\mathsf{gen}, 1^k)$. The two messages must be of equal size.
2. Let $b \xleftarrow{r} \{0,1\}$.
3. Let $\hat{\mathcal{S}}$ engage in two parallel interactive protocols, the first with $\mathcal{U}_0 = \mathcal{U}(m_b)$, and the second with $\mathcal{U}_1 = \mathcal{U}(m_{1-b})$.
4. For $i = 0, 1$, let $\sigma_i \leftarrow \langle \hat{\mathcal{S}}, \mathcal{U}_i \rangle$, but set $(\sigma_0, \sigma_1) \leftarrow (\perp, \perp)$ if either U_0 or U_1 aborts or outputs $\perp$.
5. Let $b' \leftarrow \hat{\mathcal{S}}(\mathsf{guess}, \sigma_0, \sigma_1)$.

We put a hat on the signer $\hat{\mathcal{S}}$ to remember that he is an adversary that is not restricted to follow any particular protocol. We see that the signer chooses which messages to sign, but the (uniformly) random bit b decides which of the two instances of the user $\mathcal{U}_0$ and $\mathcal{U}_1$ gets which message (see Figure 2.1). Also note that the adversary does not have access to the Vrfy algorithm, or else he could just discover which message σ_0 is a valid signature of by calling $\mathsf{Vrfy}(\mathsf{PK}, m_0, \sigma_0)$.

There is naturally no restriction on the behaviour of an adversary, and specifically we cannot restrict it to obey the signing protocol, and therefore he can try to trick and confuse the signer, such that it outputs $\perp$. The special case in Step 4 concerns the case where the adversary deliberately aborts the communication with one of the users.

We see that the adversary $\hat{\mathcal{S}}$ tries to distinguish between the probability distributions $\langle \hat{\mathcal{S}}, \mathcal{U} \rangle_{(m_0)}$ and $\langle \hat{\mathcal{S}}, \mathcal{U} \rangle_{(m_1)}$ (abusing the notation of Section 1.2), and that the definition of advantage is equivalent to the distinguisher of ddh in Section 1.5. We can therefore conclude that

an equivalent definition of blindness is that for all messages m_0, m_1, all public keys PK, and all PPT algorithms $\hat{\mathcal{S}}$, the output distributions of $\langle\hat{\mathcal{S}},\mathcal{U}(m_0)\rangle$ and $\langle\hat{\mathcal{S}},\mathcal{U}(m_1)\rangle$ are polynomially indistinguishable.

Buan, Gjøsteen and Kråkmo [8] call this particular version of blindness *strong*, and reserves regular blindness for the case where the adversary is in Step 1 required to output not only the public key, but a public and secret key pair. This might make a difference in cryptosystems in which not all values that appear as a public key necessarily have a corresponding private key. They also introduce a concept called *weak blindness*, where the adversary is restricted to keys that are genuinely generated (by Gen). If a scheme is designed where some public keys are weak and make the recognition of signatures easy, but the overwhelming majority of the public keys fulfils the blindness property, then this scheme has weak, but not regular blindness. (The adversary will in Experiment 2.1 of course pick one of the weak keys, but the Gen algorithm will generate these keys with negligible probability.)

2.2.2 Non-forgeability

The non-forgeability property is a guarantee that the user cannot create signatures on his own. There is no concept directly equivalent to the existential forgery of regular signatures, because the blinding makes it difficult to check which messages the users asks for a signature of [15]. The only way we can restrict a malicious user is to set a bound on the number of executions he is allowed to perform with the signer.

The correct way to specify the non-forgeability requirement is therefore to prohibit the user from doing a *one-more forgery*. This requirement states that an adversary cannot obtain $l+1$ signatures after having only l (or less) interactions with a signer [22]. In this way, we are sure that a user of a digital cash system cannot produce valid coins by other means than contacting the bank to get a blindly signed coin [7].

As discussed in Section 1.3 concerning multiple interactions, we must distinguish between whether the adversary must interact with the signer in a sequential or parallel manner, or if the adversary has the power to interact in an arbitrarily interleaved (concurrent) manner. We would hope to provide protection against the most general attacker, and in the current setting, it is also reasonable to assume that a signer could engage in many signing interactions simultaneously, without making the users wait for each other. The current definition of non-forgeability therefore concerns a concurrent, adaptive chosen-message attack in order to try to produce a one-more forgery [8, 15].

Definition 2.5 (Non-forgeability)**.** Consider the experiment $\mathbf{Exp}^{\text{nonf}}_{\mathcal{BS},\hat{\mathcal{U}}}(k)$, listed in Experiment 2.2, where $\hat{\mathcal{U}}$ is any algorithm. We define the advantage of $\hat{\mathcal{U}}$ in breaking $\mathcal{BS}$ with respect to *non-forgeability* as

$$\mathbf{Adv}^{\text{nonf}}_{\mathcal{BS},\hat{\mathcal{U}}}(k) = \Pr[j > l],$$

where the probability as usual is taken over all random choices done by all algorithms included. The scheme $\mathcal{BS}$ is *non-forgeable* if for all PPT algorithms $\hat{\mathcal{U}}$, $\mathbf{Adv}^{\text{nonf}}_{\mathcal{BS},\hat{\mathcal{U}}}(k)$ is negligible in k.

In this experiment, the key generation algorithm generates a key pair that is used throughout the experiment. The adversary $\hat{\mathcal{U}}$ is then given oracle access to the signer (with the secret key SK implicitly given to $\mathcal{S}$, but hidden from $\hat{\mathcal{U}}$). He can then concurrently obtain signatures from the signer, on any message he chooses. Again, we cannot deny the adversary trying to trick the signer by disobeying or aborting the protocol, so we let l denote the number of signatures that

Experiment 2.2 $\mathbf{Exp}^{\text{nonf}}_{\mathcal{BS},\hat{\mathcal{U}}}(k)$:

1. Let $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{Gen}(1^k)$.
2. Let $\Sigma' \leftarrow \hat{\mathcal{U}}^{\mathcal{S}}(\mathsf{PK})$, simulating that $\hat{\mathcal{U}}$ can engage in polynomially many (in k) arbitrarily interleaved interactive protocols with $\mathcal{S}$. Let l be the number of executions where $\mathcal{S}$ outputs Accept.

   $\Sigma' = \{(m_1, \sigma_1), \ldots, (m_{j'}, \sigma_{j'})\}$ should be a set of unique message-signature pairs of size j'.
3. Discard any invalid signatures, so let

$$\Sigma = \{(m_i, \sigma_i) \in \Sigma' \mid \mathsf{Vrfy}(m_i, \sigma_i) = \mathsf{Accept}\},$$

   and define $j = |\Sigma|$.

S agrees to have issued, and forget about the other interactions that might have been started, but were aborted or otherwise deemed invalid by the signer.

After the user is satisfied with his achieved signatures, he is asked to output a list of message-signature pairs. We must of course exclude invalid and duplicate signatures, so in Step 4, we filter the pairs. By the discussion above, the blind nature of the signature makes it impossible to tell whether the final messages were actually the messages the signer was asked to sign, and that only j, the number of such valid signatures, are of interest.

If an adversary has a non-negligible chance of obtaining more unique signatures than the signer has issued ($j > l$), the adversary is successful in against breaking the non-forgeability of the blind signature scheme.

Note that this experiment is different from the experiment of the blindness requirement, because this is not a distinguishing experiment. This is merely an experiment challenging the adversary to make use of the information available to him in order to gain new information that is meant to be hidden. The winning chances of an adversary is measured in the success rate, which is the probability that a certain unwanted event occurs.

Fischlin [12] also mentions a weak non-forgeability requirement, which requires the messages $m_1, \ldots, m_{j'}$ to be distinct (opposed to unique message-signature pairs), such that an adversary cannot try to produce more than one signature on any single message.

2.2.3 Summary

We have now identified the two security requirements that exist in today's literature regarding blind signature schemes, and we conclude by combining these two into one definition of the security of a blind signature scheme.

Definition 2.6 (Secure blind signature scheme)**.** If a blind signature scheme is both non-forgeable and secure with respect to blindness, then it is *secure*.

This means that we do not need to consider other security notions when analysing the security of a blind signature scheme. As we have mentioned in the discussions of the preceding subsections, the security definitions here are the strongest we have encountered, and thus it is possible to weaken some of the definitions and still regard the scheme as being secure enough for practical considerations.

2.3 Two blind signature schemes

We will in this section describe two blind signature schemes, namely the blind Schnorr signature scheme and the blind Okamoto-Schnorr signature scheme, both first presented by Okamoto [19]. The blind Schnorr scheme is an extension of the regular Schnorr signature scheme [23], and Okamoto adapts this by using a divertible zero-knowledge interactive proof [21] to introduce a blinding factor. The blind Okamoto-Schnorr signature scheme is an equivalent extension of the regular Okamoto-Schnorr signature scheme [19].

The two schemes will be put into the theoretical framework of the last section, and we will prove that they both are secure with respect to blindness. We will also outline a proof by Pointcheval and Stern [22], who show that the Okamoto-Schnorr scheme is non-forgeable in the random oracle model under the discrete log assumption. The discrete log assumption states that the discrete log problem dlog is infeasible [18, 25].

First we need to introduce some notation: We will encounter products like $\prod_{i=1}^{n} g_i^{a_i}$. For brevity, we write this as $\boldsymbol{g}^{\boldsymbol{a}}$, and this notation is used whenever $\boldsymbol{g} = (g_1, \ldots, g_n)$ and $\boldsymbol{a} = (a_1, \ldots, a_n)$ are tuples of the same length. We also treat component-wise addition and scalar multiplication of tuples, such that if c is a scalar, and $\boldsymbol{a}$ and $\boldsymbol{b}$ are tuples of corresponding lengths, then $\boldsymbol{g}^{\boldsymbol{a}+\boldsymbol{b}c} = \prod_{i=1}^{n} g_i^{a_i+b_i c}$.

Both of these schemes are described in Pointcheval and Stern [22], and we will follow their presentation, except that we would like to stress the strong similarities between the two schemes by rearranging the notation. They are therefore presented together in Algorithms 2.3–2.6, where the Schnorr scheme corresponds to $n = 1$, and the Okamoto-Schnorr scheme corresponds to $n = 2$.

The schemes use a group $\mathbb{Z}_q$ of large prime order, and they use generators of subgroups of order p as building blocks. In the setup phase, such a group is found by choosing primes p and q, such that $q \mid p - 1$, and $|p|, |q| = \mathcal{O}(k)$, where k is the security parameter[1]. It follows from the discrete log assumption that computing discrete logarithms in this group is infeasible [7]. The scheme also uses a hash function $H\colon M \times \mathbb{Z}_q \to \mathbb{Z}_q$, where M is the message space. We do not specify the hash function in more detail, since we choose to work in the random oracle model [22].

Algorithm 2.3 $\mathsf{Gen}(1^k)$, the key generation algorithm of the (Okamoto-)Schnorr scheme:

1. Choose n elements $g_i \in \mathbb{Z}_p$ $(i = 1, \ldots, n)$, all of order q, and define $\boldsymbol{g} = (g_1, \ldots, g_n)$.
2. Choose $\boldsymbol{x} \xleftarrow{r} \mathbb{Z}_q^n$, and let $y = \boldsymbol{g}^{\boldsymbol{x}} \pmod p$.
3. Output $(\mathsf{PK}, \mathsf{SK}) = ((\boldsymbol{g}, y), \boldsymbol{x})$.

The key generation algorithm finds n elements $g_1, \ldots, g_n$ of order q. The secret (or private) key consists of $\boldsymbol{g}$ and the exponents $x_1, \ldots, x_n$, which are used to form the public key $y = \boldsymbol{g}^{\boldsymbol{x}}$. We call $\boldsymbol{x}$ the *representation of y with respect to $\boldsymbol{g}$*, and Brands [7] shows that finding representations[2] is equivalent to the discrete log problem. This is formalised as the *representation problem*.

[1] Okamoto [19] suggests $q > 2^{140}$ and $p > 2^{512}$.

[2] Brands [7] only applies the definition when $n \geq 2$, because if $n = 1$, then x_1 is of course just the g_1-logarithm of y. We have however chosen to generalise the definition to include the case $n = 1$.

Remember that the Schnorr scheme arises when $n = 1$, and thus the private key is uniquely defined from the public key. The discrete log assumption however guarantees that the private key is infeasible to find. The Okamoto-Schnorr scheme uses two generators g_1 and g_2 and therefore has q possible private keys corresponding to any public key. Brands uses a variation of the blind Okamoto-Schnorr signature scheme in his off-line cash system, so he studies this property further in [7], along with a discussion on the representation problem.

The signing interaction of the two schemes is presented in Algorithms 2.4 and 2.5, where the signer $\mathcal{S}$ and the user $\mathcal{U}$ cooperate to create a signature $(e, \boldsymbol{s})$ on a message m.

Algorithm 2.4 $\mathcal{S}$, the signing algorithm of the (Okamoto-)Schnorr scheme:

1. Choose $\boldsymbol{k} \stackrel{r}{\leftarrow} \mathbb{Z}_q^n$ and let $\tilde{r} \leftarrow \boldsymbol{g}^{\boldsymbol{k}} \pmod p$.
2. Send $\tilde{r}$ to $\mathcal{U}$.
3. Receive $\tilde{e}$ from $\mathcal{U}$.
4. Calculate $\tilde{\boldsymbol{s}} \leftarrow \boldsymbol{k} + \boldsymbol{x}\tilde{e} \pmod q$ and send $\tilde{\boldsymbol{s}}$ back to $\mathcal{U}$.
5. Output Accept.

Algorithm 2.5 $\mathcal{U}(m)$, the user algorithm of the (Okamoto-)Schnorr scheme:

1. Choose $\boldsymbol{a} \stackrel{r}{\leftarrow} \mathbb{Z}_q^n$ and $b \stackrel{r}{\leftarrow} \mathbb{Z}_q$.
2. Receive $\tilde{r}$ from $\mathcal{S}$.
3. Compute $r \leftarrow \tilde{r}\boldsymbol{g}^{-\boldsymbol{a}}y^{-b} \pmod p$, $e \leftarrow H(m, r)$ and $\tilde{e} \leftarrow e + b \pmod q$.
4. Send $\tilde{e}$ to $\mathcal{S}$.
5. Receive $\tilde{\boldsymbol{s}}$ from $\mathcal{S}$, and check if $\boldsymbol{g}^{\tilde{\boldsymbol{s}}}y^{\tilde{e}} = \tilde{r} \pmod p$. If not, output $\perp$.
6. Let $\boldsymbol{s} = \tilde{\boldsymbol{s}} - \boldsymbol{a} \pmod q$.
7. Output the signature $\sigma \leftarrow (e, \boldsymbol{s})$.

After an initial request from the user, the signer starts by choosing n random exponents $k_1, \ldots, k_n$, and *commits* to these by sending $\tilde{r} = \boldsymbol{g}^{\boldsymbol{k}}$ to the user. In this way, the signer knows a representation of $\tilde{r}$ (with respect to $\boldsymbol{g}$). The user has on his side chosen blinding factors $\boldsymbol{a}$ and b, and computes e, which is the first part of the final signature. This is a hash of the message m being signed, and r, the unblinded $\tilde{r}$. To receive the last part of the signature, $\mathcal{U}$ must compute a *challenge* $\tilde{e}$, which the signer only can correctly answer if he knows a representation of $\tilde{r}$. The *answer* $\tilde{\boldsymbol{s}}$ to this challenge is then verified in Step 5, and in Step 6 unblinded into $\boldsymbol{s}$, the second part of the signature. The challenge and answer interaction is actually a witness-hiding proof of knowledge of a representation [7, 18].

The pair $\sigma = (e, \boldsymbol{s})$ becomes the signature of the message m, which can be easily verified by the verification algorithm described in Algorithm 2.6. This verification is exactly the

same as in the regular (Okamoto-)Schnorr signature scheme [22], so σ is actually also a valid (Okamoto-)Schnorr signature on m.

Algorithm 2.6 $\mathsf{Vrfy}(m, (e, \boldsymbol{s}))$, the verification algorithm of the (Okamoto-)Schnorr scheme:

1. Compute $r' \leftarrow \boldsymbol{g}^{\boldsymbol{s}} y^e \pmod p$.
2. Output Accept if $e = H(m, r')$, else output Reject.

Theorem 2.7. The blind Schnorr and Okamoto-Schnorr signature schemes are correct.

Proof. Assume that $\mathcal{S}$ and $\mathcal{U}$ was run with keys that were generated by Gen, and that they both followed the protocol to create a signature $(e, \boldsymbol{s})$ on a message m. Then, since

$$\boldsymbol{g}^{\tilde{\boldsymbol{s}}} y^{\tilde{e}} = \boldsymbol{g}^{\boldsymbol{k}+\boldsymbol{x}\tilde{e}} \boldsymbol{g}^{-\boldsymbol{x}\tilde{e}} = \boldsymbol{g}^{\boldsymbol{k}} = \tilde{r} \pmod p,$$

the user does not output $\perp$ in Step 4 of Algorithm 2.5. Also, since

$$\boldsymbol{g}^{\boldsymbol{s}} y^e = \boldsymbol{g}^{\tilde{\boldsymbol{s}}-\boldsymbol{a}} y^{\tilde{e}-b} = \boldsymbol{g}^{\boldsymbol{k}+\boldsymbol{x}\tilde{e}-\boldsymbol{a}-\boldsymbol{x}\tilde{e}+\boldsymbol{x}b} = \boldsymbol{g}^{\boldsymbol{k}-\boldsymbol{a}} y^{-b} = \tilde{r}\boldsymbol{g}^{-\boldsymbol{a}} y^{-b} = r \pmod p,$$

then $H(m, r') = H(m, \boldsymbol{g}^{\boldsymbol{s}} y^e) = H(m, r) = e$, so Vrfy accepts in Algorithm 2.6. The schemes are therefore correct. □

Theorem 2.8. The blind Schnorr ($\mathcal{S}$) and Okamoto-Schnorr ($\mathcal{OS}$) signature schemes are secure with respect to blindness.

Proof. Let $\hat{\mathcal{S}}$ be an adversary of Experiment 2.1, and let μ_0 and μ_1 be the output distributions of $\langle \hat{\mathcal{S}}, \mathcal{U}(m_0) \rangle$ and $\langle \hat{\mathcal{S}}, \mathcal{U}(m_1) \rangle$. The adversary is challenged to distinguish between μ_0 and μ_1, knowing the values m, $\boldsymbol{k}$, $\tilde{r}$, $\tilde{e}$ and $\tilde{\boldsymbol{s}}$ of both interactions (but not knowing which m corresponds to which user interaction). The only value $\hat{\mathcal{S}}$ receives from $\mathcal{U}$ is $\tilde{e}$, but in Step 3 of Algorithm 2.5, this value is blinded using b, which is uniformly distributed on $\mathbb{Z}_q$. Hence, $\tilde{e}$ contains no information about neither $\tilde{r}$ nor m, which are the values that are chosen by the adversary.

We will now prove that the output of any (completed) signing interaction is statistically indistinguishable from the uniform distribution on $\mathbb{Z}_q \times \mathbb{Z}_q^n$, and it then follows that μ_0 and μ_1 are statistically indistinguishable, and using Theorem 1.5 also polynomially indistinguishable. The advantage $\mathbf{Adv}^{\text{blind}}_{\mathcal{S}/\mathcal{OS},\hat{\mathcal{S}}}(k)$ is therefore negligible for all polynomial adversary $\hat{\mathcal{S}}$, and the blind Schnorr and Okamoto-Schnorr schemes must be blind.

Let $(e, \boldsymbol{s}) \leftarrow \langle \hat{\mathcal{S}}, \mathcal{U}(m) \rangle$ be the output of any completed interaction between $\hat{\mathcal{S}}$ and $\mathcal{U}$, with the public key chosen by the adversary. The last part $\boldsymbol{s}$ is uniformly distributed on $\mathbb{Z}_q^n$, because in Step 6 of Algorithm 2.5, this tuple is computed from $\tilde{\boldsymbol{s}}$ and $\boldsymbol{a}$, where $\boldsymbol{a}$ is by an honest user uniformly chosen (in Step 1 of the same algorithm). Any information that the adversary possibly has hidden in $\tilde{\boldsymbol{s}}$ is therefore effectively obscured. The first part e is the result of applying the hash function H. In the random oracle model, the output value of H is also uniformly distributed. The conclusion is that $(e, \boldsymbol{s})$ is uniformly distributed on $\mathbb{Z}_q \times \mathbb{Z}_q^n$, and μ_0 and μ_1 are therefore both equal to the uniform distribution.

□

As mentioned before, the crucial difference between the two schemes is the uniquely defined private key of the Schnorr scheme. Because of this characteristic, there does not seem to exist proofs for the non-forgeability of the Schnorr scheme, so we will now turn to Pointcheval and Stern's treatment on the non-forgeability of the Okamoto-Schnorr scheme. This proof reduces the discrete log problem to the problem of forging the Okamoto-Schnorr signature.

Theorem 2.9. The blind Okamoto-Schnorr signature scheme ($\mathcal{OS}$) is non-forgeable.

Proof. We will only sketch the proof of Pointcheval and Stern [22], and refer to their paper for details.

Assume that the scheme is forgeable, then there exists a PPT adversary $\hat{\mathcal{A}}$ having non-negligible chance of success ε in Experiment 2.2. We will build a polynomial algorithm $\hat{\mathcal{B}}$ that uses this adversary as a subroutine to extract the discrete logarithm of g_2 with respect to g_1, where $(g_1, g_2) = \boldsymbol{g}$. Recall that g_1 and g_2 are generators defined in the public key, and that in an attack, the public key is freely chosen by the adversary. Thus in the group $\mathbb{Z}_q$, any discrete logarithm can be computed by $\hat{\mathcal{B}}$, and since this algorithm violates the discrete log assumption, we conclude that $\mathcal{OS}$ is non-forgeable.

Whenever $\hat{\mathcal{A}}$ successfully does a forgery of the blind Okamoto-Schnorr signature scheme, let l, as before, denote the number of interactions with the signer. Since the adversary outputs more than l valid signatures, let $(m_i, (e_i, \boldsymbol{s}_i))$, $i = 1, \ldots, l+1$ be the first $l+1$ of them. Also, let T be the running time of $\hat{\mathcal{A}}$, and let Q be the number of queries to the hash function.

A critical observation is that since we assume that the hash function H behaves as a random oracle, we know that $\hat{\mathcal{A}}$ cannot exploit the behaviour of this function. The adversary will therefore be just as successful in the attack if we supply him with any other random-looking function. We also note that for $\mathsf{Vrfy}(m_i, (e_i, \boldsymbol{s}_i))$ to accept in Step 3 of the experiment, $\hat{\mathcal{A}}$ must with high probability have asked for the value $H(m_i, r_i)$ during the attack, for some value r_i. If not, he has correctly guessed one of the q equally likely outputs. We can therefore assume that $H(m_i, r_i)$ in fact was asked during the interaction, and let j_i be the identifier of this query, where the identifiers run from 1 through Q, ordered by the time of the query.

The adversary $\hat{\mathcal{B}}$ is defined in Algorithm 2.7, and after Step 1, with probability greater than $1/2$, $\hat{\mathcal{A}}$ has succeeded in doing a forgery, so $\mathsf{Vrfy}(m_i, (e_i, \boldsymbol{s}_i)) = \mathsf{Accept}$, meaning $r_i = \boldsymbol{g}^{\boldsymbol{s}_i} y_i^e = \boldsymbol{g}^{\boldsymbol{s}_i - \boldsymbol{x}e_i} \pmod p$ for all $i = 1, \ldots, l+1$.

After this success, $\hat{\mathcal{B}}$ tries to obtain a different representation of the same r_i by replaying the attack. Since $\hat{\mathcal{B}}$ controls all (including the random) input to $\hat{\mathcal{A}}$, the attack will proceed exactly as before (including the choice of m_i and r_i), up to the point where $\hat{\mathcal{A}}$ asks for $H(m_i, r_i)$. We then instruct the oracle to return fresh answers, hoping to provide $\hat{\mathcal{B}}$ with a different representation of r_i.

Pointcheval and Stern formally prove that with non-negligible probability, for some i, $\hat{\mathcal{B}}$ learns another representation of r_i, namely $r_i = \boldsymbol{g}^{\boldsymbol{s}'_i} y^{e'_i} = \boldsymbol{g}^{\boldsymbol{s}'_i - \boldsymbol{x}e'_i} \pmod p$ with $\boldsymbol{s}'_i - \boldsymbol{x}e'_i \neq \boldsymbol{s}_i - \boldsymbol{x}e_i$. We then see that (removing i from the subscripts)

$$\begin{aligned}
\boldsymbol{g}^{\boldsymbol{s}-\boldsymbol{x}e} &= \boldsymbol{g}^{\boldsymbol{s}'-\boldsymbol{x}e'} \\
g_1^{s_1 - x_1 e} g_2^{s_2 - x_2 e} &= g_1^{s'_1 - x_1 e'} g_2^{s'_2 - x_2 e'} \\
g_1^{(s_1 - x_1 e) - (s'_1 - x_1 e')} &= g_2^{(s'_2 - x_2 e') - (s_2 - x_2 e)} \\
g_1^{\frac{(s_1 - x_1 e) - (s'_1 - x_1 e')}{(s'_2 - x_2 e') - (s_2 - x_2 e)}} &= g_2,
\end{aligned}$$

Algorithm 2.7 $\hat{\mathcal{B}}$, an algorithm solving dlog, given access to $\hat{\mathcal{A}}$:

1. Run $\hat{\mathcal{A}}$ with random input and a random oracle function at most $1/\varepsilon$ times, until it does a successful forgery. In case of a forgery, record all valid signatures $(m_i, (e_i, \boldsymbol{s}_i))$ and all j_i, for $i = 1, \ldots, l+1$.
2. Let N be the number of runs, or repeat Step 1 if no success occurs.
3. For $i = 1, \ldots, l+1$, run $\hat{\mathcal{A}}$ again, with the same random input as in the last success, but modify the oracle so that it returns the very same answers in the first $j_i - 1$ queries, but let the rest of the answers be unrestricted.
4. Repeat Step 3 until another attack succeeds, or stop if $\hat{\mathcal{A}}$ does not success within $120N(1+1/54k)$ tries. In case of success, record the index i giving success, and let $(m_i, (e_i', \boldsymbol{s}_i'))$ be the i'th valid signature.
5. If no success occurs in Step 4, restart from Step 1.
6. Return the logarithm of g_2 with respect to g_1 (see below).

and $\hat{\mathcal{B}}$ has with non-negligible probability found the discrete logarithm

$$\log_{g_1} g_2 = \frac{(s_1 - x_1 e) - (s_1' - x_1 e')}{(s_2' - x_2 e') - (s_2 - x_2 e)}$$

of g_2 with respect to g_1.

We are left with showing that this is a polynomial reduction, or that $\hat{\mathcal{B}}$ is a polynomial algorithm. Pointcheval and Stern do not show this explicitly, but they compute the expected running time of $\hat{\mathcal{B}}$, which they find to be bounded by $T' = 10^6(l+1)^2 k^2 QT/\varepsilon$. (Remember, T is the running time of $\hat{\mathcal{A}}$ and Q is the number of queries to the hash function.) If we now abort $\hat{\mathcal{B}}$ after cT' steps, for a constant c, then we have constructed a PPT algorithm that solves the discrete log problem.

□

Corollary 2.10. The blind Okamoto-Schnorr signature scheme is secure.

Pointcheval and Stern also mentions that the proof of non-forgeability is built upon the witness-hiding property of the signing protocol, and that it therefore easily can be adapted to cover other blind signature schemes as well, such as the scheme based on the Okamoto-Guillou-Quisquater identification scheme [19], and other schemes presented by Pointcheval and Stern.

In addition, there have been proposed other schemes that are secure under more general assumptions, like the existence of an arbitrary one-way trapdoor permutation family [16], in the common reference string model [12], or even in the standard model by Hazay et al. [15] and Okamoto [20].

Chapter 3

Group signatures

Group signature schemes were first described by Chaum and van Heyst [11], and just like blind signatures, they are an extension of the regular digital signature scheme. In a group signature scheme, a group of participants is allowed to sign a document *on behalf of the group*. Any receiver of a group signature can easily verify that this is a signature originating from some member of the group, but they cannot tell from whom. If this was the complete story, such a scheme could be defined just like a regular digital signature scheme, where all member of the group would share the same secret key. But group signatures impose the additional requirement that it should be possible for a group manager, a special person holding a secret group manager key, to open the signature and reveal the identity of the signer. This last property is important if there should be a dispute on some signature, or a malign member issues signatures that the group cannot represent. In this case, the group manager should be called upon to unambiguously reveal the misbehaving member.

This kind of cryptographic primitive is useful for instance in card payment systems, where the holder of a payment card signs a transaction and in this way guarantees the merchant that he can later go to the bank and withdraw such an amount [26]. The bank, being the group manager, can open the signature to see which account should be charged for this amount. It is also useful in voting or bidding, for example in an auction where only the identity of the highest bidder is of importance [1]. In addition, it is natural to use group signatures in authentication settings, where a member must prove he is a member of some given group, or to conceal organisational structures within a company when making official or signed statements [2].

There are several variants and extensions of the group signature scheme. For instance, we can allow the scheme to have *dynamic groups*, where members can enter and leave the group arbitrarily [11]. But then the schemes naturally become more complex, and many more security issues arise. We restrict ourselves in this paper to the case of static groups, where the number of members n is fixed and decided during setup, and refer to Bellare, Shi and Zhang [5] for a detailed study on dynamic groups. Also, Trolin [26] has introduced the additional notion of *hierarchical group signatures*, where all participants (members and managers) are organised in an hierarchical tree-like structure.

3.1 Definition

A formal definition of a group signature scheme was first made by Bellare, Micciancio and Warinschi [3], and it requires the group to have a common public key, while each member has

his own secret key. We can assume that the identity of the members corresponds with the integers $1 \leq i \leq n$. The manager also has a secret key which enables him to reveal the identity of the signer.

Definition 3.1 (Group signature scheme)**.** A *group signature scheme* $\mathcal{GS}$ consists of four polynomial-time algorithms (Gen, Sign, Vrfy, Open):

- The PPT *key generation* algorithm Gen takes the security parameter k and the group size n as input (encoded as string lengths 1^k and 1^n) and returns a tuple $(\mathsf{PK}, \mathsf{MSK}, \mathsf{GSK})$, where PK is the *group public key*, MSK is the *group manager's secret key*, and GSK is a vector (of size n) of the *members' secret keys*. We denote by GSK_i the *secret signing key* for member i.
- The PPT *group signing* algorithm Sign takes a secret signing key GSK_i and a message m as input, and returns a group signature σ of the message, signed with the given member's key.
- The DPT *group signature verification* algorithm Vrfy takes the group's public key PK, a message m and a candidate signature σ of this message and outputs either Accept or Reject.
- The DPT *opening* algorithm Open takes the group manager's secret key MSK, a message m and a signature σ as input, and outputs either i, an identity of a group member, or $\perp$ to indicate that this signature is either invalid or cannot be traced to a member.

We would of course like the verification algorithm to correctly decide between valid and invalid signatures, and we want the opening algorithm to in fact discover the originator of a correctly generated signature. Therefore, we need a correctness definition to exclude algorithms that do not behave as wanted.

Definition 3.2 (Correctness)**.** A group signature scheme (Gen, Sign, Vrfy, Open) is *correct* if for all $k, n \in \mathbb{N}$, any $(\mathsf{PK}, \mathsf{MSK}, \mathsf{GSK})$ output by $\mathsf{Gen}(1^k, 1^n)$, and all suitable i and m,

$$\mathsf{Vrfy}(\mathsf{PK}, m, \mathsf{Sign}(\mathsf{GSK}_i, m)) = \mathsf{Accept} \quad \text{and} \quad \mathsf{Open}(\mathsf{MSK}, m, \mathsf{Sign}(\mathsf{GSK}_i, m)) = i.$$

As we can see, the verification algorithm must accept, and the opening algorithm must correctly open any signature generated from a member's secret signing key. Note that correctness for example does not imply that Vrfy rejects any other signature, since this is a security requirement which is treated in detail later.

As in the blind signatures case, the key, message and signature spaces are not mentioned here, and the same discussion as in Section 2.1 applies here as well.

Length-regularity, that restricts the size of the signatures compared to the message size, also applies to group signatures, but since we in addition have a new parameter, namely the group size, we have another subtle point of the sizes of keys and signatures, named compactness [3]. This states that we want the sizes of the keys and signatures to not grow faster than polynomially in k and $\log(n)$.

Definition 3.3 (Compactness)**.** A group signature scheme is *compact* if there exist polynomials p_1 and p_2, such that for all $k, n \in \mathbb{N}$, for any $(\mathsf{PK}, \mathsf{MSK}, \mathsf{GSK})$ output by $\mathsf{Gen}(1^k, 1^n)$, all suitable i and m, and all valid signatures σ,

$$|\mathsf{PK}|, |\mathsf{MSK}|, |\mathsf{GSK}_i| \leq p_1(k, \log(n)), \text{ and } |\sigma| \leq p_2(k, \log(n), |m|).$$

It is clear (by the necessary uniqueness of the members' secret keys) that the sizes of keys and signatures must grow at least at the rate of $\log(n)$, but it is also known that this bound is reachable [3].

We will from now on assume that all group signature schemes are correct, length-regular and compact, and proceed to the security requirements of group signatures.

3.2 Security requirements

Bellare, Micciancio and Warinschi [3] present two security requirements regarding group signatures. The first concerns the anonymity of a group member, guaranteeing that it should be infeasible to open the signature without having knowledge of the group manager's secret key. The other regards non-forgeability, which in this setting not only prevents a user outside the group from creating signatures on behalf of the group, but also maintains the integrity of an honest group member, such that it is infeasible for any other individual or group of people to create a signature that opens to the honest member.

These two requirements can be thought of as representing attacks respectively by an adversary trying to find the originator of a signature, and by forgers trying to create signatures on behalf of others.

Other notions of security regarding blind signatures have also existed, and Ateniese et al. [1, 2] mention unforgeability, anonymity, unlinkability, exculpability, traceability and coalition resistance, but they appear quite vague and never explicitly defined. However, Bellare, Micciancio and Warinschi [3] try to formalise all these diffuse concepts, and they show that they all can be reduced to two requirements, which they call full-anonymity and full-traceability. Since we feel there is an inconsistency in the nomenclature[1], and believe the latter relates more to forging than tracing, we have chosen to change the name from full-traceability to full-non-forgeability.

3.2.1 Full-anonymity

This requirement concerns the anonymity of group members. A group member can be confident of that a signature he makes cannot be opened without the use of the manager's secret key. There are many different attacks an adversary can carry out in order to try to find the signer of a signature. He might have seen the opening of other signatures, or he might even be able to ask the manager to open other signature of his choice, similar to a chosen-message attack on regular signatures. There is also a possibility that the adversary controls other members of the group, and is able to create signatures with these corrupted members' keys. And last, we will also ensure that even if the signer himself becomes corrupted, earlier signatures will not automatically trace back to him. Even an honest member can be corrupted if his private key somehow becomes exposed.

We want to protect the scheme against very powerful adversaries, and we will therefore allow any adversary attacking anonymity to adaptively corrupt any member of the group and to ask the manager to open any signature he wants (except of course the target signature). Formally, this is achieved by giving the adversary a list of all the group member's keys, and giving him oracle access to the opening algorithm [5].

[1] Anonymity is an attractive characteristic, while traceability certainly is not.

Similar to the blindness requirement of blind signatures, we will define an experiment where the adversary wins if he can distinguish between the signatures of two chosen group members. Similar to the existential forgery setting of regular digital signatures, both the message and the two members are chosen by the adversary, guaranteeing that there does not exist any significantly large set of message-signature pairs being weak and easy to trace.

Definition 3.4 (Full-anonymity)**.** Consider the experiment $\mathbf{Exp}^{\text{anon}}_{\mathcal{GS},\hat{\mathcal{A}}}(k,n)$, where $\hat{\mathcal{A}}$ is any algorithm. We define the advantage of $\hat{\mathcal{A}}$ in breaking $\mathcal{GS}$ with respect to *full-anonymity* as

$$\mathbf{Adv}^{\text{anon}}_{\mathcal{GS},\hat{\mathcal{A}}}(k,n) = \left|\Pr[b'=1 \mid b=1] - \Pr[b'=1 \mid b=0]\right|,$$

where the probability is taken over all random choices done by all of the algorithms in the experiment.

The group signature scheme $\mathcal{GS}$ is *fully-anonymous* if for all PPT algorithms $\hat{A}$, $\mathbf{Adv}^{\text{anon}}_{\mathcal{GS},\hat{\mathcal{A}}}(k,n)$ is negligible in the sense of two-argument functions.

Experiment 3.1 $\mathbf{Exp}^{\text{anon}}_{\mathcal{GS},\hat{\mathcal{A}}}(k,n)$:

---

1. Let $(\mathsf{PK},\mathsf{MSK},\mathsf{GSK}) \leftarrow \mathsf{Gen}(1^k,1^n)$.
2. Let $(i_0,i_1,m) \leftarrow \hat{\mathcal{A}}^{\mathsf{Open}(\mathsf{MSK},\cdot,\cdot)}(\mathsf{choose},\mathsf{PK},\mathsf{GSK})$.
3. Let $b \xleftarrow{r} \{0,1\}$, and $\sigma \leftarrow \mathsf{Sign}(\mathsf{GSK}_{i_b},m)$.
4. Let $b' \leftarrow \hat{\mathcal{A}}^{\mathsf{Open}(\mathsf{MSK},\cdot,\cdot)}(\mathsf{guess},\sigma)$.
5. Set $b' \leftarrow 0$ if $\hat{\mathcal{A}}$ queried $\mathsf{Open}(\mathsf{MSK},m,\sigma)$ (its oracle) during the guess stage.

Recall that the notation $\hat{A}^{\mathsf{Open}(\mathsf{MSK},\cdot,\cdot)}$ means that the adversary $\hat{A}$ has oracle access to the algorithm Open with the first input being fixed and hidden from $\hat{A}$. The adversary can query the oracle a polynomially number of times (in k and n). The adversary is also allowed to know and use the other algorithms of the scheme. Note that this implies that the signing algorithm cannot be deterministic, or else the adversary could just run the signing algorithm himself to discover which user did the signing. We give the adversary oracle access to Open such that he can open any signature he wants without being given MSK explicitly, and because we then can monitor if the adversary tries to open the candidate signature.

In Step 2, the adversary is given the secret keys of all group members, modelling that he possibly has corrupted all of them. He shall then choose two candidate members of the group, and the message he wants to have signed. In Step 3 of the experiment, one of the two members is uniformly chosen to sign the message, and in Step 4, the adversary is asked to guess which of the two candidates did the actual signing. He can ask the manager to reveal the signer of any message, it could for instance be helpful to ask $\mathsf{Open}(\mathsf{MSK},m,\sigma')$ for a slightly different σ', but if the adversary asks about the specific message signature pair, his guess is declared void in Step 5.

It is clear that this adversary tries to distinguish between the output distributions of $\mathsf{Sign}(\mathsf{GSK}_{i_0},m)$ and $\mathsf{Sign}(\mathsf{GSK}_{i_1},m)$, and that if these distributions are not polynomially indistinguishable, the anonymity of the members are not guaranteed.

It seems natural that if the adversary is allowed to choose many messages that he wants to have signed (instead of just one), he would have a better chance of winning. But if an adversary has non-negligible success rate after asking for polynomially many signatures, a hybrid argument on a series of games or experiments [13, 24] shows that there must be at least one signature that is decisive. In short, the hybrid argument is that the sum of polynomially many negligible functions is still negligible. The consequence is that on this decisive signature, the adversary has a non-negligible chance of distinguishing the two members. So the case of polynomially many messages reduces to this experiment of a single message [3].

3.2.2 Full-non-forgeability

Full-non-forgeability (in [3] denoted full-traceability) concerns the possible *framing* of a group member. Framing occurs when a member does not take part in creating a signature, but the opening algorithm says he is the originator. This could be a forgery by an outsider (*unforgeability*), by a member of the group (*exculpability*), or by many group members that join forces in order to try to frame another member (*coalition-resistance*). In addition, we require that framing is infeasible even if the manager's key is available to the adversary [2].

Another possibility is that an adversary can create a signature that looks valid to the verification algorithm, but if the signature is opened, the manager cannot find a member and outputs $\perp$ (*traceability*).

Since the group manager has the power to revoke the anonymity of a member by opening a signature, it is of great importance that the integrity of an honest member is secure. However, as noted in [3], framing attempts by a corrupt or dishonest group manager is a different story, because then we cannot assume that the opening algorithm runs as described. Perhaps the manager always blames one specific member, regardless of input, or he may refuse to open any signature. We will from now on assume that all algorithms of the scheme (Gen, Sign, Vrfy, and Open) behave as expected, and refer to Bellare, Micciancio and Warinschi [3] for a further treatment on dishonest managers.

Definition 3.5 (Full-non-forgeability)**.** Consider the experiment $\mathbf{Exp}^{\text{forge}}_{\mathcal{GS},\hat{\mathcal{A}}}(k,n)$, listed in Experiment 3.2, where $\hat{\mathcal{A}}$ is any algorithm. We define the advantage of $\hat{\mathcal{A}}$ in breaking $\mathcal{GS}$ with respect to *full-non-forgeability* as

$$\mathbf{Adv}^{\text{forge}}_{\mathcal{GS},\hat{\mathcal{A}}}(k,n) = \Pr[\mathbf{Exp}^{\text{forge}}_{\mathcal{GS},\hat{\mathcal{A}}}(k,n) = 1],$$

where the probability is taken over all random choices done by all of the algorithms in the experiment.

The scheme $\mathcal{GS}$ is *fully-non-forgeable* if for all PPT algorithms $\hat{A}$, $\mathbf{Adv}^{\text{forge}}_{\mathcal{GS},\hat{\mathcal{A}}}(k,n)$ is negligible in the sense of two-argument functions.

The attack on full-non-forgeability is very powerful, where in the first step of the experiment, the adversary learns the public key and the manager's secret key. In Step 2, the adversary can choose any set C of members he wants to control, so he is successively given the secret key of the last member he chose to corrupt. The adversary ends Step 2 by outputting $\perp$, signalling that he is content with his set of corrupted members. Recall that we implicitly assume that the adversary is allowed to maintain state between calls[2], and so at the end of this step, in addition

[2]To make this clear, we could like [3] introduce a state variable St, that is always part of the adversary's input and output, but we believe this only clutters the notation.

Experiment 3.2 $\mathbf{Exp}^{\text{forge}}_{\mathcal{GS},\hat{\mathcal{A}}}(k,n)$:

1. Let $(\mathsf{PK},\mathsf{MSK},\mathsf{GSK}) \leftarrow \mathsf{Gen}(1^k,1^n)$, and run $\hat{\mathcal{A}}(\mathsf{PK},\mathsf{MSK})$.
2. Let $\hat{\mathcal{A}}$ adaptively choose a set C of members to corrupt: Set $i \leftarrow \perp$, and repeatedly run $i \leftarrow \hat{\mathcal{A}}(\mathsf{choose},\mathsf{GSK}_i)$, until $i = \perp$ again.
3. Let $(m,\sigma) \leftarrow \hat{\mathcal{A}}^{\mathsf{Sign}(\mathsf{GSK}_{[\cdot]},\cdot)}(\mathsf{guess})$.

   $\hat{\mathcal{A}}$ succeeds if he can produce a valid signature that opens to a non-corrupt member, or that cannot be traced to any member of the group at all:
4. If $\mathsf{Vrfy}(\mathsf{PK},m,\sigma) = \mathsf{Reject}$, return 0.
5. If $\mathsf{Open}(\mathsf{MSK},m,\sigma) = \perp$, return 1.
6. If $\mathsf{Open}(\mathsf{MSK},m,\sigma) = i$ for an $i \notin C$, and in addition $\mathsf{Sign}(\mathsf{GSK}_i,m)$ was not queried by $\hat{\mathcal{A}}$, then return 1.
7. Else, return 0.

to PK and MSK, the adversary knows GSK_i for all $i \in C$.

In Step 3, we ask the adversary to provide a signature, and the goal of the adversary is to produce any kind of valid signature that opens to a non-corrupted member (Step 6) or to no one at all (Step 5). During the guess stage, the adversary will have access to a signing oracle, letting him produce valid signatures on any message by any (even non-corrupted) members. This is necessary, because the adversary might get some valuable information by asking the target member to sign some messages, without the member being aware of that the other part of the interaction is an adversary trying to forge signatures. But it is crucial that any attempt to produce a signature of this specific message via the oracle will make the adversary fail the experiment. This is checked in Step 6.

As in the full-anonymity experiment, the adversary will know the signing algorithm, and he has access to this algorithm (or could just simulate it) to sign a message with any key that he chooses. Again, the oracle enables him to get signatures from a specific member without explicitly knowing this member's key.

3.2.3 Other security requirements

Before Bellare, Micciancio and Warinschi [3] simplified the various existing security notions of group signature schemes and introduced full-anonymity and full-non-forgeability, there were many more security notions that had to be checked. We will now present these obsolete notions, for example mentioned by Ateniese and Tsudik [2] and Ateniese et al. [1], with proofs or arguments that they all reduce to either full-anonymity or full-non-forgeability.

Exculpability Exculpability is the requirement that neither the group manager nor another group member can create signatures that open to another member. This can be seen to be a special case of full-non-forgeability, where the adversary is restricted to corrupt only one member, and perhaps not knowing MSK at all. For instructional purposes, we will give a formal

definition of exculpability, and prove that exculpability is a redundant requirement if we require full-non-forgeability [3].

Definition 3.6 (Exculpability)**.** Consider the two experiments $\mathbf{Exp}^{\text{exculp1}}_{\mathcal{GS},\hat{\mathcal{A}}}(k,n)$ and $\mathbf{Exp}^{\text{exculp2}}_{\mathcal{GS},\hat{\mathcal{A}}}(k,n)$, listed in Experiments 3.3 and 3.4, where $\hat{\mathcal{A}}$ is any algorithm. The scheme $\mathcal{GS}$ is *secure with respect to exculpability* if for all PPT algorithms $\hat{\mathcal{A}}$, the advantage in both of these experiments (defined in the usual manner) is negligible in the sense of two-argument functions.

Experiment 3.3 $\mathbf{Exp}^{\text{exculp1}}_{\mathcal{GS},\hat{\mathcal{A}}}(k,n)$:

---

1. Let $(\mathsf{PK},\mathsf{MSK},\mathsf{GSK}) \leftarrow \mathsf{Gen}(1^k,1^n)$.
2. Let $i \leftarrow \hat{\mathcal{A}}(\mathsf{choose},\mathsf{PK},\mathsf{MSK})$ (where i is allowed to be $\perp$).
3. Let $(m,\sigma) \leftarrow \hat{\mathcal{A}}^{\mathsf{Sign}(\mathsf{GSK}_{[\cdot]},\cdot)}(\mathsf{guess},\mathsf{GSK}_i)$ (where $\mathsf{GSK}_\perp = \perp$).

Steps 4 - 7: The same as Experiment 3.2 (with $C = \{i\}$).

Experiment 3.4 $\mathbf{Exp}^{\text{exculp2}}_{\mathcal{GS},\hat{\mathcal{A}}}(k,n)$:

---

1. Let $(\mathsf{PK},\mathsf{MSK},\mathsf{GSK}) \leftarrow \mathsf{Gen}(1^k,1^n)$.
2. Let $i \leftarrow \hat{\mathcal{A}}(\mathsf{choose},\mathsf{PK})$.
3. Let $(m,\sigma) \leftarrow \hat{\mathcal{A}}^{\mathsf{Sign}(\mathsf{GSK}_{[\cdot]},\cdot)}(\mathsf{guess},\mathsf{GSK}_i)$.

Steps 4 - 7: The same as Experiment 3.2 (with $C = \{i\}$).

Theorem 3.7 (Full-non-forgeability $\rightsquigarrow$ Exculpability)**.** If a group signature scheme $\mathcal{GS}$ is fully-non-forgeable, then it is also secure with respect to exculpability.

Proof. We want to show that if the scheme is not secure with respect to exculpability, then we can use an adversary breaking this requirement to construct another adversary that breaks the full-non-forgeability requirement:

Let $\hat{\mathcal{A}}$ be an adversary breaking the exculpability requirement, having success in either Experiment 3.3 or 3.4. Now use $\hat{\mathcal{A}}$ to construct the following PPT algorithm $\hat{\mathcal{B}}$ which will interact in Experiment 3.2:

The first call to the adversary is $\hat{\mathcal{B}}(\mathsf{PK},\mathsf{MSK})$, so store these keys in $\hat{\mathcal{B}}$'s state information. When $\hat{\mathcal{B}}$ is called with choose as the first parameter, call $\hat{\mathcal{A}}$ and let $i \leftarrow \hat{\mathcal{A}}(\mathsf{choose},\mathsf{PK},\mathsf{MSK})$. (Possibly excluding MSK if $\hat{\mathcal{B}}$ is built from an adversary running against Experiment 3.4.) Return this i. If $i \neq \perp$, another call, $\hat{\mathcal{B}}(\mathsf{choose},\mathsf{GSK}_i)$, is performed by the experiment, and if so, store i's secret key and return $\perp$ this time. When being challenged to forge a signature in Step 3 of Experiment 3.2, forward the challenge to $\hat{\mathcal{A}}$ and return whatever $\hat{\mathcal{A}}^{\mathsf{Sign}(\mathsf{GSK}_{[\cdot]},\cdot)}(\mathsf{guess},\mathsf{GSK}_i)$ returns.

Since the rest of the steps of the experiments are equal, the advantage of $\hat{\mathcal{B}}$ is equal to the advantage of $\hat{\mathcal{A}}$ (in their respective experiments). If a group signature scheme does not have

exculpability, then $\hat{\mathcal{A}}$ as a non-negligible chance of success in either Experiments 3.3 or 3.4, and then $\hat{\mathcal{B}}$ has also in Experiment 3.2, leading to a break of full-non-forgeability. We have thus carried out a reduction from full-non-forgeability to exculpability.

□

We do not describe or treat the rest of the other security requirements of group signature schemes in the same formal detail as above, but some informal notions are introduced, and sketches of reduction proofs are shown.

Non-forgeability As with regular signatures, non-forgeability or unforgeability is related to the creation of valid signatures without having the secret key [1]. Informally, a group signature scheme is *non-forgeable* if there does not exist a PPT algorithm having a non-negligible probability of success of creating a message-signature pair (m, σ) such that $\mathsf{Vrfy}(\mathsf{PK}, m, \sigma) = \mathsf{Accept}$ without having access to any part of GSK [3].

Theorem 3.8 (Full-non-forgeability $\rightsquigarrow$ Non-forgeability)**.** If a group signature scheme is fully-non-forgeable, then it is also non-forgeable.

Proof. We want to show that if an adversary violates the forgeability requirement, then the same adversary breaks the full-non-forgeability requirement:

Let $\hat{\mathcal{A}}$ be an adversary breaking the non-forgeability requirement, then in Experiment 3.2, $\hat{\mathcal{A}}$ does not need to corrupt any members. Because the message signature pair created by this adversary is accepted by Vrfy, it passes Step 4, and because C is empty, it will also succeed either Step 5 or Step 6 with the same probability of success as in the potential experiment regarding non-forgeability, and $\hat{\mathcal{A}}$ therefore breaks full-non-forgeability. □

Traceability Traceability in its original meaning meant that all validly produced signatures must be traced to the correct member [11] . This is now incorporated in the statement about the opening algorithm in the correctness requirement. It has later been used in another meaning [2], which is more correctly named coalition resistance [1].

Coalition resistance and framing Ateniese et al. considered in [1] a new type of attack, where a group of signers would collude and perhaps create signatures that cannot be traced to any of them. Either to no members at all (called *coalition resistance*), or a member not part of the colluding group (*framing*). In the strongest attacks of coalition resistance and framing, the colluding group could be chosen adaptively, so a definition using a variation of Experiment 3.2, except where the adversary is not given the manager's key MSK (since the manager is not part of the group) would seem appropriate. Since these notions with this observation become a special case of full-non-forgeability, it is clear that both of them follow from full-non-forgeability.

Theorem 3.9 (Full-non-forgeability $\rightsquigarrow$ Coalition resistance)**.** If a group signature scheme is fully-non-forgeable, then it is also coalition resistant.

Proof. If there exists a PPT algorithm $\hat{\mathcal{A}}$ that breaks coalition resistance, we can use the exact same adversary to break full-non-forgeability, by just ignoring the extra information MSK given to $\hat{\mathcal{A}}$ in Step 1 of the experiment. The reduction follows immediately. □

Anonymity Anonymity is the natural requirement that the signature message pair itself should not reveal anything about the identity of the signer. Thus, an adversary that does not know the manager's key or having access to the opening oracle should not be able to effectively guess which member did the actual signature. Anonymity intuitively follows from full-anonymity.

Theorem 3.10 (Full-anonymity $\rightsquigarrow$ Anonymity)**.** If a group signature scheme is fully-anonymous, then it is also anonymous.

Proof. If a PPT adversary $\hat{\mathcal{A}}$ breaks anonymity, then in Experiment 3.1 he neither needs the opening oracle nor the secret keys GSK in order to have success, and $\hat{\mathcal{A}}$ breaks full-anonymity with the exact same probability of success. □

Unlinkability The last security requirement mentioned in Bellare, Micciancio and Warinschi [3] is *unlinkability*. This concept is, like in the blind signature case, the notion that no adversary after seeing a list of valid signatures, can select two of them and conclude with a non-negligible probability of success that they were made by the same member. Note that the adversary, in contrast to the anonymity case, does not need to say which member supposedly made the signature. This requirement is useful, because even if such an adversary does not threaten the anonymity of any specific member, it could reveal some structural information about the group.

Intuitively, this is a different flavour of anonymity, but it is nevertheless claimed in [3] that regardless of how they choose to formalise the notion of unlinkability, they can show that it both follows from and leads to anonymity, so the two concepts are totally equivalent.

3.2.4 Summary

Again, we have identified two security requirements, now regarding group signature schemes. From the discussion above, it seems that any other reasonable security requirement is implied by full-anonymity and full-non-forgeability. Remember that the discussion in this paper only applies when we assume that the opening algorithm is honest. We then conclude the discussion by defining a secure group signature scheme.

Definition 3.11 (Secure group signature scheme)**.** If a group signature scheme is both fully-anonymous and fully-non-forgeable, then it is *secure*.

After Bellare, Micciancio and Warinschi [3] summarised the vague concepts mentioned above and identified the two main security requirements, formal proofs of the security of group signature schemes will hopefully be shorter and easier than they were before.

In the same paper, they created a scheme that is provably secure (from the definition above), under the assumption of the existence of trapdoor permutations. This scheme is sketched in the next section.

As mentioned, we have not taken dynamic groups into consideration, but Bellare, Shi and Zhang [5] give formal security requirements for group signature schemes that allow group members to be added after the initial key generation. They also extend the scheme presented in [3] to allow the addition of members, still being secure in the dynamic setting. However, they do not propose a fully dynamic scheme that allow both addition and removal of group members. Such schemes are only shortly discussed in [5], and they are the next natural step of group signatures.

3.3 A secure general group signature scheme

In the paper where Bellare, Micciancio and Warinschi [3] present the two main security arguments for group signatures, they also construct a provably secure scheme, in order to justify their security definitions. The scheme is a general construction built from a regular digital signature scheme $\mathcal{DS}$, a public-key encryption scheme $\mathcal{AE}$, and a non-interactive zero-knowledge (NIZK) proof system $\mathcal{PS}$.

The digital signature scheme $\mathcal{DS} = (\mathsf{Gen_s}, \mathsf{Sign_s}, \mathsf{Vrfy_s})$ must be non-forgeable under adaptive chosen-message attack (Section 1.3), and the encryption scheme $\mathcal{AE} = (\mathsf{Gen_e}, \mathsf{Enc}, \mathsf{Dec})$ must satisfy indistinguishability under chosen-ciphertext attack (IND-CCA) [3, 25]. Both of these security notions are standard, and such schemes are known to exist assuming the existence of a family of trapdoor permutations [3].

Finally, for anonymity, the scheme uses a NIZK proof system $\mathcal{PS} = (\mathcal{P}, \mathcal{V})$ for an NP-relation [26]. It requires the proof system to be simulation-sound. A NIZK proof system is a system where a user in one move (non-interactive) proves that he knows a secret, without revealing any part of the secret (zero-knowledge). Specifically, for NP-relations, it means that the user can prove that he knows the *witness* w to a *theorem* x in an NP-relation ρ. If $(x, w) \in \rho$, then the witness w can be sent to the prover $\mathcal{P}$ to generate a proof $\pi \leftarrow \mathcal{P}(x, w)$, and the verifier $\mathcal{V}(x, \pi)$ can check if this is a valid proof. It should be hard to find a witness of a random x, and without a witness, it should be hard to generate a proof. We refer to [5] for the complete definition of a NIZK proof system, and for the description of simulation-soundness. We have also left out the details regarding the common reference string (see Section 1.4), which is needed to make the system work. Such proof systems are known to exist over any NP-relation if trapdoor permutations exists.

Using these cryptographic primitives, Bellare, Micciancio and Warinschi prove that their scheme is both fully-anonymous and fully-non-forgeable, and therefore secure. We see that under the assumption that trapdoor permutations exist, all three building blocks of this scheme exist, having the required security. Therefore, this group signature exists and is secure under the same assumption.

We will now sketch the system while referring to Figure 3.1, but we do not cover the security proof.

The key generation algorithm Gen produces a *group encryption key pair* $(\mathsf{PK_e}, \mathsf{SK_e}) \leftarrow \mathsf{Gen_e}$, which is used for the opening of signatures. Whenever a signature is made, the user encrypts his identity i with the *public encryption key* $\mathsf{PK_e}$ and passes this along with the signature. The manager is given the corresponding *secret encryption key* $\mathsf{SK_e}$, so that he can decrypt the signer's identity. The generation algorithm also creates *signature key pairs* $(\mathsf{PK}_i, \mathsf{SK}_i) \leftarrow \mathsf{Gen_s}$ and gives each member one such pair. To prevent a user from creating and using other signature keys, the generation algorithm must make a signature cert_i on (PK_i, i) (using a *secret certificate signing key* $\mathsf{SK_s}$), which is a *certificate* that the public key PK_i in fact corresponds to member i. The *public certificate signing key* $\mathsf{PK_s}$ is given to all members, so they can prove the validity of their certificate. But the secret key must be thrown away, so that it is impossible to create more members. In this way, all signatures can eventually be traced to a member of the group.

For completeness' sake, we define the keys of the group signature scheme as $\mathsf{PK} = (\mathsf{PK_e}, \mathsf{PK_s})$, $\mathsf{MSK} = (\mathsf{SK_e}, \mathsf{PK})$, and $\mathsf{GSK}_i = (i, (\mathsf{PK}_i, \mathsf{SK}_i), \mathsf{cert}_i, \mathsf{PK_s}, \mathsf{PK})$.

To sign a message m, the member just makes a regular signature $s \leftarrow \mathsf{Sign_s}(\mathsf{SK}_i, m)$ with his private signing key. However, if this signature was supposed to be verified with his public

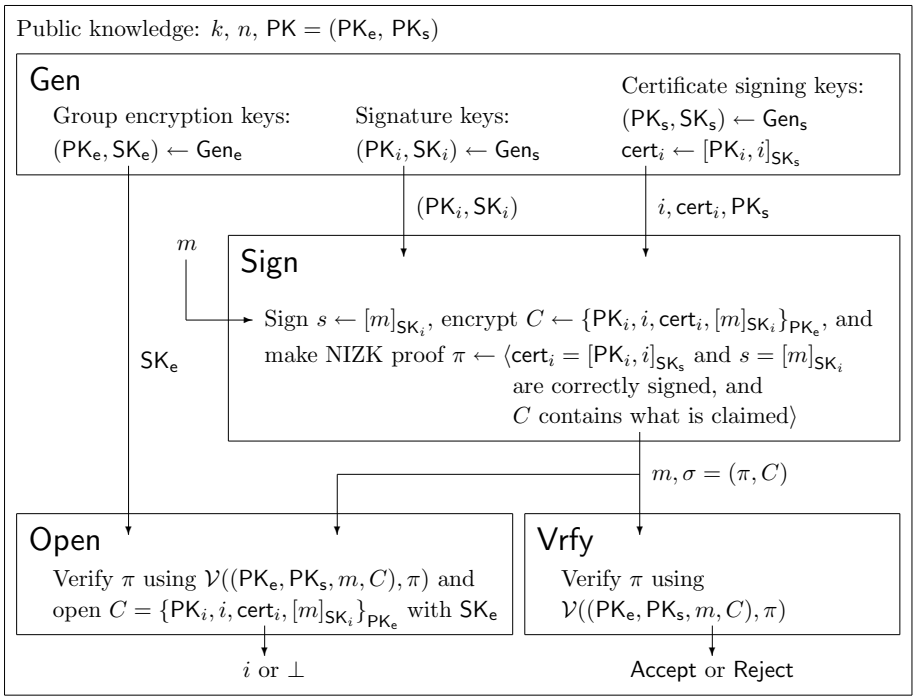

Figure 3.1: A model of the secure group signature scheme, where $[\cdot]_K$ denotes signing with key K, $\{\cdot\}_K$ denotes encrypting with key K, and $\langle\cdot\rangle$ denotes making a NIZK proof

key PK_i, his identity must first be revealed. Therefore, he creates a NIZK proof that s is actually signed using his private key, and that verification with his public key would succeed. To prove that he is a valid member of the group (and did not come up with a separate signature key), he must also prove in zero-knowledge that his certificate would verify his identity under the public certificate signing key. At last, he must prove that the encryption of his identity $C = \mathsf{Enc}(\mathsf{PK_e}, (\mathsf{PK}_i, i, \mathsf{cert}_i, s))$ is correct. He therefore uses the NIZK prover $\mathcal{P}$ to create a proof $\pi \leftarrow \mathcal{P}((\mathsf{PK_e}, \mathsf{PK_s}, m, C), (i, \mathsf{PK}_i, \mathsf{cert}_i, s))$ of all these claims. The tuple $(i, \mathsf{PK}_i, \mathsf{cert}_i, s)$ is the witness that his hidden inside the proof. The resulting signature $\sigma = (\pi, C)$ is the combination of the NIZK proof and his encrypted identity.

The verification algorithm Vrfy is only left with checking (using $\mathcal{V}$) that the proof π is a valid proof of the claims above, and if there is a need to open a signature, the Open algorithm can call $\mathsf{Dec}(\mathsf{SK_e}, C)$ to decrypt and reveal the signer's identity. The opening algorithm outputs $\perp$ if the signature was invalid in the first place, or if the signature was forged.

Bellare, Micciancio and Warinschi then note that this scheme is compact, and proceed to prove that this scheme is secure, using the following lemmas.

Lemma 3.12. If $\mathcal{AE}$ is an IND-CCA secure encryption scheme and $\mathcal{PS}$ is a simulation-sound computational zero-knowledge proof system for ρ, then the group signature scheme is fully-anonymous.

Lemma 3.13. If the digital signature scheme $\mathcal{DS}$ is secure against forgery under chosen-message attack and $\mathcal{PS}$ is a sound non-interactive proof system for ρ, then the group signature scheme is fully-non-forgeable.

The proofs of these lemmas use the standard reduction technique to show that any efficient attack on full-anonymity or full-non-forgeability will lead to a successful attack on one of the underlying primitives.

This scheme is built from the ideas of Ateniese et al. [1], but Bellare, Micciancio and Warinschi adjusted this scheme to make it provably secure under the two new security requirements. For example, they found that the underlying encryption scheme had to be secure against chosen-ciphertext attack, and not only chosen-plaintext attack [3].

As mentioned in the previous section, this scheme is extendible to allow later addition of group members, called partially dynamic groups. However, the definition and the security requirements of such schemes become increasingly complex, and we refer to Bellare, Shi and Zhang [5] for the description of this extension.

Chapter 4

Conclusion

We have in this thesis seen that there exists a formal framework for proving security of various cryptographic protocols and schemes. We have in detail presented the formal definitions of blind and group signature schemes, and we have treated their security requirements likewise. We have seen that both schemes have two main security requirements, which both must be fulfilled in order to call the scheme secure. From the presentation of two secure schemes, one of each type, we conclude that the theory about such schemes is well-founded.

However, blind and group signature schemes still remain very theoretical, and they have not seen widespread practical use. The proposed schemes are either general or inefficient, or not provably secure. But we hope that since the theoretical framework now seems ready, these digital signature schemes will begin to be utilised.

Bibliography

[1] Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. A practical and provably secure coalition-resistant group signature scheme. In Mihir Bellare, editor, *CRYPTO*, volume 1880 of *Lecture Notes in Computer Science*, pages 255–270. Springer, 2000.

[2] Giuseppe Ateniese and Gene Tsudik. Some open issues and new directions in group signatures. In Matthew K. Franklin, editor, *Financial Cryptography*, volume 1648 of *Lecture Notes in Computer Science*, pages 196–211. Springer, 1999.

[3] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In Eli Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 614–629. Springer, 2003.

[4] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[5] Mihir Bellare, Haixia Shi, and Chong Zhang. Foundations of group signatures: The case of dynamic groups. In Alfred Menezes, editor, *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 136–153. Springer, 2005.

[6] Jean-Paul Boly, Antoon Bosselaers, Ronald Cramer, Rolf Michelsen, Stig Frode Mjølsnes, Frank Muller, Torben P. Pedersen, Birgit Pfitzmann, Peter de Rooij, Berry Schoenmakers, Matthias Schunter, Luc Vallée, and Michael Waidner. The esprit project cafe - high security digital payment systems. In Dieter Gollmann, editor, *ESORICS*, volume 875 of *Lecture Notes in Computer Science*, pages 217–230. Springer, 1994.

[7] Stefan A. Brands. An efficient off-line electronic cash system based on the representation problem. Technical report, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, The Netherlands, 1993.

[8] Aslak Bakke Buan, Kristian Gjøsteen, and Lillian Kråkmo. Universally composable blind signatures in the plain model. Cryptology ePrint Archive, Report 2006/405, 2006. `http://eprint.iacr.org/`.

[9] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO*, Advances in Cryptology: Proceedings of CRYPTO '82, pages 199–203. Plemum, 1982.

[10] David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In Shafi Goldwasser, editor, *CRYPTO*, volume 403 of *Lecture Notes in Computer Science*, pages 319–327. Springer, 1988.

[11] David Chaum and Eugène van Heyst. Group signatures. In Donald W. Davies, editor, *EUROCRYPT*, volume 547 of *Lecture Notes in Computer Science*, pages 257–265. Springer, 1991.

[12] Marc Fischlin. Round-optimal composable blind signatures in the common reference string model. In Cynthia Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 60–77. Springer, 2006.

[13] Kristian Gjøsteen. A self-study introduction to game hopping and provable security. Notes for a cryptology course at Norwegian University of Science and Technology (NTNU), 2007.

[14] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *STOC*, pages 291–304. ACM, 1985.

[15] Carmit Hazay, Jonathan Katz, Chiu-Yuen Koo, and Yehuda Lindell. Concurrently-secure blind signatures without random oracles or setup assumptions. In Salil P. Vadhan, editor, *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 323–341. Springer, 2007.

[16] Ari Juels, Michael Luby, and Rafail Ostrovsky. Security of blind digital signatures (extended abstract). In Burton S. Kaliski Jr., editor, *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 150–164. Springer, 1997.

[17] Harry R. Lewis and Christos H. Papadimitriou. *Elements of the Theory of Computation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1997.

[18] Børge Nordli. Digital cash systems. Report from the final year literature study project at Norwegian University of Science and Technology (NTNU), 2006.

[19] Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In Ernest F. Brickell, editor, *CRYPTO*, volume 740 of *Lecture Notes in Computer Science*, pages 31–53. Springer, 1992.

[20] Tatsuaki Okamoto. Efficient blind and partially blind signatures without random oracles. In Shai Halevi and Tal Rabin, editors, *TCC*, volume 3876 of *Lecture Notes in Computer Science*, pages 80–99. Springer, 2006.

[21] Tatsuaki Okamoto and Kazuo Ohta. Divertible zero knowledge interactive proofs and commutative random self-reducibility. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *EUROCRYPT*, volume 434 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 1989.

[22] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 13(3):361–396, 2000.

[23] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 4(3):161–174, 1991.

[24] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. `http://eprint.iacr.org/`.

[25] Douglas R. Stinson. *Cryptography: Theory and Practice, Second Edition*. CRC Press, Boca Raton, FL, USA, 2002.

[26] Mårten Trolin. *Electronic Cash and Hierarchical Group Signatures*. PhD thesis, Royal Institute of Technology (KTH), 2006.

The End

Børge Nordli, 6 July 2007