# Chapter 2

# Cryptographic Preliminaries

In this chapter we review all the basic cryptographic primitives needed in the rest of the book: one-way and collision-intractable functions, proofs of knowledge, digital signatures, and public-key certificates. We introduce two new functions that are one-way and collision-intractable, and for both design practical techniques for proving knowledge of an inverse and for constructing digital signatures. These functions are central to our constructions of issuing and showing protocols in the next two chapters. We also introduce a new kind of digital certificates, called secret-key certificates; the benefits of these will become clear in Chapters 4 and 5.

## 2.1 Notation, terminology, and conventions

### 2.1.1 Basic notation

The notation "$x := y$" means that the value of $y$ is assigned to $x$. Typically, $y$ is an arithmetic expression. The "$=$" symbol denotes equality; upon $x := y$ we have $x = y$.

Whenever we say that a number is chosen "at random" from a set $V$, we imply a uniform distribution over the set $V$, independent of the probability distributions of any other variables explicitly considered in the same context. Within a figure, the notation $x \in_{\mathcal{R}} V$ is used to denote the same.

All the constructions in this book are based on elementary algebra. The notation $\mathbb{Z}_t$ denotes the ring of integers modulo $t$ and $\mathbb{Z}_t^*$ its multiplicative group, and GF$_t$ denotes the finite field with $t$ elements. Computations involving numbers in a finite ring or in a multiplicative group must always be interpreted as computations in these structures. For example, if $x, y \in \mathbb{Z}_t^*$, then "$xy$" stands for $xy \bmod t$. In cases where we consider arithmetic involving exponents, the modulo operator is often made explicit for greater certainty. In mathematical proofs, an element in $\mathbb{Z}_t$ or in $\mathbb{Z}_t^*$ may be interpreted in the algebraic sense of denoting a congruence class, but in constructions of algorithms and protocols it always represents a unique number between $0$ and $t-1$, usually encoded in binary. In particular, if $x, y \in \mathbb{Z}_t$, then "$x = y$" means that $x$ and $y$ are the same number, not merely that they are congruent modulo $t$.

For any $x \in \mathbb{Z}$ and any $y \in \mathbb{N}$, $x \bmod y$ denotes the number $x^* \in \{0, \ldots, y-1\}$ such that $x^* - x$ is an integer multiple of $y$. In keeping with mathematical tradition, the $\mathrm{mod}$ operator acts on the entire arithmetic expression preceding it, unless parentheses specify otherwise. For instance, $a + b \bmod y$ stands for $(a + b) \bmod y$.

The notation $\mathrm{div}$ is defined by $x = x \bmod y + y\,(x \;\mathrm{div}\; y)$, for any $x \in \mathbb{Z}$ and any $y \in \mathbb{N}$. Parentheses indicate the arithmetic expression on which $\mathrm{div}$ operates.

The notation "$|X|$" can have three different meanings. If $X$ is a set, then $|X|$ denotes the size of $X$, i.e., the number of elements in $X$; if $X$ is a number in $\mathbb{R}$, then $|X|$ denotes its absolute value; and if $X$ is by definition a positive integer then $|X|$ denotes its binary size (length). The appropriate meaning will always be clear from the context.

### 2.1.2 Algorithms, security parameters, and probability

An *algorithm* is a procedure that, on given some input, always halts after a finite number of steps. For concreteness, intractability assumptions are always stated in the uniform complexity model, and we construct algorithms that can be formalized by Turing machines.[1] Thus, all algorithms have a read-only *input tape*, a *work tape*, and a write-only *output tape*. Probabilistic algorithms in addition have a *random tape* containing their coin flips. Once the computation of an algorithm comes to a halt, its finite-state control enters into either an accept or reject state; correspondingly, the algorithm is said to accept or reject. In addition, it may have written an output onto the cells of its output tape; $A(x)$ denotes the output of algorithm $A$ on input $x$. If $A$ is probabilistic, then $A(x)$ is a random variable whose probability distribution is taken over the coin flips of $A$; that is, $A(x)$ defines a probability space of output strings. We write $x := A(y)$ or $A(y) = x$ to specify that $x$ is generated by running algorithm $A$ on input $y$.

A *security parameter* is a number that is taken from an infinite subset of the set of positive integers. Security parameters are used to measure the time and space complexity of an algorithm, the binary sizes of algorithm inputs and outputs, success probabilities, and security levels. Although multiple security parameters may be specified for any one protocol or system, for simplicity all the protocols and systems in this book make reference to only a single security parameter, denoted by $k$. The notation $1^k$ denotes $k$ encoded in unary.

For inputs of the same binary size, the possible outputs of an algorithm are all

[1] It would be easy to rephrase our constructions and security reductions in the non-uniform complexity model of Boolean circuits, since they do not hinge on the use of polynomial-size advice strings that cannot be computed in polynomial time.

assumed to be of the same binary size; this can be achieved by the standard practice of padding.

As is common in the cryptographic literature, we measure the *running time* of an algorithm in terms of elementary algebraic operations, typically modular multiplications, instead of in terms of the number of transitions of the Turing machine's read-write head. The terms *feasible* and *infeasible* have the standard complexity-theoretical meaning: feasible computations are those that can be performed in time polynomial[2] in $k$, while infeasibility corresponds to *superpolynomial* running time[3] (which includes *subexponential* and *exponential* running time). Whenever we speak of a polynomial-time algorithm, it may be either deterministic or probabilistic.

A function $f : \mathbb{N} \mapsto [0, 1]$ is *negligible* in $k$ if $f(k)$ is smaller than the inverse of any polynomial in $k$, for all sufficiently large $k$; it is *non-negligible* if there exists a positive integer $c$ such that $f(k)$ is greater than $1/k^c$, for all sufficiently large $k$; and, it is *overwhelming* if $1 - f(\cdot)$ is negligible. (Note that a function can be neither negligible nor non-negligible.) An *intractable* problem is one that cannot be solved in polynomial time with non-negligible probability of success.

The notation $\mathsf{P}_k(E_k)$ denotes the probability of event $E_k$. For probability spaces $S_1, S_2, \ldots$, the notation

$$\mathsf{P}_k(E_k(x_1, x_2, \ldots) \mid x_1 := S_1; x_2 := S_2; \ldots)$$

denotes the probability that the event $E_k(x_1, x_2, \ldots)$ holds when each $x_i$ is chosen, in the given order, from the corresponding probability space $S_i$.

An *expected* polynomial-time algorithm is a probabilistic algorithm whose expected running time (over its coin flips) is polynomial for any input. By running a probabilistic polynomial-time algorithm that has non-negligible success probability $\epsilon(k)$ an expected number of $k/\epsilon(k)$ times, one can construct an expected polynomial-time algorithm that has overwhelming success probability.

A *language* over an alphabet is a subset of the set of all finite strings of symbols from that alphabet. If $A$ is a probabilistic algorithm and $L$ a language, then the collection $\{A(x)\}_{x \in L}$ is an *ensemble* of random variables. Two ensembles $\{A(x)\}_{x \in L}$ and $\{B(x)\}_{x \in L}$ are *perfectly indistinguishable* if, for all $x \in L$, the random variables $A(x)$ and $B(x)$ have the same distribution. They are *statistically indistinguishable* (or almost-perfectly indistinguishable) on $L$ if for all $x \in L$ of binary size $k$ their statistical difference

$$\sum_{\alpha} \Big| \mathsf{P}_k(A(x) = \alpha) - \mathsf{P}_k(B(x) = \alpha) \Big|$$

[2]A function $f(\cdot)$ is polynomial in $k$ if there exists a positive integer $c$ such that $f(k) \leq k^c$ for all sufficiently large $k$.

[3]A function $f(\cdot)$ is superpolynomial in $k$ if for all positive integers $c$, $f(k) \geq k^c$ for all sufficiently large $k$.

is negligible in $k$.[4] Finally, they are *computationally indistinguishable* on $L$ if, for all polynomial-time algorithms $D$ with Boolean output (representing accept or reject), and for all $x \in L$ of binary size $k$,

$$\left| \mathsf{P}_k(D(y) = 1 \mid y := A(x)) - \mathsf{P}_k(D(y) = 1 \mid y := B(x)) \right|$$

is negligible in $k$.

### 2.1.3 Interactive algorithms and protocols

An algorithm may consist of two or more *interactive algorithms* that communicate according to one or more stepwise descriptions called *protocols*. Interacting parties in this book are always modeled as interactive algorithms. Formally, an interactive algorithm is a Turing machine enhanced with a *communication tape* and a read-only *common input tape*. A pair of interactive algorithms share their communication tapes, for exchanging messages, and their common input tapes. The work tape and random tape of each algorithm are private to the algorithm, and their private input tapes enable each to make use of *auxiliary input*, also known as private input; this may be present initially, or computed as protocol executions are taking place.

More generally, there can be any number of interactive algorithms communicating with each other in an arbitrary fashion. This can easily be formalized by endowing each interactive algorithm with a set of communication and common input tapes for any other algorithm it needs to be able to communicate with. Correspondingly, protocols may be defined among more than two parties. In general, any collection of interactive algorithms can be viewed as a single (interactive or non-interactive) algorithm.

With $\mathcal{P}$ and $\mathcal{V}$ denoting two interactive algorithms, ($\mathcal{P}$, $\mathcal{V}$) denotes the protocol performed by them, and $<\mathcal{P}, \mathcal{V}>$ denotes the two algorithms viewed as a single (interactive or non-interactive) algorithm. The input of $<\mathcal{P}, \mathcal{V}>$ is the initial contents of their common input tape, and the output of $<\mathcal{P}, \mathcal{V}>$ is the concatenation of the contents on the output tapes of $\mathcal{P}$ and $\mathcal{V}$. Usually at most one of $\mathcal{P}$ and $\mathcal{V}$ is defined to produce an output.

A *move* in a protocol is a message transfer from one interactive algorithm to another; a *round* is two consecutive moves. After the last move in a protocol, the receiving algorithm checks a (protocol) *verification relation* in order to decide whether to accept or reject. Additional verifications may be applied in intermediate stages. A protocol is said to be *non-interactive* if it consists of a single move, and interactive if it has more. None of the protocols in this book have more than two rounds.

[4]The practical interpretation of this is that an infinitely powerful algorithm that is restricted to taking polynomially many samples cannot distinguish between the two distributions; such an algorithm can infer the same information from either distribution. Note that perfect indistinguishability corresponds to zero statistical difference.

Protocols are sometimes depicted in figures, together with a description of the preliminary stage for setting up keys and other prerequisite information. Within a figure, the actions performed by an interactive algorithm are all displayed in the same column, with a label on top indicating the algorithm that performs the actions. A move is shown in the form of an arrow from the transmitter to the receiver, with the message that is transferred displayed on top of the arrow. Protocol figures should be read from top to bottom. Any additional data that is displayed in a column that contains one or more message transmittals, such as a public key, is also considered known to both communicating parties.

The *transcript* of a protocol executed by two interactive algorithms is the entire ordered sequence of messages exchanged between them until one of them halts.

The *view* of an interactive algorithm in a protocol execution is an ordered set consisting of all the information that the algorithm has "seen" in the protocol execution. This comprises the protocol transcript, as well as the common input, any private input, and its own coin flips (if any). An *accepting view* of an interactive algorithm is the view of that algorithm in a protocol execution in which it accepts. Usually at least one of the algorithms in a protocol will be probabilistic, in which case the view is a random variable defined by the coin flips in the protocol. By parameterizing over all possible protocol executions an ensemble of random variables is obtained, and so we can consider indistinguishability of protocol views.

With $\mathcal{P}$ and $\mathcal{V}$ denoting two interactive algorithms, $\mathcal{V}_{\mathcal{P}(\mathsf{aux1})}(x; \mathsf{aux2})$ denotes $\mathcal{V}$'s output when interacting with $\mathcal{P}$ on common input $x$, auxiliary input $\mathsf{aux1}$ to $\mathcal{P}$ and auxiliary input $\mathsf{aux2}$ to $\mathcal{V}$. When there is no auxiliary input to $\mathcal{V}$, we simply write $\mathcal{V}_{\mathcal{P}(\mathsf{aux})}(x)$. In either case, for fixed inputs this is a random variable, whose probability distribution is taken over the coin flips (if any) of $\mathcal{P}$ and $\mathcal{V}$.

The notation $\mathcal{K}(x; A)$ denotes the output of algorithm $\mathcal{K}$ on input $x$, when having write access to the random tape and the private input tape of algorithm $A$ that in all other respects behaves as a *black box* to $\mathcal{K}$. The standard method for extracting knowledge from an algorithm $A$ is to rewind $A$ for the same tape configurations but different queries. By means of oracle replay in the so-called random oracle model, which we discuss shortly, it may be possible to extract secrets even if rewinding is not an option, even though it is unclear in this case how to properly define knowledge outside the random oracle model. (See Assumption 4.3.9 for an example.)

### 2.1.4 Attack models

An *honest* interactive algorithm does not deviate from its behavior specified in the protocol description, and does not engage in any other actions. In particular, it follows the protocol description in all protocol executions in which it engages.

An *attacker* is an interactive algorithm that may deviate from its prescribed actions, for instance by deviating from its actions in the specified protocols or by wiretapping the protocol executions of others. When assessing the security of a new cryptographic construction, we typically view the subset of all parties that deviate from the prescribed protocol(s) (either cooperative or non-cooperative) as a single algorithm, which we then use as a subroutine for an algorithm to solve a supposedly intractable problem. A collection of attackers that may but need not share all their tapes is referred to as an *adversary*.[5]

In assessing whether a protocol satisfies a property of interest, one must consider not only the computing power of the attackers, but also the flexibility they have in engaging in executions of the protocol. Among the factors contributing to whether an adversary can break a presumed property of a protocol are the following:

- The extent to which multiple executions of the same protocol, or of different protocols, can be interleaved. Parallel executions of a protocol are more efficient than sequential executions, not only in the number of moves but also in that some additional operations (such as line encryption) may be applied once on all the moves that occur concurrently. On the other hand, parallelization may enable an adversary to compute information that it could not compute otherwise.

  The most powerful attack model is that of *arbitrary composition* of protocols or protocol executions; here, the adversary can adaptively (depending on its protocol views in the past) decide at each stage which moves of one protocol execution to interleave with which moves of another protocol execution (not necessarily of the same protocol), and in what manner.

- The number of protocol executions attackers are able to engage in.[6] A *passive* attacker is not allowed to interact at all, but can wiretap protocol executions that take place by honest parties; an *active attacker* is allowed to engage in protocol executions. In all our protocol descriptions in this book honest parties are assumed to be polynomial-time, and so even an infinitely powerful adversary can never engage in more than polynomially many protocol executions.

- The computing power given to the adversary. All the privacy results in this book are proved under the assumption that attackers have infinite computing power. Digital signatures, on the other hand, can be proved unforgeable only against attackers that are polynomially bounded.

[5]The protocol executions of an attacker that operates in isolation may be wiretapped by other attackers and in this manner add to their power; for this reason isolated attackers must be considered part of the adversary. The attempt to circumvent this by encrypting all message transfers is not satisfactory. First, it is unreasonable to characterize parties that do not properly encrypt their own messages as attackers of the system. Second, it would be poor design practice to let the requirement for session encryption interfere with the design of the system core and the analysis of systemic security.

[6]For example, in Section 5.4 we will construct proofs of knowledge in which the prover does not leak any information about its secret key when performing the protocol once, but multiple protocol executions using the same public key leak the entire secret key.

The power of an adversary with unlimited resources is not restricted to computations before or after a protocol execution; at each step, it may use infinite computing time to compute its next message. While this model may seem unrealistic, in the Turing machine model it makes perfect sense: the running time of an interactive algorithm is determined by the number of state transitions defined by the transition function, and so the time taken by one machine does not affect the running time of the other. Moreover, the model of infinite computing power serves as a useful abstraction to model the complete absence of one-way functions (see Section 2.2) and other intractable computational tasks. When a construction is proved invulnerable against an infinitely powerful adversary, there is no need to worry about attackers that have embedded cryptographic trapdoors or have come up with a cryptographic breakthrough.

The first two of these factors are under the control of the honest party that is being attacked. It can determine which protocol executions to perform sequentially and how many executions it engages in.

The *aggregate view* of an interactive algorithm in multiple executions of the same or of different protocols is the collection of its views in the individual protocol executions. The ordering of the components of the aggregate view is naturally defined by the fashion in which the protocol executions are interleaved.

Following Feige, Fiat, and Shamir [168], $\overline{\mathcal{A}}$ denotes an honest interactive algorithm, $\widehat{\mathcal{A}}$ denotes an attacker with polynomially bounded resources, and $\widetilde{\mathcal{A}}$ denotes an attacker with unbounded computing resources. Attack algorithms $\widehat{\mathcal{A}}$ and $\widetilde{\mathcal{A}}$ may but need not deviate from the protocol description, and can engage in as many protocol executions as they desire, confined only by the limitations imposed by the other parties in the protocols they engage in. For instance,

$$\widehat{\mathcal{V}}_{\overline{\mathcal{P}}(\mathsf{aux1})}(x; \mathsf{aux2})$$

denotes $\widehat{\mathcal{V}}$'s output after interacting with $\overline{\mathcal{P}}$, whereby $\widehat{\mathcal{V}}$ can query $\overline{\mathcal{P}}$ as if it were an oracle; in particular, $\widehat{\mathcal{V}}$'s output may be the result of a phase in which $\widehat{\mathcal{V}}$ engages in a plurality of protocol executions. In contrast,

$$\overline{\mathcal{V}}_{\overline{\mathcal{P}}(\mathsf{aux1})}(x; \mathsf{aux2})$$

always refers to a single execution of the protocol.

In analyzing whether a certain property holds for an honest party in a protocol, it is always assumed that the party aborts an execution of the protocol as soon as the other party (parties) deviates from the description of the protocol in a manner that is detectable with certainty by the honest party. The interpretation of "detectable" depends on the verification relations and other actions specified in the description of the protocol, as well as on actions that we always assume implicitly. Specifically, it is implicit in protocol descriptions that the parties to a protocol always apply range-checking to numbers supposed to be in an algebraic structure. For examples of attacks that become possible when a party inadvertently does not apply range checking, see Bleichenbacher [34] and Anderson and Vaudenay [14]. Arithmetic relations that must apply to these numbers (i.e., verification relations) are always mentioned explicitly. In some case it is necessary to perform group membership tests; see Section 2.4.3 for an example. Deviation by the other party from a specified probability distribution, though, does not constitute a reason to abort a protocol execution.

### 2.1.5 Security reductions and the random oracle model

Since the emphasis in this book is on practicality, *exact security* is of importance. Suppose that a problem $P$ is conjectured to be intractable: problem instances of size $k$ cannot be solved with non-negligible success probability $\epsilon(k)$ in fewer than $r_P(k)$ steps, for some superpolynomial running-time function $r_P(\cdot)$. To prove the infeasibility of a cryptanalytic attack on a new protocol construction, we construct an algorithm $A$ that can solve problem instances of size $k$ of problem $P$ in polynomial time, with success probability negligibly close to $\epsilon(k)$, by making no more than polynomially many subroutine calls to a black-box algorithm $B$ that can feasibly solve instances of the new construction. Suppose that, for inputs of size $k$, $B$ runs in time $f_B(k)$, and $A$ makes $f_A(k)$ calls to $B$, using $g_A(k)$ additional processing steps for each call and $h_A(k)$ additional one-time processing steps. The functions $g_A(\cdot)$ and $h_A(\cdot)$ are polynomials, typically of low degree. The total running time $r_{AB}(k)$ of $A$ and $B$ is equal to

$$f_A(k)\Big(f_B(k) + g_A(k)\Big) + h_A(k).$$

If $f_B(\cdot)$ is polynomially bounded there exists a positive integer $k_0$ such that

$$r_{AB}(k) < r_P(k) \quad \forall k \geq k_0,$$

contradicting the presumed intractability of problem $P$. But what practical assurances on the parameter sizes for the new construction can we infer from the proof reduction? We obtain a contradiction only for security parameter sizes that exceed $k_0$, and so it is desirable that $k_0$ be as small as possible. Hereto the functions $f_A(\cdot)$, $g_A(\cdot)$ and $h_A(\cdot)$ should all be as small as possible. In situations of practical interest $f_B(k)$ typically exceeds $h_A(k)$ by far, and so $f_A(\cdot)$ is the dominating contribution to the total running time. We therefore equate the *overhead factor* of a security reduction with the (expected) number of calls to algorithm $B$. A security reduction is *tight* if the overhead factor is a (small) constant, and *optimal* tightness is achieved when the overhead factor is negligibly close 1. For any given security level, a tight reduction allows one to implement the new construction using smaller parameter sizes than would be allowed by a non-tight reduction. Because practicality is an important objective of this book, we strive for tight security reductions throughout. In practice it is recommended to choose $k$ in such a manner that an adversary needs an expected number of at least $2^{80}$ elementary operations to break the construction at hand. (Imagine a supercomputer that can do 1 elementary operation each picosecond, and that 300 of these are running in parallel; it would take over 128 years to cycle through all $2^{80}$ operations.)

Sometimes the security of a new construction can be proved only in the *random oracle model*. In this model, a function that is believed hard to invert may be *idealized* by substituting applications of the function by calls to a random oracle. The oracle, on input an element in the domain of the function, produces a random output in the range of the function, to be interpreted as the outcome of the function. Of course, multiple oracle queries with the same input produce the same output. The success probability of the resulting security reduction is taken over the space of all random functions. Cannetti, Goldreich, and Halevi [76] concocted example constructions that are provably secure in the random oracle model but provably insecure when the oracle is instantiated using any function. Therefore, the ability to give a security reduction in the random oracle model in general does not imply that a successful attack requires the exploitation of a weakness in the function that is being idealized. Nevertheless, in the "natural" cryptographic constructions that arise in practice, and in particular in this book, provable security in the random oracle model is believed to be a relevant measure of security. Most of the constructions in this book are amenable to security proofs in the random oracle model.

## 2.2 One-way functions

### 2.2.1 Definition

From now on we are mainly interested in functions $f(\cdot)$ that can be expressed as an infinite collection of functions, $\{f_i(\cdot)\}_{i\in V}$. Each $f_i(\cdot)$ operates on a finite domain, $D_i$, and $V$ is an enumerable infinite *index set*, with $i$ uniquely specifying $D_i$. A description of $f(\cdot)$ entails specifying $V$ and the mapping $f_i(\cdot)$. Any collection of functions $\{f_i(\cdot)\}_{i\in V}$ can be represented by a single function $f(\cdot)$ that operates on an infinite domain, by defining

$$f(i,x) := (i, f_i(x)),$$

and so the two notions can be used interchangeably.

Informally, a one-way function is a function that is easy to compute, but computing inverses is difficult on average. To formalize this notion, we require an *instance generator* for the function. An instance generator for a collection of functions, $\{f_i(\cdot)\}_{i\in V}$, is a pair $(I, D)$ of probabilistic polynomial-time algorithms, operating as follows:

- $I$ takes as input $1^k$, and outputs an index $i \in V$ of binary size $l(k)$, where $l(\cdot)$ is a fixed polynomial; and

- $D$ takes as input the output $i$ of $I$, and outputs an element $x$ in the domain $D_i$ of $f_i(\cdot)$. (Elements in $D_i$ may occur with zero probability as an output of $D(i)$.)

The output of $(I, D)$ is $(i, x)$. We will simply write $(i, x) \in V \times D_i$ to denote that first $i$ is taken from $V$ and then $x$ is taken from $D_i$. (Although this is not the standard Cartesian product, no confusion can arise.)

**Definition 2.2.1.** *A collection of functions, $\{f_i(\cdot)\}_{i \in V}$, is (strongly)* one-way *over an instance generator $(I, D)$ if and only if the following two properties hold:*

1. *(Computable in one direction) There exists a deterministic polynomial-time algorithm that, on input any $(i, x) \in V \times D_i$, outputs $f_i(x)$; and*

2. *(Uninvertable in the other direction) For any polynomial-time algorithm $A$, the probability function defined by*

$$\mathsf{P}_k\Big(f_i(A(i, f_i(x))) = f_i(x) \mid i := I(1^k); x := D(i)\Big)$$

   *is negligible in $k$.*

Whenever we say that a function $f(\cdot)$ is one-way, we mean that the collection of functions that it represents is one-way, in the above sense. In case the instance generator is clear from the context, we will not mention it explicitly.

Note that functions with superpolynomial range are always one-way when idealized in the random oracle model.

If $\{f_i(\cdot)\}_{i \in V}$ is one-way over $(I, D)$, then $(I, D)$ is said to be an *invulnerable* instance generator for $\{f_i(\cdot)\}_{i \in V}$. The usefulness of the notion of invulnerability (originating from Abadi, Allender, Broder, Feigenbaum, and Hemachandra [1]) becomes apparent when reducing the problem of inverting a function that is conjectured to be one-way to the problem of breaking a new construction, to prove the security of the latter. Instead of specifying a particular instance generator for the conjectured one-way function, it is sometimes possible to make the same security reduction work for all invulnerable instance generators; this makes the resulting security statement much stronger. All the security reductions in this book are of this form. Note that if an instance generator is invulnerable for a function, then so are all instance generators with a computationally indistinguishable output distribution.

A *commitment function* enables a sender to commit to a secret, in such a manner that the receiver cannot determine the secret until the sender *opens* the commitment, while the sender cannot open the commitment to reveal a different secret than that originally committed to. A one-way permutation can serve as a commitment function that is unconditionally secure for the receiver and computationally secure for the sender. Hereto the sender embeds its secret into the hard-core bits[7] of an otherwise randomly chosen argument $x$ to a one-way permutation $f(\cdot)$.

We now discuss two well-known functions that are widely believed to be one-way. Their one-wayness is crucial to the security of all the constructions in this book.

### 2.2.2 The DL function

The output of an instance generator $(I, D)$ for a *DL function*, defined below, satisfies the following format:

- On input $1^k$, with $k \geq 2$, $I$ generates a pair $(q, g)$, satisfying the following properties:
  - $q$ is a prime number of binary size $k$ that uniquely specifies a group of order $q$, from now on denoted by $G_q$. Without loss of generality, the group operation is assumed to be multiplication.
  - $g$ is a generator of $G_q$.
- On input $(q, g)$, $D$ generates an element $x$ in $\mathbb{Z}_q$.

A DL function is a collection of functions, $\{f_i(\cdot)\}_{i \in \{(q,g)\}}$, defined as follows:

$$f_{q,g} : x \mapsto g^x.$$

The number $x$ is called the *discrete logarithm* of $g^x$ with respect to $g$. Note that different algebraic constructions for $G_q$ give rise to different DL functions.

Under what conditions is a DL function one-way? If the construction of $G_q$ is such that multiplication in $G_q$ is feasible, then $f_{q,g}(x)$ can be feasibly computed by means of repeated squaring (see Menezes, van Oorschot, and Vanstone [266, Section 14.6]), possibly in combination with additional preprocessing.[8] The hardness of inverting a DL function depends on the construction of $G_q$ and on the instance generator $(I, D)$. The following two constructions are widely believed to give rise to a one-way DL function.

**(Subgroup construction)** $G_q$ is a subgroup of $\mathbb{Z}_p^*$, where $p$ is a prime such that $q \mid (p-1)$ and $|p|$ is polynomial in $k$. The following instance generator is believed to be invulnerable:

[7]If $f(\cdot)$ is one-way, then there must exist a Boolean predicate of the bits of the argument of $f(\cdot)$ that is at least somewhat hard to compute when given $f(x)$. A Boolean predicate $b(\cdot)$ is said to be hard-core for $f(\cdot)$ if $b(\cdot)$ can be computed in polynomial time, but no probabilistic polynomial-time algorithm can compute $b_i(x)$ from $f_i(x)$ with success probability non-negligibly greater than $1/2$. More generally, several bits are said to be (simultaneously) hard-core for $f(\cdot)$ if no polynomial-time algorithm can extract any information about these bits of $x$ when given $f_i(x)$.

[8]Alternatively, addition chains or other techniques may be used. For overviews and comparisons of exponentiation methods, see Knuth [232, Subsection 4.6.3], Menezes, van Oorschot, and Vanstone [266, Chapter 14.6], von zur Gathen and Nöcker [383], Gordon [196], and O'Connor [280].

- $q$ is generated at random from the set of all primes of binary size $k$, using a primality test. The number $p$ is the first prime in the sequence $aq + 1$, for successive (even) integer values $a$, starting from a fixed positive integer $a_0$ that is "hard-wired" into $I$; heuristic evidence suggests that only polynomially many values of $a$ need to be tested (see Wagstaff [384]). It is not known how to test primality in deterministic polynomial time. A polynomial-time primality test with negligible error probability may be used instead, though, because its outputs are computationally indistinguishable.
- $g$ is generated at random from $G_q \setminus \{1\}$. This can be accomplished by taking $g := f^{(p-1)/q}$, for a random $f \in \mathbb{Z}_p^*$, and testing that $g \neq 1$. (There is only one subgroup with order $q$.) Other distributions are not necessarily improper, but results of Bleichenbacher [34] and Anderson and Vaudenay [14] for groups of non-prime order suggest that a random selection is preferred.
- $x$ is best chosen at random from $\mathbb{Z}_q$. This maximizes its entropy, and allows one to prove the hardness of inverting assuming the seemingly weaker assumption that any polynomial-time algorithm for inverting the DL function has non-negligible error probability. In particular, either this choice is successful or the DL-function cannot be one-way at all.

Alternatively, one first generates a random prime $p$ and checks whether $q = (p-1)/a_0$ is a prime, repeating this process until a prime $q$ is found. Another variation is to generate a random composite with known prime factorization (see Bach [16]), and to test whether its increment by one is a prime, $p$; if this is the case, then with high probability the binary size of the largest prime factor of $p - 1$ is proportional to $|p|$. (In this case $|q|$ should be allowed to be linear in $k$.)

Primes $p$ of a special form may provide even better protection against attacks. For instance, it is believed preferable to generate $p$ subject to the restriction that $(p-1)/2q$ contains only prime factors greater than $q$. Other reasons for generating $p$ of a special form are related to security issues of protocols that operate in $G_q$; see Section 2.4.3.

In practical applications it is often important that the pair $(p, q)$ do not contain a trapdoor that enables the rapid computation of discrete logarithms. Although no trapdoor constructions are known in the public literature that cannot be feasibly detected (and, therefore, tested for), confidence can be increased by using an instance generator that generates $q$ and $p$ in a pseudorandom manner, starting from a seed value that must be output as well by the instance generator; see Federal Information Processing Standards no. 186 [277, Appendix 2]. In addition, a succinct proof may be output for proving that $g$ has been generated using a pseudorandom process.

**(Elliptic curve construction)** $G_q$ is an elliptic curve of order $q$ over a finite field. (See Menezes [263] for an introduction to elliptic curves as applied in cryptography.) It is common to take $\mathbb{Z}_p$ as the underlying field, for a prime $p$ such that $|p|$ is polynomial in $k$. The following three instance generators are believed to be invulnerable:

- Select a (probable) prime $p$, and randomly try elliptic curve coefficients until a curve of prime order is found. (The process could be sped up by allowing $|q|$ to be linear in $k$.) The elliptic curve coefficients must be specified as part of the output of the instance generator. Generate $g$ and $x$ at random from $G_q \setminus \{1\}$ and $\mathbb{Z}_q$, respectively.
- Alternatively, the coefficients determining the elliptic curve are hardwired into the instance generator, and $p$ and $q$ of the appropriate form are sought by trial and error. The numbers $g$ and $x$ are generated at random from $G_q \setminus \{1\}$ and $\mathbb{Z}_q$, respectively.
- Another possibility is to generate an elliptic curve of prime order together with a generator $g$, using an algorithm of Koblitz [234]. As before, $x$ is best chosen at random from $\mathbb{Z}_q$.

Alternatively, one can generate elliptic curves over a field of the form $\mathrm{GF}_{2^m}$ instead of over $\mathbb{Z}_p$; as we will see shortly, this offers practical advantages.

Any uncertainty about the presence of a trapdoor can be removed in the manner described for the subgroup construction.

Other constructions have been proposed, but these are considerably more difficult to understand and have not been scrutinized by more than a few experts. See Biehl, Meyer, and Thiel [28] and the references therein for constructions in real-quadratic number fields, and Koblitz [233] for a construction in groups obtained from Jacobians of hyperelliptic curves.

The prime $q$ need not be generated at random, if only the underlying field is chosen in a substantially random manner. A smart choice for $q$ enables one to speed up the reduction modulo $q$. Example choices for $q$ are $2^{127} - 1$ (a Mersenne prime) or, more generally, $2^a \pm b$, for small $b$. Furthermore, in some applications it may be useful to use the same $q$ in combination with multiple primes $p_i$, all chosen at random subject to the condition $q \mid (p_i - 1)$.

With the exception of the technique described in Section 4.4.1, none of the constructions in this book depend on the manner in which $G_q$ is constructed. In general, we merely need the following assumption to be true.

**Assumption 2.2.2.** *There exists a DL function that has an invulnerable instance generator.*

From now on we will use the notation $(I_{\mathrm{DL}}, D_{\mathrm{DL}})$ to denote an invulnerable instance generator for "the" DL function. Although specific embodiments may have additional outputs, such as $p$, elliptic curve coefficients, a compact proof that $g$ has been generated at random, and information evidencing that no trapdoor has been built in, for concreteness we will always assume that the output of $(I_{\text{DL}}, D_{\text{DL}})$ is $(q, g, x)$. We will also assume that $(q, g)$ is always *properly formed*, meaning that $q$ is a prime and $g$ a generator of $G_q$, but will never make any assumptions about the probability distribution of the outputs of $(I_{\text{DL}}, D_{\text{DL}})$.

The one-wayness of a function is an asymptotic property. For overviews of algorithms to compute discrete logarithms, see Odlyzko [281] and McCurley [259]. In practice, the binary size of the parameters must be selected such that adequate security for the application at hand is offered. For instance, the parameter sizes for a digital signature that is to have legal meaning several decades from now must be much greater than for an identification protocol that only seeks to withstand replay attacks. Currently recommended parameter sizes for the DL function are as follows:

**(Subgroup construction)** To compute discrete logarithms in $G_q$, one can either work in $G_q$ or proceed indirectly[9] by computing discrete logarithms with respect to generators of $\mathbb{Z}_p^*$.

The best known algorithms for computing discrete logarithms in $\mathbb{Z}_p^*$ all have subexponential running time. In May 1998, Joux and Lercier computed a discrete logarithm modulo a $299$-bit prime using a network of Pentium PRO $180$ MHz personal computers and a total of 4 months of CPU time. Odlyzko [282] states that primes $p$ should be at least $1024$ bits even for moderate security, and at least $2048$ bits for anything that should remain secure for a decade.[10] This recommendation is based on the presumption that future progress in algorithmic and hardware capabilities will be along the lines witnessed in the past. According to Silverman [353], it has been estimated that for large $k$, breaking a discrete logarithm of $k - 30$ bits takes about the same time as factoring a $k$-bit composite that is the product of two random primes of approximately equal binary size, but Odlyzko [283] recommends to ignore this difference when choosing parameter sizes.

In $G_q$ itself, only exponential-time algorithms are known, with running time $O(\sqrt{q})$. Shoup [352] proved in his so-called generic string encoding model that algorithms with a better performance than $O(\sqrt{q})$ operations must make use of the structure of $G_q$, suggesting that a universal subexponential-time inverting algorithm that works for any construction of $G_q$ cannot exist. Assuming that the best algorithms for computing discrete logarithms in $G_q$ have running time $O(\sqrt{q})$, a $160$-bit prime $q$ offers the same security level as a $1024$-bit $p$; see Menezes [264]. Odlyzko [282] recommends primes $q$ of $200$ bits for

[9]With $(p-1)/q$ polynomial in $k$, the infeasibility of computing discrete logarithms in $G_q$ follows from that in $\mathbb{Z}_p^*$.

[10]This overturns an earlier recommendation of Odlyzko [283] to use $1024$-bit primes for long-term security (i.e., at least the next two decades) and $768$-bit primes for medium-term security.

long-term security. Lenstra and Verheul [247] are more optimistic; for security until the year 2020 they recommend using primes $p$ of at least 1881 bits and primes $q$ of at least 151 bits.

**(Elliptic curve construction)** As with the subgroup construction, the fastest methods known for computing discrete logarithms in $G_q$ require $O(\sqrt{q})$ steps. When so-called supersingular curves are used, one can invert the DL function indirectly (by computing discrete logarithms in the underlying field $\mathbb{Z}_p$or an extension of it) in subexponential time, and the binary size of $p$ must be comparable to that in the subgroup construction; see Menezes, Vanstone, and Okamoto [265]. A linear-time algorithm for so-called trace-1 elliptic curves was announced in September 1997 independently by Smart [357] and by Satoh and Araki [335]. Both cases occur with negligible probability for randomly chosen curves, and can easily be detected. For randomly generated curves only exponential-time algorithms are known, taking $O(\sqrt{p})$ steps. Barring algorithmic breakthroughs, numbers in the base and numbers in the ring of exponents can therefore be taken of the same binary size; this is a huge efficiency improvement over the subgroup construction.

In May 1998, a Certicom elliptic curve challenge over a field $\mathbb{Z}_p$ with a $97$-bit prime $p$ was solved after 53 days of distributed computation using more than $1200$ computers from at least $16$ countries. In a whitepaper [85], Certicom estimates that a $160$-bit prime $p$ offers the same security as factoring a $1024$-bit composite, and that a $210$-bit prime $p$ compares with factoring a $2048$-bit composite. More recently, Lenstra and Verheul [247] estimated that a 139-bit $p$ and a 1024-bit composite or a 160-bit $p$ and a 1375-bit composite offer computationally equivalent security. They estimate that, for security until the year 2020, key sizes of at least 161 bits should be used if no cryptanalytic progress is expected, and at least 188 bits to “obviate any eventualities.”

Assuming again that subexponential-time inverting algorithms do not exist, working over a field of the form $\text{GF}_{2^m}$ offers significant advantages. VLSI circuits have been designed that can rapidly perform operations in these fields; see Agnew, Mullin, and Vanstone [6] for a VLSI implementation of $\text{GF}_{2^{155}}$. Even in general software environments, the use of $\text{GF}_{2^m}$ offers performance advantages over $\mathbb{Z}_p$. Wiener [390] estimates that $m$ should be in the range 171–180 to make computing discrete logarithms as hard as factoring $1024$-bit composites. In his analysis, Wiener assumes the strongest attack known: a parallel collision search attack using a fully pipelined chip for elliptic curve additions over $\text{GF}_{2^m}$. In April 2000, a team led by Harley, Doligez, de Rauglaudre, and Leroy broke an elliptic curve challenge for $m = 108$ after four months of distributed computation using $9500$ computers; the estimated workload for solving a 163-bit challenge is about 100 million times larger.

The belief in the strength of short moduli for the elliptic curve construction is not ubiquitous. Odlyzko [282] warns that "it might be prudent to build in a considerable safety margin against unexpected attacks, and use key sizes of at least $300$ bits, even for moderate security needs." Several renowned cryptographers have even expressed disbelief that the complexity of the discrete logarithm problem for elliptic curves is more than subexponential.

### 2.2.3 The RSA function

The output of an instance generator $(I, D)$ for the *RSA function*, defined below, satisfies the following format:

- On input $1^k$, with $k \geq 4$ and $k$ even, algorithm $I$ generates a pair $(n, v)$, satisfying the following properties:
  - $n$ is the product of binary size $k$ of two primes, $p$ and $q$.
  - $v$ is a number smaller than $n$ that is co-prime to $\varphi(n)$, where $\varphi(\cdot)$ is Euler's phi-function.
- On input $(n, v)$, algorithm $D$ generates an element $x$ in $\mathbb{Z}_n^*$.

The *RSA function* is a collection of functions, $\{f_i(\cdot)\}_{i \in \{n,v\}}$, defined as follows:

$$f_{n,v} : x \mapsto x^v.$$

Under what conditions is the RSA function one-way? Clearly, the RSA function can be evaluated efficiently, using repeated squaring or addition chain techniques. Note that the exponent is fixed instead of the base number; this makes repeated squaring less and addition chains more attractive than in the case of the DL function. The following instance generator is believed to be invulnerable:

- $p$ and $q$ are chosen at random[11] from the set of primes of binary size $k/2$, using a (probabilistic) primality test. In case $p$ and $q$ are generated after $v$ has been determined, a test for $\gcd(\varphi(n), v) = 1$ can be used to decide whether to keep $(p, q)$ or to repeat the experiment.

- $v$ can be chosen in an almost arbitrary fashion, including an invariant choice hard-wired into $I$. Certain choices must be avoided, such as $(n, v)$ such that $v^{-1} \bmod \varphi(n) < n^{0.292}$ (see Wiener [389] and Boneh and Durfee [36]), but bad choices are believed to be detectable and so they can be easily avoided. In fact, they should occur with negligible probability if $v$ and $n$ are generated independently at random.

[11]Rivest and Silverman [326] argue that for practical purposes using random primes is as secure as using primes of a special form.

- $x$ is best chosen at random from $\mathbb{Z}_n^*$, for the same reasons as with the DL function. In particular, either this choice is successful or the RSA function cannot be one-way at all.

Other methods for generating $(n, v)$ have been proposed, all differing in the distribution according to which $p$ and $q$ are generated; see, for example, Boneh [37] and Kaliski and Robshaw [225].

Although $v$ may be an arbitrary small constant or a composite, in the rest of this book we are interested only in primes $v$ that are superpolynomial in $k$. The reasons for this will become apparent in Section 2.4.4 and in the next two chapters.

**Assumption 2.2.3.** *There exists an invulnerable instance generator for the RSA function that outputs primes $v$ that are superpolynomial in $k$.*

Boneh and Venkatesan [40] proved that breaking RSA with small $v$ cannot be equivalent to factoring $n$ under algebraic reductions unless factoring is easy, but their result does not apply to superpolynomial $v$. Consequently, it may well be the case that there exists an invulnerable instance generator such that inverting the RSA function is as hard as factoring.

From now on we use the notation $(I_{\text{RSA}}, D_{\text{RSA}})$ to denote an invulnerable instance generator for the RSA function that outputs primes $v$ superpolynomial in $k$. Its output is $(n, v, x)$, and we will always assume that $(n, v)$ is always *properly formed*, meaning that $v$ is a prime that is co-prime to $\varphi(n)$. In practice, additional outputs may be specified, such as a succinct proof demonstrating that $(n, v)$ is properly formed. (In the applications in this book, the proof that $v$ is co-prime to $\varphi(n)$ will always be implicitly given by the party that generates $v$, as a side consequence of its protocol executions; see Section 4.2.3.) Moreover, in Sections 4.2.2 and 4.4.2 we will construct protocols on the basis of an invulnerable instance generator that provides the prime factorization of $n$ as "side information." We will never make any assumptions about the probability distribution of the outputs of $(I_{\text{RSA}}, D_{\text{RSA}})$.

The fastest known algorithms for inverting the RSA function all proceed by factoring $n$; for an overview, see Bressoud [61]. They have subexponential running time and have been used successfully to factor composites of up to 512 bits. [12] Shamir's TWINKLE device [345] brings 512-bit moduli within reach of a single device. According to Odlyzko [282], "even with current algorithms, within a few years it will be possible for covert efforts (involving just a few people at a single institution, and thus not easily monitored) to crack $768$ bit RSA moduli in a year or so." Lenstra and Shamir [246] estimate that it would take 5000 TWINKLE devices connected by a fast network to 80 000 standard Pentium II computers in order to factorize a $768$ bit composite within 6 months. Odlyzko [282] projects that "even $1024$ bit RSA moduli

[12] RSA-155, which has 512 bits and is the product of two 78-digit primes, was factored in August 1999 using the Number Field Sieve algorithm. The effort used the equivalent of roughly 8000 mips years, and involved 292 desktop computers and a Cray C916 supercomputer. See Cavallar et al. [77] for details.

might be insecure for anything but short-term protection.” Lenstra and Verheul [247] recommend using RSA moduli of at least 1881 bits for security until the year 2020, and Odlyzko [282] recommends to use at least $2048$-bit moduli. Silverman [353], however, argues that these estimates are highly unrealistic on the grounds that taking the total number of computing cycles on the Internet as a model of available computing power ignores memory and accessibility problems. In particular, he strongly disagrees with the conclusion of Lenstra and Verheul [247] that $1024$ bit moduli are insecure after 2002, and estimates that $1024$ bit moduli will remain secure for at least 20 years and $768$ bit moduli for perhaps another 10 years.

The requirement that $v$ be superpolynomial can be met in practice by taking the binary size of $v$ similar to that of $q$ in the DL function; at least $160$ bits is recommended, and $200$ bits is preferable. As in the case of $q$, $v$ need not be chosen at random. In particular, a smart choice for $v$ enables a faster reduction modulo $v$.

## 2.3 Collision-intractable functions

### 2.3.1 Definition

In practical applications, it is often required of a function that it be infeasible to compute two arguments that are mapped to the same outcome. Formally:

**Definition 2.3.1.** *A collection of functions, $\{f_i(\cdot)\}_{i \in V}$, is* collision-intractable *over an instance generator $(I, D)$ if and only if the following two properties hold:*

1. *(Computable in one direction) There exists a deterministic polynomial-time algorithm that, on input any $(i, x) \in V \times D_i$, outputs $f_i(x)$; and*
2. *(Collision-intractable in the other direction) For any polynomial-time algorithm $A$, the probability function defined by*

$$\mathsf{P}_k\Big(A(i) = (x, y) \text{ such that } x, y \in D_i,\ x \neq y,\ f_i(x) = f_i(y) \mid i := I(1^k)\Big)$$

   *is negligible in $k$.*

Note that the coin flips of algorithm $D$ are irrelevant. We may therefore say that the function is collision-intractable over $I$.

For many-to-one functions, collision-intractability is *non-trivial* and is a stronger property than one-wayness. Note that functions with superpolynomial range are always collision-intractable when idealized in the random oracle model.

A non-trivial collision-intractable function $f(\cdot)$ can serve as a commitment function that is unconditionally secure for the sender and computationally secure for the receiver. The straightforward implementation whereby the sender commits to $x$ by sending $f_i(x)$ is unsatisfactory, though: the distribution of $x$ in general will differ from the distribution for which one-wayness is guaranteed, and even if it is the same there is no guarantee that no partial information leaks. This problem can be fixed by using a function $f(\cdot)$ with special uniformity properties. (We omit a formal definition, because it is irrelevant for the purposes of this book.) The candidate collision-intractable functions that will be introduced in the next two sections meet these properties and are at the heart of all the constructions in the remainder of this book.

### 2.3.2 The DLREP function

We refer to the collection of functions considered in this section as the *DLREP function*. The output of an instance generator $(I, D)$ for the DLREP function satisfies the following format:

- On input $1^k$, with $k \geq 2$, algorithm $I$ generates a tuple

  $$(q, g_1, \ldots, g_l)$$

  satisfying the following properties:

  - $q$ is a prime number of binary size $k$ that uniquely specifies a group $G_q$ of order $q$.
  - $g_1, \ldots, g_l$ are elements of $G_q$, for an integer $l \geq 1$, and $g_l \neq 1$. The integer $l$ can be hard-wired into $I$, but may also be determined by $I$ itself, depending on its input; in the latter case $l$ may be polynomial in $k$. (Since $l$ can be inferred from the output of $(I, D)$, it is not made explicit in $I$'s output.)

- On input $(q, g_1, \ldots, g_l)$, algorithm $D$ generates a tuple

  $$(x_1, \ldots, x_l),$$

  with $x_1, \ldots, x_l \in \mathbb{Z}_q$.

A DLREP function is a collection of functions, $\{f_i(\cdot)\}_{i \in \{q, g_1, \ldots, g_l\}}$, defined as follows:

$$f_{q,g_1,\ldots,g_l} : (x_1, \ldots, x_l) \mapsto \prod_{i=1}^{l} g_i^{x_i},$$

with domain $(\mathbb{Z}_q)^l$. The tuple $(x_1, \ldots, x_l)$ is called a *DL-representation* of $h := \prod_{i=1}^{l} g_i^{x_i}$ *with respect to* $(g_1, \ldots, g_l)$. The tuple $(0, \ldots, 0)$ is a DL-representation of 1 with respect to any tuple $(g_1, \ldots, g_l)$; we call this the *trivial* DL-representation. We simply call $(x_1, \ldots, x_l)$ a DL-representation of $h$ in case $(g_1, \ldots, g_l)$ is clear from the context. Note that we do not require the $g_i$'s to be generators, in contrast to $g$ in the DL function, nor do we require the $g_i$'s to be different from one another.

The DLREP function is a generalization of the DL function. As with the DL function, different constructions for $G_q$ give rise to different DLREP functions. The following construction is of special importance to our later constructions in this book.

**Construction 2.3.2.** *Given an invulnerable instance generator* $(I_{\text{DL}}, D_{\text{DL}})$ *for the DL function, construct an instance generator* $(I_{\text{DLREP}}, D_{\text{DLREP}})$ *for the DLREP function as follows:*

- *On input* $1^k$*, with* $k \geq 2$*,* $I_{\text{DLREP}}$ *calls* $I_{\text{DL}}$*, on input* $1^k$*, to obtain a pair* $(q, g)$*.* $I_{\text{DLREP}}$ *generates* $l-1$ *exponents,* $y_1, \ldots, y_{l-1}$*, at random from* $\mathbb{Z}_q$*, and computes* $g_i = g^{y_i}$*, for all* $i \in \{1, \ldots, l-1\}$*.* $I_{\text{DLREP}}$ *sets* $g_l := g$*, and outputs* $q, (g_1, \ldots, g_l)$*. (Alternatively, if the construction of* $G_q$ *is such that it is easy to generate random elements from* $G_q$ *without knowing an element in* $G_q$*,* $I_{\text{DLREP}}$ *may generate random* $g_1, \ldots, g_{l-1}$ *directly.)*

- $D_{\text{DLREP}}$ *generates* $x_l$ *at random from* $\mathbb{Z}_q$*. The other elements,* $x_1, \ldots, x_{l-1}$*, may all be generated in an arbitrary manner.*

**Proposition 2.3.3.** *If* $(I_{\text{DL}}, D_{\text{DL}})$ *is invulnerable, then the DLREP function is one-way and collision-intractable over* $(I_{\text{DLREP}}, D_{\text{DLREP}})$*.*

*Proof.* The DLREP function is easy to compute, using $l$ exponentiations and $l-1$ multiplications. (More efficient methods are discussed shortly.)

The DLREP function is trivially one-way if $l = 1$. For the case $l \geq 2$, note that if $(x_1, \ldots, x_l)$ is a DL-representation of $h \in G_q$ with respect to $(g_1, \ldots, g_l)$, then $x := \sum_{i=1}^{l-1} x_i y_i + x_l \bmod q$ is the discrete logarithm of $h$ with respect to $g$; therefore, an efficient algorithm for inverting the DL function can be constructed from one for inverting the DLREP function.

Collision-intractability is vacuously true for the case $l = 1$. Consider now the case $l \geq 2$. If $(x_1, \ldots, x_l)$ and $(y_1, \ldots, y_l)$ are any two different DL-representations of the same number, then $(x_1 - y_1 \bmod q, \ldots, x_l - y_l \bmod q)$ is a non-trivial DL-representation of 1. Consequently, if we can find collisions then we can find a non-trivial DL-representation of 1, at virtually no overhead. We may therefore assume that we are given an algorithm $B$ that, on input $(q, (g_1, \ldots, g_l))$ generated by $I_{\text{DLREP}}$, outputs a non-trivial DL-representation of 1 in at most $t$ steps, with success probability $\epsilon$. We construct an algorithm $A$ that, on input $((q, g), h)$, computes $\log_g h \bmod q$, as follows:

**Step 1.** $A$ generates $2l-2$ random numbers, $r_1, \ldots, r_{l-1}, s_1, \ldots, s_{l-1} \in \mathbb{Z}_q$. $A$ sets

$$g_i := h^{r_i} g^{s_i} \quad \forall i \in \{1, \ldots, l-1\},$$

and $g_l := g$. $A$ then feeds $(q, (g_1, \ldots, g_l))$ to $B$.

**Step 2.** $A$ receives $(x_1, \ldots, x_l)$ from $B$, and checks whether or not it is a non-trivial DL-representation of 1. If it is not, then $A$ halts.

**Step 3.** If $\sum_{i=1}^{l-1} r_i x_i = 0 \bmod q$, then $A$ halts.

**Step 4.** $A$ computes

$$-(\sum_{i=1}^{l-1} s_i x_i + x_l)(\sum_{i=1}^{l-1} r_i x_i)^{-1} \bmod q,$$

and outputs the result.

It is easy to verify that the output in Step 4 is equal to $\log_g h \bmod q$, and that the total running time is $O(l\,k)$ plus the running time of $B$. We now determine the success probability of $A$.

Because the joint distribution of $(g_1, \ldots, g_l)$, generated in Step 1, is the same as that induced by the output of $I_{\text{DLREP}}$, the transition from Step 2 to Step 3 occurs with probability $\epsilon$. To determine the probability that the transition from Step 3 to Step 4 takes place, we observe that there exists an integer $j \in \{1, \ldots, l-1\}$ such that $x_j \neq 0 \bmod q$ (because $B$'s output is non-trivial). Therefore, there are exactly $q^{l-2}$ "bad" tuples $(r_1, \ldots, r_{l-1}) \in (\mathbb{Z}_q)^{l-1}$, for which $\sum_{i=1}^{l-1} r_i x_i = 0 \bmod q$: for any choice of $(r_1, \ldots, r_{j-1}, r_{j+1}, \ldots, r_{l-1})$, the remaining number, $r_j$, exists and is uniquely determined because $q$ is a prime. The tuple $(r_1, \ldots, r_{l-1})$ is unconditionally hidden from $B$, owing to the randomness of the $s_i$'s and the fact that $g$ is a generator of $G_q$, and it is therefore independent of $B$'s output. Any choice of $(r_1, \ldots, r_{l-1})$ is equally likely to have been made by $A$, and so the probability of having chosen a bad tuple is $q^{l-2}/q^{l-1} = 1/q$.

Because Step 4 takes place only if Step 3 is successful, and Step 3 takes place only if Step 2 is successful, the overall success probability of $A$ is $\epsilon(1 - 1/q)$. □

The proof reduction is optimally tight. Because the influence of $l$ on the running time overhead is merely linear in the security parameter, using the parameter sizes recommended for the DL function to implement the DLREP function results in roughly the same security level. In particular, a prime $q$ of $200$ bits should offer long-term protection against collision-finding attempts.

Note that Proposition 2.3.3 also applies if $g_l$ is generated at random; Construction 2.3.2 is simply more general.

The number $x_l$ need not be generated at random. Construction 2.3.2, however, is all we need for our purposes in this book. From now on, $(I_{\text{DLREP}}, D_{\text{DLREP}})$ always denotes an invulnerable instance generator that has been constructed from an invulnerable instance generator $(I_{\text{DL}}, D_{\text{DL}})$ for the DL function in the manner of Construction 2.3.2. Of course, it is permitted to use any instance generator with a distribution that is indistinguishable from that generated by $(I_{\text{DLREP}}, D_{\text{DLREP}})$.

The constructed DLREP function can be used by a sender to commit to $l-1$ attributes, $(x_1, \ldots, x_{l-1})$. As we will show in Chapter 3, this commitment function has a special property: the sender can *gradually open* its commitment to an infinitely powerful receiver, and in intermediate stages demonstrate all sorts of properties about the attributes without leaking additional information about them.

The method described in the proof of Proposition 2.3.3 for computing the DLREP function is polynomial-time, but is not very practical for large $l$ in case (some of) the $x_i$'s are large or randomly chosen. For $l \geq 2$, one can evaluate $\prod_{i=1}^{l} g_i^{x_i}$ much more efficiently by using simultaneous repeated squaring with a single precomputed table whose $2^l - 1$ entries consist of the products of the numbers in the non-empty subsets of $\{g_1, \ldots, g_l\}$. See Knuth [232, Exercises 27 and 39 of Section 4.6.3].

Several variations and optimizations of this basic technique exist. For example, one can process $t > 1$ exponent bits at once; the size of the precomputed table then increases by a factor close to $2^{(t-1)l}$, while the number of multiplications decreases by a factor of $t$ and the number of squarings remains unaffected. For large $l$ one can break up the computation into a number of blocks: with $1 < j \leq l$, the product $\prod_{i=1}^{l} g_i^{x_i}$ can be computed using $d = \lceil l/j \rceil$ precomputed tables, using simultaneous repeated squaring for each of the $d$ subproducts and multiplying the subproduct results.

Alternatively, one can apply vector addition chain techniques; see, for instance, Coster [121].

### 2.3.3 The RSAREP function

We refer to the collection of functions considered in this section as the *RSAREP function*. The output of an instance generator $(I, D)$ for the RSAREP function satisfies the following format:

- On input $1^k$, with $k \geq 4$ and $k$ even, algorithm $I$ generates a tuple

  $$(n, v, g_1, \ldots, g_l)$$

  satisfying the following properties:

  – $n$ is the product of binary size $k$ of two primes, $p$ and $q$.

  – $v$ is a prime smaller than $n$ that is co-prime to $\varphi(n)$.

  – $g_1, \ldots, g_l$ are elements of $\mathbb{Z}_n^*$, for an integer $l \geq 0$. The integer $l$ can be hard-wired into $I$ but may also be determined by $I$ itself, depending on its inputs; in the latter case $l$ may be polynomial in $k$.

- On input $(n, v, g_1, \ldots, g_l)$, algorithm $D$ generates a tuple

  $$(x_1, \ldots, x_{l+1}),$$

  with $x_1, \ldots, x_l \in \mathbb{Z}_v$ and $x_{l+1} \in \mathbb{Z}_n^*$.

The RSAREP function is a collection of functions, $\{f_i(\cdot)\}_{i \in \{n,v,g_1,\ldots,g_l\}}$, defined as follows:

$$f_{n,v,g_1,\ldots,g_l} : (x_1, \ldots, x_l, x_{l+1}) \mapsto \prod_{i=1}^{l} g_i^{x_i} x_{l+1}^v,$$

with domain $(\mathbb{Z}_v)^l \times \mathbb{Z}_n^*$. The tuple $(x_1, \ldots, x_l, x_{l+1})$ is an *RSA-representation* of $h := \prod_{i=1}^{l} g_i^{x_i} x_{l+1}^v$ *with respect to* $(g_1, \ldots, g_l, v)$. The tuple $(0, \ldots, 0, 1)$ is an RSA-representation of $1$ with respect to any tuple $(g_1, \ldots, g_l, v)$; we call this the *trivial* RSA-representation. We simply call $(x_1, \ldots, x_l, x_{l+1})$ an RSA-representation of $h$ in case $(g_1, \ldots, g_l, v)$ is clear from the context. Note that we do not require the $g_i$'s to be different or to have large order.

While a tuple $(y_1, \ldots, y_l, y_{l+1}) \in \mathbb{Z}^l \times \mathbb{Z}_n^*$ satisfying

$$h = \prod_{i=1}^{l} g_i^{y_i} y_{l+1}^v$$

is not an RSA-representation in case one of $y_1, \ldots, y_l$ is not in $\mathbb{Z}_v$, it is easy to check that the *normalized* form

$$(y_1 \bmod v, \ldots, y_l \bmod v, \prod_{i=1}^{l} g_i^{y_i \operatorname{div} v} y_{l+1})$$

is an RSA-representation of $h$. In practice, it may sometimes be more efficient to use $(y_1, \ldots, y_l, y_{l+1})$ directly in a computation, instead of first normalizing it; this avoids one multi-exponentiation. On the other hand, normalization is desirable for the purpose of implementing simultaneous repeated squaring or related techniques.

The following construction is of special importance to our later constructions in this book.

**Construction 2.3.4.** *Given an invulnerable instance generator $(I_{\text{RSA}}, D_{\text{RSA}})$ for the RSA function, construct an instance generator $(I_{\text{RSAREP}}, D_{\text{RSAREP}})$ for the RSAREP function as follows:*

- *On input $1^k$, with $k \geq 4$ and $k$ even, $I_{\text{RSAREP}}$ calls $I_{\text{RSA}}$, on input $1^k$, to obtain a pair $(n, v)$. $I_{\text{RSAREP}}$ generates $l$ random numbers, $g_1, \ldots, g_l$, from $\mathbb{Z}_n^*$, and outputs $n, v, (g_1, \ldots, g_l)$.*
- *$D_{\text{RSAREP}}$ generates $x_{l+1}$ at random from $\mathbb{Z}_n^*$. The other elements, $x_1, \ldots, x_l$, may all be generated from $\mathbb{Z}_v$ in an arbitrary manner.*

**Proposition 2.3.5.** *If $(I_{\text{RSA}}, D_{\text{RSA}})$ is invulnerable, then the RSAREP function is one-way and collision-intractable over $(I_{\text{RSAREP}}, D_{\text{RSAREP}})$.*

*Proof.* The RSAREP function is easy to compute, using $l+1$ exponentiations and $l$ multiplications. (For practicality, one can apply the techniques mentioned in Section 2.3.2 to $\prod_{i=1}^{l} g_i^{x_i}$, and multiply $x_{l+1}^{v}$ into the result.)

The RSAREP function is trivially one-way if $l=0$. For the case $l \geq 1$, note that from an RSA-representation of $h \in \mathbb{Z}_n^*$ with respect to $(g_1, \ldots, g_l, v)$ it is easy to compute the $v$-th root of $h$, assuming one knows the $v$-th root of each $g_i$; from this observation it is easy to see how an efficient inverting algorithm for the RSA function can be constructed from one from the RSAREP function.

Collision-intractability is vacuously true for the case $l=0$. Consider now the case $l \geq 1$. If $(x_1, \ldots, x_l, x_{l+1})$ and $(y_1, \ldots, y_l, y_{l+1})$ are any two different RSA-representations of the same number, then

$$(x_1 - y_1 \bmod v, \ldots, x_l - y_l \bmod v, \prod_{i=1}^{l} g_i^{(x_i - y_i) \operatorname{div} v} x_{l+1} y_{l+1}^{-1})$$

is a non-trivial RSA-representation of 1. Consequently, if we can find collisions then we can find a non-trivial RSA-representation of 1, at modest cost. We may therefore assume that we are given an algorithm $B$ that, on input $(n, v, (g_1, \ldots, g_l))$ generated by $I_{\text{RSAREP}}$, outputs a non-trivial RSA-representation of 1 in at most $t$ steps, with success probability $\epsilon$. We construct an algorithm $A$ that, on input $((n, v), h)$, computes $h^{1/v}$, as follows:

**Step 1.** $A$ generates $l$ random numbers, $r_1, \ldots, r_l \in \mathbb{Z}_v$, and $l$ random numbers, $s_1, \ldots, s_l \in \mathbb{Z}_n^*$. $A$ sets

$$g_i := h^{r_i} s_i^v \quad \forall i \in \{1, \ldots, l\},$$

and feeds $(n, v, (g_1, \ldots, g_l))$ to $B$.

**Step 2.** $A$ receives $(x_1, \ldots, x_l, x_{l+1})$ from $B$, and checks whether or not it is a non-trivial RSA-representation of 1. If it is not, then $A$ halts.

**Step 3.** If $\sum_{i=1}^{l} r_i x_i = 0 \bmod v$, then $A$ halts.

**Step 4.** Using the extended Euclidean algorithm, $A$ computes integers $e, f \in \mathbb{Z}$ satisfying

$$e\,(\sum_{i=1}^{l} r_i x_i) + fv = 1.$$

$A$ then computes

$$h^f (x_{l+1} \prod_{i=1}^{l} s_i^{x_i})^{-e},$$

and outputs the result.

It is easy to verify that the output in Step 4 is equal to $h^{1/v}$, and that the total running time is $O(l|v|)$ plus the running time of $B$. We now determine the success probability of $A$.

Because the joint distribution of $(g_1, \ldots, g_l)$, generated in Step 1, is the same as that of the output of $I_{\text{RSAREP}}$, the transition from Step 2 to Step 3 occurs with probability $\epsilon$. To determine the probability that the transition from Step 3 to Step 4 takes place, we observe that there exists an integer $j \leq l$ such that $x_j \neq 0 \bmod v$ (because $B$'s output is non-trivial). Therefore, there are exactly $v^{l-1}$ "bad" tuples $(r_1, \ldots, r_l) \in (\mathbb{Z}_v)^l$, for which $\sum_{i=1}^{l} r_i x_i = 0 \bmod v$: for any choice of $(r_1, \ldots, r_{j-1}, r_{j+1}, \ldots, r_l)$, the remaining number, $r_j$, exists and is uniquely determined because $v$ is a prime. The tuple $(r_1, \ldots, r_l)$ is unconditionally hidden from $B$, owing to the randomness of the $s_i$'s and the fact that $v$ is co-prime to $\varphi(n)$, and it is therefore independent of $B$'s output. Any choice of $(r_1, \ldots, r_l)$ is equally likely to have been made by $A$, and so the probability of having chosen a bad tuple is $v^{l-1}/v^l = 1/v$.

Because Step 4 takes place only if Step 3 is successful, and Step 3 takes place only if Step 2 is successful, the overall success probability of $A$ is $\epsilon(1 - 1/v)$. □

The proof reduction is optimally tight, and so using the parameter sizes recommended for the RSA function to implement the RSAREP function results in roughly the same security level. In particular, taking a $2048$-bit $n$ and 200-bit $v$ should suffice for long-term security.

As in the case of the DLREP function, Construction 2.3.4 is not the only one for which Proposition 2.3.5 can be proved, but it suffices for our purposes in this book. From now on, $(I_{\text{RSAREP}}, D_{\text{RSAREP}})$ always denotes an invulnerable instance generator constructed from an invulnerable instance generator $(I_{\text{RSA}}, D_{\text{RSA}})$ for the RSA function in the manner of Construction 2.3.4. Of course, it is permitted to use any instance generator with a distribution that is indistinguishable from that generated by $(I_{\text{RSAREP}}, D_{\text{RSAREP}})$.

The constructed RSAREP function can be used by a sender to commit to $l$ attributes, $(x_1, \ldots, x_l)$. We will show in Chapter 3 how the sender can gradually and selectively open its commitment to an infinitely powerful receiver, and more generally can demonstrate all sorts of properties about its attributes without leaking additional information about them.

### 2.3.4 Comparison

It is clear that the DLREP function and the RSAREP function have much in common, and indeed most constructions in this book can be based on either function. There are some notable differences, though:

- In the RSAREP function the factorization of $n$ can serve as a trapdoor, enabling the computation of arbitrary RSA-representations for any number in $\mathbb{Z}_n^*$. The

DLREP function is not known to have a trapdoor.

- In the RSAREP function, $v$ can be arbitrarily small or be a fixed constant (hard-wired into the instance generator). It is easy to see that Proposition 2.3.5 remains valid for these choices (but the reduction is no longer optimally tight). Similar choices do not exist for the DLREP function, because the infeasibility of collision-finding is directly related to the binary size of $q$.

As a result, the RSAREP function offers greater flexibility. As we will see in the next section, though, a large $v$ is desirable to construct highly practical showing protocols, and so the second advantage of the RSAREP function is not of interest to us.

For our purposes in this book the DLREP function is usually preferable, for reasons related to practicality:

- The DLREP function can be evaluated faster than the RSAREP function.
- The DL-representation takes less storage space than the RSA-representation for the same security level, assuming that exponents are smaller than numbers in the base.
- Assuming the elliptic curve construction for $G_q$ resists subexponential-time inverting algorithms, storage of numbers in $G_q$ requires significantly less space than storage of numbers in $\mathbb{Z}_n^*$ for the same security level, and computations involving base numbers are much faster.

Moreover, as we will see in the next section, the real-time operations needed to prove knowledge of a DL-representation are much fewer than in the case of an RSA-representation, because all exponentiations can be precomputed.

## 2.4 Proofs of knowledge

### 2.4.1 Definition

With $(i, x)$ denoting the output of an instance generator for a function $\{f_i(\cdot)\}_{i \in V}$, and with $i$ understood, $f_i(x)$ may be called a *public key* and $x$ a *secret key* (or *witness*) corresponding to the public key. A *key pair* consists of a secret key and a public key. The outputs of algorithm $I$ form the *system parameters*, and the process of running $D$ and forming the public key is referred to as the *key set-up*.[13]

The public key uniquely corresponds to the secret key, but unless $f_i(\cdot)$ is a permutation there may be many secret keys corresponding to each public key. On input $i$ and the public key, it is infeasible to compute a corresponding secret key if and

[13]This definition in terms of collections of one-way functions is not standard, but makes sense in our situation as well as in most other cases that consider only polynomially bounded key holders.

only if the function is one-way over $(I, D)$. Moreover, if the function is collision-intractable over $(I, D)$, no party that is given $i$ can feasibly generate a public key for which it knows two corresponding secret keys.

We now come to the notion of a proof of knowledge, originating from Goldwasser, Micali, and Rackoff [193]. Informally, this is a protocol by means of which one party can convince another that it "knows" a secret key corresponding to its public key.

**Definition 2.4.1.** *A* proof of knowledge *($\mathcal{P}$, $\mathcal{V}$) for a function $\{f_i(\cdot)\}_{i \in V}$ is a protocol performed by a pair of interactive polynomial-time algorithms. $\mathcal{P}$ is called the* prover *and $\mathcal{V}$ is called the* verifier. *The protocol ($\mathcal{P}$, $\mathcal{V}$) must satisfy the following two properties:*

- *(Completeness) For all $k$, for all $(i, x) \in V \times D_i$,*

$$\mathsf{P}_k\left(\overline{\mathcal{V}}_{\overline{\mathcal{P}}(x)}(i, f_i(x)) \text{ accepts}\right) = 1.$$

  *The probability is taken over the coin flips (if any) of $\overline{\mathcal{V}}$ and $\overline{\mathcal{P}}$.*

- *(Soundness) There exists an expected polynomial-time algorithm $\mathcal{K}$, called a* knowledge extractor, *such that for all $\widehat{\mathcal{P}}$, for all constants $c > 0$, for all $(i, x) \in V \times D_i$ with $|i|$ sufficiently large, and for all auxiliary inputs* $\mathsf{aux}$,

$$\left| \mathsf{P}_k\left(\overline{\mathcal{V}}_{\widehat{\mathcal{P}}(\mathsf{aux})}(i, f_i(x)) \text{ accepts}\right) - \mathsf{P}_k\left(f_i(\mathcal{K}((i, f_i(x)), \mathsf{aux}; \widehat{\mathcal{P}})) = f_i(x)\right) \right|$$

  *is smaller than $1/k^c$.*

Loosely speaking, the two properties state that the prover can convince the verifier if and only if the prover knows a secret key corresponding to its public key. [14]

The simplest proof of knowledge is one in which $\mathcal{P}$ sends $x$ to $\mathcal{V}$, whereupon $\mathcal{V}$ checks its correspondence to the public key by applying $f_i(\cdot)$. For our purposes in this book, however, $\mathcal{P}$ should not reveal its secret key. protocol.

## 2.4.2 Security for the prover

Soundness is a formalization of security for $\overline{\mathcal{V}}$. Many flavors of security for $\overline{\mathcal{P}}$ have been studied in the literature. We now examine the four most useful ones.

[14] Definition 2.4.1 originates from Feige and Shamir [169]. Bellare and Goldreich [22] provided a more general definition of proof of knowledge that takes into account provers that have superpolynomial computing power. We do not consider this alternative definition here, since it is considerably more complex and the presented one is adequate for our purposes.

**Definition 2.4.2.** *A proof of knowledge ($\mathcal{P}$, $\mathcal{V}$) for $f(\cdot)$ is computationally* zero-knowledge *if there exists an expected polynomial-time algorithm $S$, called a* simulator, *such that for all $\widehat{\mathcal{V}}$ and for all auxiliary inputs* aux, *the two ensembles*

$$\left\{\widehat{\mathcal{V}}_{\overline{\mathcal{P}}(x)}((i, f_i(x)); \mathsf{aux})\right\}_{(i,x)\in V\times D_i} \quad \textit{and} \quad \left\{S((i, f_i(x)), \mathsf{aux}; \widehat{\mathcal{V}})\right\}_{(i,x)\in V\times D_i}$$

*are computationally indistinguishable.*

Equivalently, the views of $\widehat{\mathcal{V}}$ in protocol executions with $\overline{\mathcal{P}}$ can be simulated with indistinguishable probability distribution.

In a similar manner one can define statistical and perfect zero-knowledge; in these cases the simulator must be able to output protocol transcripts that are statistically indistinguishable from, or identically distributed to, the protocol transcripts that a verifier with unlimited computing power sees when interacting with $\overline{\mathcal{P}}$. Statistical and perfect zero-knowledge are meaningful notions in case $f_i(\cdot)$ is not a permutation; even though $\widetilde{\mathcal{V}}$ can compute all secret keys corresponding to $\mathcal{P}$'s public key, it cannot find out more about which one is known to $\mathcal{P}$ than what is known in advance.

It is possible to construct protocols that are zero-knowledge when protocol executions are performed sequentially, but that leak the secret key of the prover in case attackers are able to engage in parallel executions of the protocol; see Feige and Shamir [169] for an example.

The zero-knowledge property states that a misbehaving $\mathcal{V}$ (with either polynomial or unlimited computing power, depending on the flavor) cannot learn any information beyond what it can infer from merely the system parameters and $\mathcal{P}$'s public key. A weaker notion, which will be very useful in Chapter 3 to prove the unforgeability of digital signature schemes in the random oracle model, is the following.

**Definition 2.4.3.** *A proof of knowledge ($\mathcal{P}$, $\mathcal{V}$) for a function $f(\cdot)$ is (computationally, statistically, perfectly)* honest-verifier zero-knowledge *if there exists an expected polynomial-time simulator $S$ such that the two ensembles*

$$\left\{\overline{\mathcal{V}}_{\overline{\mathcal{P}}(x)}((i, f_i(x)))\right\}_{(i,x)\in V\times D_i} \quad \textit{and} \quad \left\{S((i, f_i(x)); \overline{\mathcal{V}})\right\}_{(i,x)\in V\times D_i}$$

*are (computationally, statistically, perfectly) indistinguishable.*

The following notion (due to Feige and Shamir [169]) is also weaker than zero-knowledge, but in many applications it is at least as useful.

**Definition 2.4.4.** *A proof of knowledge ($\mathcal{P}$, $\mathcal{V}$) for a function $f(\cdot)$ is statistically* witness-indistinguishable *if, for any $x_1, x_2 \in D_i$ such that $f_i(x_1) = f_i(x_2)$, and for any auxiliary input* aux *to $\widetilde{\mathcal{V}}$, the two ensembles defined by*

$$\widetilde{\mathcal{V}}_{\overline{\mathcal{P}}(x_1)}((i, f_i(x_1)); \mathsf{aux}) \quad \textit{and} \quad \widetilde{\mathcal{V}}_{\overline{\mathcal{P}}(x_2)}((i, f_i(x_2)); \mathsf{aux}),$$

*respectively, are statistically indistinguishable.*

In other words, $\widetilde{\mathcal{V}}$ cannot learn any information about which particular secret key is applied by $\overline{\mathcal{P}}$. In a similar manner one can define computational and perfect witness-indistinguishable proofs of knowledge. More generally, one can define witness-indistinguishability for protocols that are not proofs of knowledge.

A proof of knowledge for a permutation is trivially witness-indistinguishable; this is an uninteresting property because it holds even for protocols in which $\mathcal{P}$ transmits its secret key to $\mathcal{V}$. Later in this section, and more importantly in Chapter 3, we will introduce witness-indistinguishable proofs of knowledge for many-to-one functions.

The following proposition is due to Feige and Shamir [169].

**Proposition 2.4.5.** *Witness-indistinguishability is preserved under arbitrary composition of protocols.*

This property, which does not hold for zero-knowledge, applies not only to different executions of the same protocol but also to executions of different witness-indistinguishable protocols.

In contrast to the properties of completeness, soundness, (general and honest-verifier) zero-knowledge, and witness-indistinguishability, the last notion of security for $\overline{\mathcal{P}}$ discussed here is defined only over the output distribution of a specific instance generator.

**Definition 2.4.6.** *A proof of knowledge ($\mathcal{P}$, $\mathcal{V}$) for a function $f(\cdot)$ is* witness-hiding *over the instance generator $(I, D)$ for $f(\cdot)$ if there exists an expected polynomial-time algorithm $W$, called a* witness extractor, *such that for all verifiers $\widehat{\mathcal{V}}$, for all constants $c > 0$, for all sufficiently large $k$, and for all auxiliary inputs* $\mathsf{aux}$,

$$\Big| \mathsf{P}_k\Big(f_i(\widehat{\mathcal{V}}_{\overline{\mathcal{P}}(x)}(i, f_i(x); \mathsf{aux})) = f_i(x) \mid i := I(1^k); x := D(i)\Big) \\ - \mathsf{P}_k\Big(f_i(W((i, f_i(x), \mathsf{aux}); \widehat{\mathcal{V}})) = f_i(x)\Big)\Big| < 1/k^c.$$

The witness-hiding property states that $\widehat{\mathcal{V}}$, after having engaged in (at most) polynomially many protocol executions with $\overline{\mathcal{P}}$, cannot compute an entire secret key that corresponds to $\overline{\mathcal{P}}$'s public key, unless it already knew or could compute such a secret key before any protocol executions with $\mathcal{P}$ were performed. The latter case is not interesting, and so we will refer to proofs of knowledge over vulnerable instance generators as being *trivial witness-hiding*. Note that non-trivial witness-hiding does not exclude the possibility that $\widehat{\mathcal{V}}$ can uniquely determine half of the bits of $\mathcal{P}$'s secret key, say, once it has engaged in sufficiently many protocol executions. Due to the soundness property, however, non-trivial witness-hiding offers adequate security in most applications of proofs of knowledge.

The following proposition also originates from Feige and Shamir [169].

**Proposition 2.4.7.** *Let $(I, D)$ be an instance generator for a function $f(\cdot)$ such that the following two properties hold for each $y$ in the range of $f_i(\cdot)$:*

- *$y$ has at least two preimages in the domain $D_i$ of $f_i(\cdot)$; and*
- *conditional on the event that $D(i)$ outputs an element in the preimage set of $y$, none of the preimages of $y$ has overwhelming probability of being output by $D(i)$.*

*Then the following holds: if ($\mathcal{P}$, $\mathcal{V}$) is a computationally witness-indistinguishable proof of knowledge for $f(\cdot)$, and $f(\cdot)$ is collision-intractable over $(I, D)$, then ($\mathcal{P}$, $\mathcal{V}$) is non-trivially witness-hiding over $(I, D)$.*

To prove this result, suppose that $\widehat{\mathcal{V}}$ outputs a secret key corresponding to $\overline{\mathcal{P}}$'s public key, after having engaged in polynomially many protocol executions. Owing to the witness-indistinguishability property this key differs with non-negligible probability from the secret key used by $\overline{\mathcal{P}}$. Therefore, the algorithm $<\overline{\mathcal{P}}$, $\widehat{\mathcal{V}}>$ finds collisions for the function with non-negligible success probability.

Note that the key set-up and the process of generating the system parameters do not enter the definition of proofs of knowledge. Definition 2.4.1 simply assumes correct formation. When designing a system, care must be exercised as to which party controls algorithms $I$ and $D$:

- The process of generating the system parameters must be controlled by the party or parties to whom improperly formed system parameters pose a security threat. For example, if $\mathcal{V}$ runs $I$ then it may be able to embed trapdoor information so that it can feasibly compute a secret key corresponding to the public key of $\mathcal{P}$; whether this is a threat to the security of $\mathcal{P}$ depends on the application at hand. On the other hand, if $\mathcal{P}$ runs $I$, $\mathcal{P}$ may be able to determine system parameters for which it can find collisions. Again, whether or not this is a problem depends on the application at hand; for a situation in which $\mathcal{P}$ should not run $I$ by itself, see Chapter 3.

  Any interests of $\mathcal{V}$ and $\mathcal{P}$ can be met by letting a trusted party run $I$. Using cryptographic multi-party computation techniques (see, for instance, Chaum, Crepeau, and Damgård [89] and Chaum, Damgård, and van de Graaf [106]), it is possible for $\mathcal{V}$ and $\mathcal{P}$ to create a "virtual" trusted party to run $I$. Although multi-party computation techniques are not practical in general, in all the constructions in this book they can be implemented in a practical manner. For instance, with the RSA-based constructions that we will present there is no need for the prover to prove that $n$ is the product of two primes of equal size, and the proof that $v$ is co-prime to $\varphi(n)$ is a by-product of the certificate issuing protocol. More generally, correct formation by one of $\mathcal{V}$ and $\mathcal{P}$ in our constructions can always be proved by providing an additional output evidencing that the process has taken as input a source of randomness substantially outside of its control.

- Normally the key set-up is performed by $\mathcal{P}$, to make sure that its secret key does not become known to $\mathcal{V}$. In some applications, $\mathcal{V}$ and $\mathcal{P}$ should jointly perform this process, by means of an interactive protocol. For example, Chapter 4 addresses the situation where the CA ensures that a part of the secret key generated for a receiver contains pre-approved attributes, while the receiver ensures that the CA cannot learn its entire secret key.

We now introduce practical proofs of knowledge for both the DLREP function and the RSAREP function. These will be central to our constructions of issuing and showing protocols in the next two chapters.

## 2.4.3 Proving knowledge of a DL-representation

Consider any instance generator for the DLREP function. $\mathcal{P}$'s public key is $h := \prod_{i=1}^{l} g_i^{x_i}$. In order to prove knowledge of a DL-representation of $h$ with respect to $(g_1, \ldots, g_l)$, $\mathcal{P}$ and $\mathcal{V}$ perform the following protocol steps:

**Step 1.** $\mathcal{P}$ generates at random $l$ numbers $w_1, \ldots, w_l \in \mathbb{Z}_q$. It then sends $a := \prod_{i=1}^{l} g_i^{w_i}$ to $\mathcal{V}$. The number $a$ is called the *initial witness*.

**Step 2.** $\mathcal{P}$ computes $l$ *responses*, responsive to a *challenge* $c \in \mathbb{Z}_s$ of $\mathcal{V}$, where $1 < s \leq q$, according to $r_i := cx_i + w_i \bmod q$, for $i = 1, \ldots, l$, and sends them to $\mathcal{V}$. (The role of $s$ and the process of forming $c$ will be discussed shortly.)

$\mathcal{V}$ accepts if and only if the verification relation $\prod_{i=1}^{l} g_i^{r_i} h^{-c} = a$ holds.

Note that both $a$ and the left-hand side of the verification relation can be rapidly computed using simultaneous repeated squaring.

A variation is for $\mathcal{P}$ in Step 1 to send a one-way hash of $a$; $\mathcal{V}$ must then check whether this number is equal to the hash of $\prod_{i=1}^{l} g_i^{r_i} h^{-c}$. Also, if $\mathcal{V}$ knows $y_i := \log_g g_i$, for some generator $g$ and all $i \in \{1, \ldots, l\}$, then the verification relation can be collapsed to

$$g^{\sum_{i=1}^{l} y_i r_i} h^{-c} = a.$$

We will not consider these variations any further.

The integer $s$ in Step 2 must be known to both $\mathcal{P}$ and $\mathcal{V}$. It may be deterministically related to the system parameters (e.g., a predetermined rounded fraction of $q$, or $q$ itself). Alternatively, it may be specified as part of the process of generating the system parameters or $\mathcal{P}$ may specify it when informing $\mathcal{V}$ of its public key.

The challenge $c$ need not be generated at random, nor need it be generated by $\mathcal{V}$. Nevertheless, we will always refer to it as *$\mathcal{V}$'s challenge*, because it determines the security for $\mathcal{V}$.

The protocol description is *generic* in the sense that the binary size of $s$ and the process of generating $c$ have yet to be specified. Also, we have not yet stated any requirements for the instance generator.

**Proposition 2.4.8.** *($\mathcal{P}$, $\mathcal{V}$) is complete and perfectly witness-indistinguishable, regardless of the binary size of $s$ and the process of generating $\mathcal{V}$'s challenge.*

*Proof.* Completeness follows from

$$\begin{aligned}\prod_{i=1}^{l} g_i^{r_i} h^{-c} &= \prod_{i=1}^{l} g_i^{cx_i+w_i} h^{-c}\\ &= (\prod_{i=1}^{l} g_i^{x_i})^c (\prod_{i=1}^{l} g_i^{w_i}) h^{-c}\\ &= h^c a h^{-c}\\ &= a.\end{aligned}$$

To prove witness-indistinguishability, we will show that any view of $\widetilde{\mathcal{V}}$ could have resulted from any secret key of $\overline{\mathcal{P}}$, with equal probability. Suppose $\overline{\mathcal{P}}$ used secret key $(x_1^*, \ldots, x_l^*)$. In Step 1 it would have sent

$$a^* := \prod_{i=1}^{l} g_i^{w_i^*}$$

to $\mathcal{V}$, and in Step 3 it would have sent responses $r_i^* := cx_i^* + w_i^* \bmod q$, for $i \in \{1, \ldots, l\}$. From $r_i = r_i^* \bmod q$ it follows that $w_i^* = r_i - cx_i^* \bmod q$, for $i = 1, \ldots, l$, and since $\mathcal{P}$'s responses make $\widetilde{\mathcal{V}}$ accept it follows that

$$\begin{aligned}a^* &= \prod_{i=1}^{l} g_i^{w_i^*}\\ &= \prod_{i=1}^{l} g_i^{r_i - cx_i^*}\\ &= \prod_{i=1}^{l} g_i^{r_i} (\prod_{i=1}^{l} g_i^{x_i^*})^{-c}\\ &= (h^c a)\, h^{-c}\\ &= a.\end{aligned}$$

Since the $w_i$'s are chosen at random from $\mathbb{Z}_q$, the view perfectly hides which secret key has been used, and the claimed result follows. □

Soundness and security for $\overline{\mathcal{P}}$ depend on the binary size of $s$ and the process of generating $c$. Furthermore, for the property of witness-hiding we need to specify an instance generator. Assuming that $c$ is chosen at random by $\mathcal{V}$, and becomes known to $\mathcal{P}$ only after $\mathcal{P}$ has chosen its initial witness $a$, the following security implications hold:

**(Large $s$)** If $s$ is superpolynomial in $k$, then the protocol is a proof of knowledge as is; no repetitions are needed to achieve soundness. It is easy to prove that it is honest-verifier zero-knowledge: the simulator generates $r_1, \ldots, r_l$ and $c$ at random, and computes $a := \prod_{i=1}^{l} g_i^{r_i} h^{-c}$. The following two cases describe conditions under which $(\mathcal{P}, \mathcal{V})$ is witness-hiding over $(I_{\text{DLREP}}, D_{\text{DLREP}})$:

- In case $\mathcal{V}$ in advance knows $(x_1, \ldots, x_{l-1})$ with overwhelming probability, but has no a priori information about $x_l$, the protocol is believed to be witness-hiding over $(I_{\text{DLREP}}, D_{\text{DLREP}})$. It is easy to prove that the case of arbitrary $l$ is as secure as the special case $l = 1$, which is the Schnorr proof of knowledge [337]. In his generic string encoding model, Shoup [352] proved that an active attacker in the Schnorr proof of knowledge cannot learn enough information to be able to subsequently prove knowledge of $\mathcal{P}$'s secret key by itself. In other words, the Schnorr proof of knowledge is witness-hiding in the generic string encoding model, a result that can easily be adapted to the case of arbitrary $l$.
- In case $l \geq 2$ and $\mathcal{V}$ cannot identify $(x_1, \ldots, x_{l-1})$ in advance with overwhelming probability (i.e., $\mathcal{V}$ has *non-negligible uncertainty* about the tuple), it follows from Propositions 2.3.3, 2.4.7, and 2.4.8 that the protocol is provably (non-trivially) witness-hiding over $(I_{\text{DLREP}}, D_{\text{DLREP}})$.

In either case, the protocol can be made zero-knowledge by prepending a fourth move in which $\mathcal{V}$ commits to its challenge; the required strength of the commitment depends on whether or not $\mathcal{V}$ is polynomially bounded. [15] Alternatively, $\mathcal{P}$ and $\mathcal{V}$ determine $\mathcal{V}$'s challenge in a mutually random fashion.

**(Small $s$)** If $s$ is polynomial in $k$, then the protocol steps must be repeated polynomially many times in order to result in a sound protocol. ($\mathcal{V}$ accepts if and only if it accepts in each iteration.) Two cases can be discerned:

- Sequential repetitions result in a zero-knowledge proof of knowledge.
- Parallel repetitions do not result in a zero-knowledge proof of knowledge. They are believed, however, to result in a proof of knowledge that is witness-hiding over $(I_{\text{DLREP}}, D_{\text{DLREP}})$. (This can be proved in case $l \geq 2$ and $\mathcal{V}$ initially has non-negligible uncertainty about $(x_1, \ldots, x_{l-1})$.) A zero-knowledge protocol can be obtained by prepending a fourth move in the manner described for the case of large $s$.

We will not consider the case of small $s$ any further in this book, because the resulting protocols are significantly less efficient. In later chapters, we will often take $s := q$.

[15] As noted by Bellare (personal communication, January 8, 1999), the commitment may not be of the form $g_i^{\alpha} h^c$, for some $i \in \{1, \ldots, l\}$ and random $\alpha \in \mathbb{Z}_q$, since this would allow an attacker to always convince $\mathcal{V}$ without knowing a DL-representation of $h$. A commitment of the form $g_i^c h^{\alpha}$ should be fine, although it is unclear how to prove the soundness property.

The most practical way to obtain a protocol that is provably witness-hiding is to set $l = 2$, to generate $x_2$ at random from $\mathbb{Z}_q$, and to set $x_1$ equal to the outcome of a coin flip (not necessarily unbiased); the resulting three-move protocol (with large $s$) is an optimization of Okamoto's extension [288, page 36] of the Schnorr proof of knowledge. Taking $l > 2$ does not improve the provability of the witness-hiding property and only makes the protocol less efficient; for this reason the situation $l > 2$ has never been considered in the literature. In Chapter 3, however, we will see that there are legitimate reasons for resorting to $l > 2$.

The security results for $\overline{\mathcal{P}}$ hold only assuming that the system parameters are formed by running $I_{\text{DLREP}}$. Depending on the application at hand, both $\mathcal{P}$ and $\mathcal{V}$ may have security interests in seeing to it that the system parameters are formed in this manner. For example, if $\mathcal{P}$ is allowed to generate at least one $g_i$ by itself, after $\mathcal{V}$ has formed the remaining ones, $\mathcal{P}$ can easily construct colliding secret keys. For our purposes in Chapter 3 it suffices that $\mathcal{V}$, or a party trusted by $\mathcal{V}$, runs $I_{\text{DLREP}}$. Additional outputs may be sent to $\mathcal{P}$ to prove that random or pseudorandom bits have been used in the process, and a proof of primality of $q$ may be included.

Furthermore, depending on the application, it may be necessary for $\mathcal{P}$ and $\mathcal{V}$ to check membership in $G_q$ of certain numbers:

- If $\mathcal{P}$ can get way with a public key $h$ that is not a member of $G_q$, then a corresponding secret key does not exist, yet $\widehat{\mathcal{P}}$ may be able to make $\mathcal{V}$ accept with non-negligible probability. Burmester [67] pointed out for the Schnorr proof of knowledge that $\widehat{\mathcal{P}}$ can convince $\mathcal{V}$ with probability $1/2$ by multiplying in a non-trivial square root of unity; the same attack applies in the general case. This issue, which applies not only to the subgroup construction but also to the elliptic curve construction, will also play a role in Chapter 3.

  To circumvent the problem, $\mathcal{V}$ should check that $\mathcal{P}$'s public key $h$ is a member of $G_q$. This one-time check can take place off-line, before the protocol takes place, and is especially practical in applications in which the same public key is used in many protocol executions. If $G_q$ is a subgroup of a commutative group of order $o$, and $q$ divides $o$ but $q^2$ does not divide $o$, then the check $h^q = 1$ suffices to verify membership in $G_q$; see Herstein [210, Corollary on page 62]. It can also be shown that the check $h^q = 1$ suffices to prove membership in case $G_q$ is not a subgroup of a cyclic group, provided the verifier accepts the protocol execution; cf. Verheul and Hoyle [382].

  In the digital certificate constructions in Chapters 4, 5, and 6, the problem does not play a role.

- Lim and Lee [252] showed that, in the subgroup construction for $G_q$, it may in general be dangerous for the prover to apply its secret key to base numbers supplied by the verifier without first checking that these are indeed members of $G_q$. Two ways around this are the following:

– The prover can check membership in $G_q$ of each supplied base number $a$ to which it is to apply its secret exponent.
– One can use a prime $p$ such that $(p-1)/2q$ contains only prime factors greater than $q$. This circumvents the need to perform real-time membership verifications.

In this book the issue is avoided altogether, because in our protocol constructions the prover never applies its secret key to base numbers supplied by the other party.

### 2.4.4 Proving knowledge of an RSA-representation

Consider any instance generator for the RSA function. $\mathcal{P}$'s public key is $h := \prod_{i=1}^{l} g_i^{x_i} x_{l+1}^v$. In order to prove knowledge of an RSA-representation of $h$ with respect to $(g_1, \ldots, g_l, v)$, $\mathcal{P}$ and $\mathcal{V}$ perform the following protocol steps:

**Step 1.** $\mathcal{P}$ generates at random $l$ numbers $w_1, \ldots, w_l \in \mathbb{Z}_v$ and a random number $w_{l+1} \in \mathbb{Z}_n^*$, and sends the initial witness $a := \prod_{i=1}^{l} g_i^{w_i} w_{l+1}^v$ to $\mathcal{V}$.

**Step 2.** $\mathcal{P}$ computes $l+1$ responses, responsive to a challenge $c \in \mathbb{Z}_s$, where $1 < s \leq v$, as follows:

$$\begin{aligned} r_i &:= cx_i + w_i \bmod v \quad \forall i \in \{1, \ldots, l\}, \\ r_{l+1} &:= \prod_{i=1}^{l} g_i^{(cx_i+w_i) \operatorname{div} v} x_{l+1}^c w_{l+1} \end{aligned}$$

$\mathcal{P}$ then sends $r_1, \ldots, r_l, r_{l+1}$ to $\mathcal{V}$. (The role of $s$ and the process of forming $c$ will be discussed shortly.)

$\mathcal{V}$ accepts if and only if the verification relation

$$\prod_{i=1}^{l} g_i^{r_i} r_{l+1}^v h^{-c} = a$$

holds.

Note that $a$, $r_{l+1}$ and the left-hand side of the verification relation can all be computed almost completely using simultaneous repeated squaring with a single precomputed table, since $(g_1, \ldots, g_l, h)$ are all fixed; only the $v$-th powers occurring in these expressions have to be multiplied separately into the products.

**Proposition 2.4.9.** *($\mathcal{P}$, $\mathcal{V}$) is complete and perfectly witness-indistinguishable, regardless of the binary size of $s$ and the process of generating $\mathcal{V}$'s challenge.*

Soundness and security for $\overline{\mathcal{P}}$ depend on the binary size of $s$ and the process of generating $c$. Furthermore, for the property of witness-hiding we need to specify an instance generator. In case $c$ is chosen at random by $\mathcal{V}$, and becomes known to $\mathcal{P}$ only after it has chosen its initial witness $a$, we have similar security implications as described for the case of the DLREP function. We mention only the following two cases, both for $s$ superpolynomial in $k$:

- In case $\mathcal{V}$ knows $(x_1, \ldots, x_l)$ with overwhelming probability, but does not know $x_{l+1}$, the protocol is believed to be witness-hiding over $(I_{\text{RSAREP}}, D_{\text{RSAREP}})$, even though this has yet to be proved. The special case $l = 0$ is the Guillou-Quisquater proof of knowledge [201, 202]. (Recall that $v$ is a prime that is superpolynomial in $k$.)

- In case $l \geq 1$ and $\mathcal{V}$ initially has non-negligible uncertainty about $(x_1, \ldots, x_l)$, it follows from Propositions 2.3.5, 2.4.7, and 2.4.9 that the protocol is provably (non-trivially) witness-hiding over $(I_{\text{RSAREP}}, D_{\text{RSAREP}})$.

There is no point in using $s > v$: if $\widehat{\mathcal{P}}$ can respond to $c$ then it can also respond to $c + jv$, for any integer $j$. In later chapters, we will frequently take $s := v$.

The most practical way to obtain a protocol that is provably witness-hiding is to set $l = 1$, to generate $x_2$ at random from $\mathbb{Z}_n^*$, and to set $x_1$ equal to the outcome of a coin flip (not necessarily unbiased); the resulting three-move protocol (with large $s$) is an optimization of Okamoto's extension [288, page 39] of the Guillou-Quisquater proof of knowledge. As in the case of the DLREP function, we will show in Chapter 3 that there are legitimate reasons for resorting to $l > 1$.

A four-move zero-knowledge proof of knowledge can be obtained by prepending a move in which $\mathcal{V}$ commits to $c$. One way to form the commitment is by encoding $c$ into the hard-core bits of the commitment function of Håstad, Schrift, and Shamir [208]; another is to use the RSAREP function.

As in the case of the proof of knowledge for the DLREP function, the above security results hold only assuming that the system parameters are indeed formed by running $I_{\text{RSAREP}}$. Note that $v$ need not be a prime or be co-prime to $\varphi(n)$ to make the protocol secure, assuming one is willing to restrict the set from which the $g_i$'s and $x_{l+1}$ and $w_{l+1}$ are chosen.[16] In light of the goal that will be pursued in Chapter 3, though, we will only consider the choices for $v$ and $n$ made here.

[16]For example, the protocol is a witness-hiding proof of knowledge if $n$ is a Blum integer, $v = 2$, the $g_i$'s, $x_{l+1}$, and $w_{l+1}$ are all quadratic residues, and the protocol moves are repeated polynomially many times. One can also consider a modification similar to that of Feige, Fiat, and Shamir [168], in which $n$ is a Blum integer and $\mathcal{P}$ randomly multiplies $\pm 1$ into $r_{l+1}$ and $a$. Yet another choice is $v = 2^t$, for $t$ such that $v$ is superpolynomial in $k$; although the prover can convince the verifier with success probability $1/2^j$ if it knows an RSA-representation of the $2^j$-th power of $h$ for some $j$, the protocol can be proved secure against an active impersonator, relative to the factoring assumption (cf. Shoup [351] and Schnorr [338]).

## 2.5 Digital signatures

### 2.5.1 Definition

Informally, a digital signature is the electronic analogue of a handwritten signature. A digital signature on a message can be verified by anyone without the help of the signer, by applying the public key of the signer, but only the signer can compute signatures on valid messages, by applying its secret key. The following definition formalizes this.

**Definition 2.5.1.** *A* digital signature scheme *consists of a function* $\{f_i(\cdot)\}_{i \in V}$*, an invulnerable instance generator* $(I, D)$*, two* message sets $\mathcal{M} = \{\mathcal{M}_i\}_{i \in V}$ *and* $\mathcal{M}^* = \{\mathcal{M}_i^*\}_{i \in V}$*, a Boolean predicate* $\mathsf{pred}(\cdot)$ *that can be evaluated in polynomial time, and a protocol ($\mathcal{P}$, $\mathcal{V}$) performed by a pair of interactive polynomial-time algorithms. $\mathcal{P}$ is called the* signer*, $\mathcal{V}$ the* receiver*, and ($\mathcal{P}$, $\mathcal{V}$) the (signature)* issuing protocol*. ($\mathcal{P}$, $\mathcal{V}$) must satisfy the following two properties:*

- *(Signature generation) For all* $(i, x) \in V \times D_i$ *and for all* $m^* \in \mathcal{M}_i^*$*, if*

$$(m, \sigma) := \overline{\mathcal{V}}_{\overline{\mathcal{P}}(x)}((i, f_i(x)), m^*)$$

  *then* $m \in \mathcal{M}_i$ *and* $m$ *is a superstring*[17] *of* $m^*$*, and the probability function defined by*

$$\mathsf{P}_k\Big(\mathsf{pred}(i, f_i(x), m, \sigma) = 1\Big)$$

  *is overwhelming in* $k$*.*

  *The pair* $(m, \sigma)$ *is called a* signed message*, and* $\sigma$ *is $\mathcal{P}$'s* digital signature *on* $m$*.*

- *(Unforgeability) For any* $t \geq 0$*, the following holds. The probability (taken over* $(I, D)$ *and the coin flips of* $\widehat{\mathcal{V}}$ *and* $\overline{\mathcal{P}}$*) that* $\widehat{\mathcal{V}}$*, after having engaged in up to* $t$ *protocol executions with* $\overline{\mathcal{P}}$*, outputs at least* $t + 1$ *distinct signed messages (with messages in* $\mathcal{M}$*) is negligible in* $k$*.*

  *The digital signature scheme is said to be* unforgeable *over* $(I, D)$*.*

Whenever the instance generator is clear from the description of the signature scheme, we will simply say that the signature scheme is unforgeable.

Note that the capability to obtain two different signatures on the same message by engaging in a single execution of the protocol with the signer is not considered to

[17]That is, the binary string $m^*$ can be obtained by pruning bits from the binary string $m$. For instance, any $m$ is a superstring of the null string. Further examples are described shortly. Definition 2.5.1 could be generalized by considering messages $m$ that have other relations to $m^*$ (e.g., $m$ is a preimage under a one-way function of $m^*$), but this is outside of the scope of the book.

fall under the scope of forgery. This convention is arbitrary, and adopted here merely for concreteness.

Definition 2.5.1 is based on the standard definition of Goldwasser, Micali, and Rivest [194], but differs in a few respects. The standard definition is not suitable to describe blind signature schemes and other schemes with interactive issuing protocols, including those that we will design in Chapter 4. Also, it includes the processes of generating the system parameters and the key set-up, but not a notion of security for the signer; this is opposite to the way the definition of proofs of knowledge is structured, and does not reflect the common basis of both notions. The definition given here is adequate for our purposes.[18]

The role of the auxiliary common input $m^*$ in Definition 2.5.1 differs depending on the type of signature scheme:

- In the most widely considered digital signatures in the cryptographic literature, the signature issuing protocol is non-interactive, $\mathcal{M}$ equals $\mathcal{M}^*$, and $m^*$ is equal to the message $m$. In this case, the unforgeability property implies that the receiver cannot obtain a signature on a message that the signer has not seen and knowingly signed.
- In the case of blind signatures (see Chaum [91, 92, 93, 94, 95, 96, 99, 100]), $m^*$ is always the empty string and $m$ is generated at random by $\mathcal{V}$. Specifically, a blind digital signature scheme is a digital signature scheme with the additional property that the signed message $(m, \sigma)$ obtained by $\overline{\mathcal{V}}$ by interacting with $\widetilde{\mathcal{P}}$ is statistically independent of $\widetilde{\mathcal{P}}$'s view in the protocol execution. (Weaker flavors are possible. The weakest flavor is that in which $\widehat{\mathcal{P}}$ cannot correlate signed messages to its views of protocol executions.)
- In Section 4.2 we will construct issuing protocols in which the message $m$ is chopped up into polynomially many message blocks, $x_1, \ldots, x_l, \alpha$. In this case, $m^*$ equals the concatenation of $x_1, \ldots, x_l$; the remaining message block, $\alpha$, is generated secretly at random by $\mathcal{V}$.

As with the witness-hiding and zero-knowledge properties for proofs of knowledge, the unforgeability property for digital signatures depends on a number of factors that have been described in Section 2.1.4. Specifically, the resistance of a digital signature scheme to forgery is influenced by the maximum number of protocol executions in which $\widehat{\mathcal{V}}$ can engage and by the degree to which the protocol executions can be interleaved. In addition, unforgeability depends on how, when, and by which party the message $m^*$ is formed.

[18] A variation of Definition 2.5.1 is to move the unforgeability property outside of the definition (just like witness-hiding is not a part of the definition of a proof of knowledge), and instead to complement the "completeness" property (signature generation) by a "weak soundness" property that states that $(\mathcal{P}, \overline{\mathcal{V}})$ results in a signed message with probability 1 only if $\mathcal{P}$ knows a secret key corresponding to its public key $f_i(x)$. This introduces the problem of defining what it means for a non-interactive algorithm to "know" information.

An attack can proceed in several manners. In a successful *key-only attack* or *forgery from scratch*, $\widehat{\mathcal{V}}$ is able to forge signed messages without being given the opportunity to interact with $\overline{\mathcal{P}}$. At the other end of the spectrum is the *adaptively chosen message attack*, in which $\widehat{\mathcal{V}}$ has the freedom to choose all its contributions (e.g., messages, blinding factors, challenges) to each execution of the issuing protocol with $\overline{\mathcal{P}}$ in a manner that may depend on its aggregate view in all the protocol executions up to that point; $\widehat{\mathcal{V}}$ can use $\overline{\mathcal{P}}$ as an oracle, and is limited only by the level of interleaving of protocol executions that $\mathcal{P}$ allows.

A successful forgery may be due to a leakage of $\mathcal{P}$'s secret key $x$; in this case $\widehat{\mathcal{V}}$ is able to forge $\mathcal{P}$'s signature on any message. Such a *total break* can be prevented by using a signature issuing protocol that is non-trivially witness-hiding. $\widehat{\mathcal{V}}$ need not necessarily know $\mathcal{P}$'s secret key, however, to be able to forge a signed message. For instance, once $\widehat{\mathcal{V}}$ has obtained a number of signed messages it may be able to algebraically combine these in such a manner that an additional signed message results. In general, we will need to protect against *existential forgery*; in this case $\widehat{\mathcal{V}}$ is able to forge one signed message, for a message not necessarily of its own choice or under its control. The unforgeability property of digital signatures states that existential forgery is infeasible, even under an adaptively chosen message attack.

### 2.5.2 From proofs of knowledge to digital signature schemes

We now describe a general construction for converting a proof of knowledge into a digital signature scheme. Consider hereto a proof of knowledge for a one-way function $f(\cdot)$ that has the following structure:

**Step 1.** $\mathcal{P}$ generates a randomly distributed initial witness $a$. It sends $a$, which may in general be a vector of numbers, to $\mathcal{V}$.

**Step 2.** $\mathcal{V}$ generates a substantially random challenge, $c \in \mathbb{Z}_s$, with $s$ superpolynomial in the security parameter $k$. It sends the challenge, which may represent a concatenation of several challenge numbers, to $\mathcal{P}$.

**Step 3.** $\mathcal{P}$ computes a response, $r$, as the outcome of a function of its secret key, the challenge and the coin flips used to construct $a$. It sends the response, which may in general be a vector of numbers, to $\mathcal{V}$.

$\mathcal{V}$ applies a Boolean polynomial-time computable predicate $\mathsf{pred}(\cdot)$ to the system parameters, $\mathcal{P}$'s public key, $\mathcal{P}$'s initial witness, its own challenge, and $\mathcal{P}$'s response, and accepts if and only if the outcome of the predicate is 1.

We refer to proofs of knowledge of this structure as *Fiat-Shamir type* proofs of knowledge, because the following technique for converting them into digital signature schemes was first proposed (for a particular instance) by Fiat and Shamir [171]. Note that the proofs of knowledge described in Section 2.4 are of this type.

The conversion into a digital signature scheme is brought about by replacing the role of $\mathcal{V}$ by a "virtual" verifier. This is accomplished by computing $\mathcal{V}$'s challenge according to $c := \mathcal{H}_i(m, a)$, where $m \in \mathcal{M}_i$ is any message and $\mathcal{H}(\cdot)$ is a sufficiently strong (see page 84 for details) one-way hash function that must be specified together with the system parameters or the public key of $\mathcal{P}$. The signature on $m$ is defined to be $\sigma := (a, r)$, and anyone can verify it by computing $c$ and applying $\mathsf{pred}(\cdot)$. Because the digital signature on $m$ is obtained by means of an issuing protocol that is derived from a proof of knowledge, it is also called a *signed proof*.

As a general rule, from a security perspective it is recommended to hash along all the information that $\mathcal{V}$ needs to check anyway to verify the signed proof, including the public key, algorithm identifiers, and purpose specifiers. (They may all be assumed to be part of the message, $m$.)

Note that the properties of witness-indistinguishable and witness-hiding are preserved under the conversion.

In case $m$ is known to $\mathcal{P}$ at the start of the protocol, for instance because $\mathcal{M}$ is the empty set or $\mathcal{V}$ provides $m$ before learning $a$, $\mathcal{P}$ can compute $c := \mathcal{H}_i(m, a)$ by itself, and the issuing protocol can be non-interactive; $\mathcal{P}$ simply sends $(a, r)$ to $\mathcal{V}$. It is conjectured that the non-interactive signature scheme is unforgeable if the signature issuing protocol is witness-hiding. Modeling $\mathcal{H}(\cdot)$ as a random oracle, Pointcheval and Stern [307] proved the following unforgeability result.

**Proposition 2.5.2.** *Suppose that the binary sizes of the outputs of $f_i(\cdot)$ and $\mathcal{H}_i(\cdot)$ are linear*[19] *in $k$ and let $(I, D)$ be an invulnerable instance generator for $f(\cdot)$. If a Fiat-Shamir type proof of knowledge for $f(\cdot)$ is honest-verifier zero-knowledge, then its conversion to a non-interactive signature scheme is unforgeable over $(I, D)$ in the random oracle model.*

This result holds even under an adaptively chosen message attack whereby the signer engages in polynomially many protocol executions that are arbitrarily interleaved.

In case $\mathcal{V}$ wants to hide (at least) the message $m$ from $\mathcal{P}$, it must supply $c$ itself. In this case the protocol remains interactive and the conditions of Proposition 2.5.2 are insufficient to prove unforgeability in the random oracle model. By generalizing a result due to Pointcheval [305], which is an optimization of a result of Pointcheval and Stern [306], it is possible to prove the following result.

**Proposition 2.5.3.** *Suppose that the binary sizes of the outputs of $f_i(\cdot)$ and $\mathcal{H}_i(\cdot)$ are linear in $k$. Let $(I, D)$ be such that the condition in Proposition 2.4.7 holds, and suppose that $f(\cdot)$ is collision-intractable over $(I, D)$. If a Fiat-Shamir type proof of knowledge for $f(\cdot)$ is computationally witness-indistinguishable and the prover performs no more than polylogarithmically*[20] *many protocol executions, then its con-*

[19] In fact, we merely need that $2^{-|\mathcal{H}_i(\cdot)|}$ is negligible in $k$.

[20] A function $f(\cdot)$ is polylogarithmic in $k$ if there exists a positive integer $c$ such that $f(k) \leq (\log k)^c$ for all sufficiently large $k$.

*version to an interactive signature scheme is unforgeable over $(I, D)$ in the random oracle model.*

This result holds under an adaptively chosen message attack, and the protocol executions may be arbitrarily interleaved.

If it were only for the ability of $\mathcal{V}$ to hide $m$ from $\mathcal{P}$, the interactive variant of the signature scheme would hardly be interesting. Okamoto and Ohta [289] showed that it is possible for $\mathcal{V}$, in interactive signature schemes derived from witness-hiding Fiat-Shamir type proofs of knowledge, to perfectly blind the issuing protocol, provided that certain properties hold. These properties, collectively referred to as *commutative random self-reducibility*, apply to virtually all practical Fiat-Shamir type proofs of knowledge proposed to date, including those described in Section 2.4. In case the condition in Proposition 2.5.3 holds, we obtain blind signature schemes that are provably secure in the random oracle model. However, as explained in Section 1.2.2, Chaum's blinding techniques are unsuitable for our purposes.

We now show how to construct practical digital signature schemes from our proofs of knowledge for the DLREP function and the RSAREP function.

### 2.5.3 Digital signatures based on the DLREP function

In Section 2.4.3 we investigated two variations for proving knowledge of a DL-representation: one in which $s$ is small and the protocol steps are repeated in parallel polynomially many times, and the other in which no repetitions are needed because $s$ is superpolynomial in $k$. Both protocols are readily seen to be Fiat-Shamir type proofs of knowledge, and can be converted into a digital signature scheme by applying the described technique. As mentioned in Section 2.4.3 we will not consider the case of small $s$, for reason of practicality. Consider now the case of large $s$. Let $\mathcal{H}(\cdot)$ be a one-way hash function, defined by

$$\mathcal{H}_{q,g_l}(\cdot) : \mathcal{M}_{q,g_l} \times G_q \mapsto \mathbb{Z}_s.$$

A description of $\mathcal{H}_{q,g_l}(\cdot)$ must be specified together with the system parameters or $\mathcal{P}$'s public key. The definition of $\mathcal{H}_i(\cdot)$ and $\mathcal{M}_i$ may also depend on $(g_1, \ldots, g_{l-1})$, $\mathcal{P}$'s public key, and any other information that is specified before protocol executions take place; for notational reasons we do not make this explicit in the notation. In practice the outputs of $\mathcal{H}_{q,g_l}(\cdot)$ will usually be $t$-bit strings, for some $t$ with $2^t$ (possibly much) smaller than $s$.

$\mathcal{P}$'s digital signature on a message $m$ is a vector $(a, r_1, \ldots, r_l)$ such that the relation

$$\prod_{i=1}^{l} g_i^{r_i} h^{-\mathcal{H}_{q,g_l}(m,a)} = a$$

holds. Alternatively, one can define the signature to be $(c, r_1, \ldots, r_l)$, and the signature verification relation is

$$c = \mathcal{H}_{q,g_l}(m, \prod_{i=1}^{l} g_i^{r_i} h^{-c}).$$

Since a signature of either one type is readily computed from one of the other type, the security for $\overline{\mathcal{P}}$ is not affected. Differences exist in terms of efficiency, though:

- Using $(c, r_1, \ldots, r_l)$ is favorable in case the subgroup construction is used to construct $G_q$, because the storage complexity of $c$ is smaller than that of $a$. In particular, using the parameter sizes recommended in Section 2.2.2, many hundreds of bits are saved.

- Using $(a, r_1, \ldots, r_l)$ may be preferable when $t > 1$ digital signatures need to be verified. With $(a_i, r_{1i}, \ldots, r_{li})$ denoting $\mathcal{P}$'s digital signature on message $m_i$, for all $i \in \{1, \ldots, t\}$, $\mathcal{V}$ sets $\alpha_1 := 1$ and generates $\alpha_2, \ldots, \alpha_t$ at random from a set $V \subseteq \mathbb{Z}_q$. $\mathcal{V}$ then computes

  $$c_i := \mathcal{H}_{q,g_l}(m_i, a_i) \quad \forall i \in \{1, \ldots, t\},$$

  and verifies the compound verification relation

  $$\prod_{i=1}^{l} g_i^{\sum_{j=1}^{t} \alpha_j r_{ij}} h^{-\sum_{i=1}^{t} \alpha_i c_i} = \prod_{i=1}^{t} a_i^{\alpha_i}.$$

  It is easy to prove that if the compound verification relation holds, then the probability that all $t$ signatures are valid is at least $1 - 1/|V|$. Since the left-hand side of the compound verification relation can be rapidly computed using simultaneous repeated squaring with a single precomputed table, this *batch-verification technique* is a substantial improvement over verifying all $t$ digital signatures separately.[21]

The security of the digital signature scheme depends on which party specifies $c$:

**(Non-interactive issuing protocol)** Proposition 2.5.2 can be applied, since the proof of knowledge is honest-verifier zero-knowledge.

**Proposition 2.5.4.** *If $(I_{\text{DL}}, D_{\text{DL}})$ is invulnerable, and the binary size of outputs of $\mathcal{H}_i(\cdot)$ is linear in $k$, then non-interactively issued signed proofs are provably unforgeable over $(I_{\text{DLREP}}, D_{\text{DLREP}})$ in the random oracle model, for any distribution of $(x_1, \ldots, x_{l-1})$.*

[21] For large $t$ it is more efficient to randomly partition the $t$ verification relations into a suitable number of "buckets," and apply batch-verification to each bucket. Cf. Bellare, Garay, and Rabin [21].

We stress that this result holds even in case $(x_1, \ldots, x_{l-1})$ is known in advance to $\mathcal{V}$ with overwhelming probability, if only $x_l$ is a random secret.

The special case $l = 1$ is the Schnorr signature scheme [337]. Note that $\mathcal{P}$ in the non-interactive issuing protocol need not transmit $a$ to $\mathcal{V}$; it can be recovered from $c$ and $\mathcal{P}$'s responses.

**(Interactive issuing protocol)** By applying Proposition 2.5.3 we obtain the following result.

**Proposition 2.5.5.** *Let $l \geq 2$ and suppose that $\mathcal{V}$ initially has non-negligible uncertainty about $(x_1, \ldots, x_{l-1})$. If the DL function used to implement the DLREP function is one-way, the binary size of outputs of $\mathcal{H}_i(\cdot)$ is linear in $k$, and $\mathcal{P}$ does not perform more than polylogarithmically many executions of the issuing protocol, then interactively issued signed proofs are provably unforgeable over $(I_{\text{DLREP}}, D_{\text{DLREP}})$ in the random oracle model, for any distribution of $(x_1, \ldots, x_{l-1})$.*

On the basis of this result[22] We make the following assumption, which will be needed in Chapter 4.

**Assumption 2.5.6.** *There exists a hash function $\mathcal{H}^*(\cdot)$ and a message set $\mathcal{M} = \{\mathcal{M}_i\}_{i \in \{q, g_l\}}$ with $\mathcal{M}_{q,g_l} \supseteq G_q$ such that interactively issued signed proofs are unforgeable over $(I_{\text{DLREP}}, D_{\text{DLREP}})$.*

It is easy to prove that this assumption holds for all $l \geq 1$ if it holds for $l = 1$, for the same choice of hash function.

In accordance with the blinding technique of Okamoto and Ohta [289], $\mathcal{V}$ can blind the issuing protocol by performing the following action after Step 1 of the proof of knowledge. It generates $l+1$ random numbers, $\alpha_0, \ldots, \alpha_l \in \mathbb{Z}_q$, and computes $a' := a h^{\alpha_0} \prod_{i=1}^{l} g_i^{\alpha_i}$, $c' := \mathcal{H}_{q,g_l}(m, a')$, and $c := c' + \alpha_0 \bmod q$. It then sends its challenge $c$ to $\mathcal{P}$. Upon receiving $\mathcal{P}$'s responses, $r_1, \ldots, r_l$, $\mathcal{V}$ computes $r'_i := r_i + \alpha_i \bmod q$, for all $i \in \{1, \ldots, l\}$. It is easy to verify that $(a', r'_1, \ldots, r'_l)$, or $(c', r'_1, \ldots, r'_l)$ for that matter, is $\overline{\mathcal{P}}$'s digital signature on $m$. Moreover, if $\mathcal{V}$ chooses $m$ at random and accepts then the signed message is statistically independent of $\widetilde{\mathcal{P}}$'s view in the protocol execution. However, there are no practical advantages in using this scheme over Chaum's RSA-based blind signature scheme [91, 92] (with small $v$), unless one trusts an elliptic curve implementation with short system parameters. Furthermore, as

[22] In their "random oracle + generic" security model, Schnorr and Jakobsson [339] prove a sharp security bound for the unforgeability of interactively issued Schnorr signatures, assuming sequential protocol executions. They also showed that parallel attacks that beat the success rate of sequential attacks must solve the problem of finding an intersection point of a subset of randomized hyperplanes. Proposition 2.5.5 can be proved for all $l \geq 1$ under the same assumption in a similar manner.

explained in Section 1.2.2, Chaum's blinding paradigm is unsuitable for our purposes. In Chapter 3 we will introduce more intricate blinding techniques that result in all sorts of previously unattainable results.

In practice, one-wayness of $\mathcal{H}(\cdot)$ is not enough for unforgeability over $(I_{\text{DLREP}}, D_{\text{DLREP}})$. Existential forgery of non-interactively issued Schnorr signatures, for instance, is feasible in case two messages $m_1, m_2$ exist such that, for random $(a_1, a_2) \in G_q \times G_q$, it is feasible to compute with non-negligible success probability a third message $m$ and two integers $\alpha, \beta$ such that the following correlation holds:

$$\alpha \mathcal{H}_{q,g_l}(m_1, a_1) + \beta \mathcal{H}_{q,g_l}(m_2, a_2) = \mathcal{H}_{q,g_l}(m, a_1^{\alpha} a_2^{\beta}) \bmod q.$$

In the interactive case, $\mathcal{H}(\cdot)$ must be even stronger. To guarantee unforgeability in practice, $\mathcal{H}(\cdot)$ must be *correlation-intractable*, meaning that it is infeasible to compute correlations like the one displayed. (Note that functions with superpolynomial range are always correlation-intractable when idealized in the random oracle model.) Because it is unclear how to formalize the notion of correlation-intractability in a useful way, we will from now on always speak of a *sufficiently strong* one-way function whenever we need a correlation-intractable function. In practice, hash functions such as SHA-I or RIPEMD-160 should suffice.[23]

Schnorr [337] suggests for his signature scheme that a one-way hash function with $10$-byte outputs should suffice for long-term practical security. There seems to be no reason not to allow this choice for arbitrary $l$ in the non-interactive case. In the interactive case this should hold as well, assuming that $\mathcal{P}$ does not respond to $\mathcal{V}$'s challenge in case the delay between sending the initial witness and receiving the challenge exceeds a short time bound. To be on the safe side, it is strongly recommended to always use a sufficiently strong collision-intractable hash function with at least $20$-byte outputs. This choice is also preferred in light of the more intricate signed proofs that will be described in Chapter 3.

## 2.5.4 Digital signatures based on the RSAREP function

Consider the case of $s$ superpolynomial in $k$ in the proof of knowledge in Section 2.4.4, for a prime $v$ superpolynomial in $k$ and co-prime to $\varphi(n)$. Let $\mathcal{H}(\cdot)$ be a one-way hash function, defined by

$$\mathcal{H}_{n,v}(\cdot) : \mathcal{M}_{n,v} \times \mathbb{Z}_n^* \mapsto \mathbb{Z}_s.$$

The definition of $\mathcal{H}_i(\cdot)$ and $\mathcal{M}_i$ may also depend on $(g_1, \ldots, g_l)$, $\mathcal{P}$'s public key, and any other information that is specified before protocol executions take place.

[23]Note that these are not infinite collections of functions: they act on messages of any size, but their outputs are of fixed size.

$\mathcal{P}$'s digital signature on a message $m$ is a vector $(a, r_1, \ldots, r_{l+1})$ such that the verification relation

$$\prod_{i=1}^{l} g_i^{r_i} r_{l+1}^{v} h^{-\mathcal{H}_{n,v}(m,a)} = a$$

holds. This form lends itself to batch-verification. Alternatively, and more compactly, one can define the signature to be $(c, r_1, \ldots, r_{l+1})$, and the corresponding signature verification relation is

$$c = \mathcal{H}_{n,v}(m, \prod_{i=1}^{l} g_i^{r_i} r_{l+1}^{v} h^{-c}).$$

Again, this does not affect the security for $\overline{\mathcal{P}}$.

As with the DLREP function, we distinguish two cases:

**(Non-interactive issuing protocol)** Proposition 2.5.2 can be applied, since the proof of knowledge is honest-verifier zero-knowledge.

**Proposition 2.5.7.** *If $(I_{\text{RSA}}, D_{\text{RSA}})$ is invulnerable, and the binary size of outputs of $\mathcal{H}_i(\cdot)$ is linear in $k$, then non-interactively issued signed proofs are provably unforgeable over $(I_{\text{RSAREP}}, D_{\text{RSAREP}})$ in the random oracle model, for any distribution of $(x_1, \ldots, x_l)$.*

The special case $l = 1$ is the Guillou-Quisquater signature scheme [201, 202].[24]

**(Interactive issuing protocol)** Proposition 2.5.3 can be invoked to prove the following result.

**Proposition 2.5.8.** *Let $l \geq 1$ and suppose that $\mathcal{V}$ initially has non-negligible uncertainty about $(x_1, \ldots, x_l)$. If the RSA function used to implement the RSAREP function is one-way, the binary size of outputs of $\mathcal{H}_i(\cdot)$ is linear in $k$, and $\mathcal{P}$ does not perform more than polylogarithmically many executions of the issuing protocol, then interactively issued signed proofs are provably unforgeable over $(I_{\text{RSAREP}}, D_{\text{RSAREP}})$ in the random oracle model, for any distribution of $(x_1, \ldots, x_l)$.*

Based on this result, we make the following assumption.

**Assumption 2.5.9.** *There exists a hash function $\mathcal{H}^*(\cdot)$ and a message set $\mathcal{M} = \{\mathcal{M}_i\}_{i \in \{n,v\}}$ with $\mathcal{M}_{n,v} \supseteq \mathbb{Z}_n^*$ such that interactively issued signed proofs are unforgeable over $(I_{\text{RSAREP}}, D_{\text{RSAREP}})$.*

[24]Four years earlier, Shamir [346] described essentially the same signature scheme, with the irrelevant difference that $\mathcal{V}$'s hashed challenge appears as the exponent of $a$ instead of $h$. This scheme also predates the paper of Fiat and Shamir [171] to which the conversion technique in Section 2.5.2 is generally attributed.

It is easy to prove that the assumption holds for all $l \geq 0$ if it holds for $l = 0$, for the same choice of hash function.

$\mathcal{V}$ can blind the issuing protocol, in accordance with the technique of Okamoto and Ohta [289], by performing the following action after Step 1 of the proof of knowledge. It generates $l + 1$ random numbers, $\alpha_0, \ldots, \alpha_l \in \mathbb{Z}_v$ and a random number $\alpha_{l+1} \in \mathbb{Z}_n^*$, and computes $a' := ah^{\alpha_0} \prod_{i=1}^{l} g_i^{\alpha_i} \alpha_{l+1}^v$, $c' := \mathcal{H}_{n,v}(m, a')$, and $c := c' + \alpha_0 \bmod v$. It then sends its challenge, $c$, to $\mathcal{P}$. Upon receiving $\mathcal{P}$'s responses, $r_1, \ldots, r_{l+1}$, $\mathcal{V}$ computes $r_i' := r_i + \alpha_i \bmod v$, for all $i \in \{1, \ldots, l\}$, and $r_{l+1}' := r_{l+1} \prod_{i=1}^{l} g_i^{(r_i + \alpha_i) \operatorname{div} v} \alpha_{l+1}$. It is easy to verify that $(a', r_1', \ldots, r_{l+1}')$, or $(c', r_1', \ldots, r_{l+1}')$ for that matter, is $\overline{\mathcal{P}}$'s digital signature on $m$. Moreover, if $\overline{\mathcal{V}}$ generates $m$ at random and accepts then the signed message is statistically independent of $\widetilde{\mathcal{P}}$'s view in the protocol execution. However, there is no practical advantage over Chaum's blind signature scheme [91, 92], which is much more efficient because small $v$ may be taken. Furthermore, as explained in Section 1.2.2, Chaum's blinding paradigm is unsuitable for our purposes. In Chapter 3 we will introduce more intricate blinding techniques that offer all sorts of benefits.

In practice, $\mathcal{H}(\cdot)$ must be a sufficiently strong one-way hash function. Similar considerations as in the case of the DLREP function apply to the binary size of the outputs of $\mathcal{H}(\cdot)$. Although in the interactive issuing protocol 10-byte outputs should be sufficiently secure when used in combination with a time-out, it is recommended to always use a sufficiently strong collision-intractable hash function with at least 20-byte outputs. In particular, this will be necessary in Chapter 3.

## 2.6 Digital certificates

### 2.6.1 Definition of public-key certificates

Finally, we get to a formal definition of digital certificates. We start with the traditional definition, which is a special case of the definition of digital signatures.

**Definition 2.6.1.** *A* public-key certificate scheme *is a digital signature scheme with the extra property that the message $m$ specifies at least a public key $p$ of $\overline{\mathcal{V}}$, for which $\overline{\mathcal{V}}$ knows a corresponding secret key, $s$.*

*The pair $(p, \sigma)$ is called a* certified public key, *$\sigma$ is $\mathcal{P}$'s* digital certificate *on $m$, and the triple $(s, p, \sigma)$ is called a* certified key pair. *$\mathcal{P}$ is also referred to as the Certificate Authority (*CA*).*

$\overline{\mathcal{V}}$'s key pair may be generated either before or during the protocol execution.

In the case of conventional identity certificates (see Section 1.1.2), the message $m$ is the concatenation of $p$ and at least a key holder identifier. In the case of attribute certificates other attributes are concatenated, either along with the identifier or instead of the identifier.

As mentioned already in Section 1.3.1, our use of the term "digital certificate" differs from the mainstream use of the term, which considers a certificate to be the data structure comprised of the CA's signature, the public key it certifies, and any information assigned to that public key. Our convention makes it easier to distinguish between various cryptographic objects.

The definition of the key pair $(s, p)$ for $\mathcal{V}$ may be the same as that of the key pair $(x, f_i(x))$ for $\mathcal{P}$, but need not; it may even be completely unrelated. Likewise, completely different instance generators may be used.

### 2.6.2 Definition of secret-key certificates

We now introduce a new kind of certificates that differ from public-key certificates in that anyone can generate certified public keys without the assistance of $\mathcal{P}$, but certified key pairs remain unforgeable. The formal definition is as follows.

**Definition 2.6.2.** *A* secret-key certificate scheme *is a digital signature scheme with the additional property that there exists another Boolean predicate,* $\mathsf{pred}^*(\cdot)$, *that can also be evaluated in polynomial time, such that:*

- *The message $m$, on which $\overline{\mathcal{V}}$ obtains a signature $\sigma$ in an execution of the issuing protocol with $\overline{\mathcal{P}}$, is a secret key of $\overline{\mathcal{V}}$. This secret key corresponds to a public key $p$ of $\mathcal{V}$ such that*
$$\mathsf{pred}^*(i, f_i(x), p, \sigma) = 1.$$
- *There exists an expected polynomial-time algorithm $S$, called a* certificate simulator, *that, on input $(i, f_i(x))$, outputs a pair $(p^*, \sigma^*)$ with a probability distribution that is indistinguishable from the probability distribution of $(p, \sigma)$.*

*As in the definition of a public-key certificate, $\sigma$ is called $\mathcal{P}$'s* digital certificate *on $\mathcal{V}$'s public key, the pair $(p, \sigma)$ is called a certified public key, and the triple $(m, p, \sigma)$ is called a certified key pair.*

The output distribution of the certificate simulator may be computationally, statistically, or perfectly indistinguishable.

**Example 2.6.3.** *$\mathcal{P}$ runs $I_{\text{RSA}}$ to obtain a pair $(n, v)$ together with the factorization of $n$, which serves as its secret key. This enables $\mathcal{P}$ to compute RSA digital signatures [325]. Its RSA signature on a message $m$ is $\sigma := f_{n,v}(m)^{1/v} \bmod n$, where $f(\cdot)$ is a sufficiently strong one-way function.*

*This signature scheme can be converted into a secret-key certificate scheme by viewing the pair $(m, f_{n,v}(m))$ as a key pair for $\mathcal{V}$. If we take $f(\cdot)$ to be the DL function implemented using the subgroup construction (i.e., $f_{n,v}(m) = g^m \bmod p$,* *for some prime $p$ that is not much smaller than $n$), then it is easy to build a certificate simulator with indistinguishable outputs. The simulator picks a random $b \in \mathbb{Z}_n^*$ and checks whether $b^v \bmod n$ is an element of $G_q$; it repeats this experiment until successful, and then outputs the pair $(b^v \bmod n, b)$.*

How can a secret-key certificate be verified? In applications of practical interest, $\mathcal{V}$ will use its public key $p$ in a subsequent showing protocol without disclosing the secret key $m$, and so certified public keys cannot be verified by applying $\mathsf{pred}(\cdot)$. Also, applying $\mathsf{pred}^*(\cdot)$ to $(p, \sigma)$ does not prove that the certified public key has been issued by $\mathcal{P}$, in view of the simulation property. Instead, verification of $\mathcal{V}$'s certified public key takes place indirectly. Namely, the ability of $\mathcal{V}$ to perform a "cryptographic action" with respect to its public key attests to the fact that $\mathcal{V}$ knows a secret key corresponding to its public key, and this in turn convinces that the certificate has been issued by $\mathcal{P}$. Since there is no point in using public-key certificates without performing some cryptographic action that attests to the possession of a corresponding secret key, secret-key certificates offer the same basic functionality as do public-key certificates.[25] The following two cryptographic actions in the showing protocol will be of particular interest to us:

1. $\mathcal{V}$ performs a zero-knowledge proof of knowledge of a secret key corresponding to its public key. If $\mathcal{V}$ can successfully perform the proof, then the verifier is convinced not only that $\mathcal{V}$ knows a secret key corresponding to the public key, by virtue of the soundness property, but also that the certificate has been issued by $\mathcal{P}$. However, the transcript of the protocol execution does not convince anyone else of either one of these facts; the entire protocol execution is zero-knowledge. (This is an advantage over public-key certificates, for which the showing protocol is zero-knowledge in its entirety only when $\mathcal{V}$ proves possession of a certificate by means of a zero-knowledge proof as well.)

   In Example 2.6.3, $\mathcal{V}$ can prove knowledge of the secret key corresponding to its public key $g^m$ by using the Schnorr proof of knowledge or its $4$-move zero-knowledge variant.

2. $\mathcal{V}$ digitally signs a message. Given the message, the digital signature of $\mathcal{V}$, and the certified public key of $\mathcal{V}$, anyone is able to verify not only that the digital signature is genuine, but also that it was indeed made with respect to a public key certified by $\mathcal{P}$.

   In Example 2.6.3, $\mathcal{V}$ can sign a message using the Schnorr signature scheme; in the random oracle model, these signatures can be issued only by a party that knows the secret key.

[25]This idea is reminiscent of Shamir's [346] *self-certified* public keys, the goal of which is to avoid the need for an explicit certificate. Hereto the CA forms the public key of each applicant as a redundant message that encodes the applicant's identity, and issues a corresponding secret key to the applicant; key pairs should be unforgeable.

Note that a third cryptographic action can be performed: decrypting a message that has been encrypted with $\mathcal{V}$'s public key. In Example 2.6.3, $\mathcal{V}$ can decrypt messages that have been encrypted under its public key $g^m$ by means of, for instance, the ElGamal encryption scheme [146] in $G_q$. Since decryption requires knowledge of the secret key, the party that encrypted the message is ensured that either $\mathcal{V}$'s public key has been certified by $\mathcal{P}$ or $\widehat{\mathcal{V}}$ cannot decrypt. In the remainder of this book we will not be interested in the case of encryption; public keys for (hybrid) session encryption can always be formed at random at the start of an authenticated session.

### 2.6.3 Comparison

Secret-key certificates have several advantages over public-key certificates:

- They appear to be much better suited to design certificate issuing protocols with the following property: $\mathcal{V}$ is able to blind the certified public key but not a non-trivial part of its secret key. See Chapter 4 for details.
- A certified public key does not serve as signed evidence that its holder has been issued a certificate by the CA. This property is preserved if certified key pairs are used only to perform zero-knowledge proofs.
- The ability to simulate certified public keys enables individuals to hide in which of several PKIs they are participating. See Section 5.2 for details.
- Knowledge of certified public keys (obtained from repositories or otherwise) cannot help in attacking the certificate scheme of the CA.

On the other hand, care has to be taken when combining an interactive secret-key certificate issuing protocol with a showing protocol. Suppose a certificate simulation algorithm exists that outputs certified public keys for which the certificate simulation algorithm and $\mathcal{P}$ together know a corresponding secret key. Then an attacker may be able in the showing protocol to perform a cryptographic action with respect to a simulated public key by *delegating* part of the action to $\overline{\mathcal{P}}$; see Section 5.1.2 for an example.

Successful delegation to executions of protocols other than the certificate issuing protocol can simply be prevented by having $\mathcal{P}$ use independently generated keys for different tasks. Key separation is recommended practice anyway; see, e.g., Kelsey, Schneier, and Wagner [229]. To assess for a given application whether a cryptographic action can be delegated to an execution of the issuing protocol, it must be investigated whether certified public keys can be simulated in such a way that the cryptographic action in the showing protocol can be performed by using $\overline{\mathcal{P}}$ as an oracle. In the certificate schemes that will be developed in this book, delegation is never a problem; see Section 5.1.2 for details.

## 2.7 Bibliographic notes

The historical background of the DLREP function in Section 2.3.2 is quite diverse. In 1987, Chaum, Evertse, and van de Graaf [108] considered the case where all $g_i$'s are random elements from $\mathbb{Z}_p^*$, and claimed that inverting is infeasible when elements from the domain are generated at random (hardly a useful instance generator). Chaum and Crepeau [60], Chaum and van Antwerpen [110], Chaum [101], Boyar, Kurtz, and Krentel [44], Pedersen [298], van Heijst and Pedersen [380], and Okamoto [288] all designed schemes based on the special case $l = 2$. Chaum, van Heijst, and Pfitzmann [111, 112] studied collision-intractability for the special case of fixed $l$, with $g_i$'s in a group of prime order; their reduction is not tight, though, and takes exponential time for $l$ polynomial in $k$. In 1993, Brands [46] examined the one-wayness and the collision-intractability of the DLREP function for arbitrary $l$ polynomial in $k$, and provided a tight reduction to prove its collision-intractability; the overhead factor is approximately 2. Bellare, Goldreich, and Goldwasser [23] modified the reduction to simplify its analysis, but the overhead is slightly larger. Pfitzmann [301], elaborating on the reduction of Chaum, van Heijst, and Pfitzmann [111, 112], gave the first (fairly intricate) optimally tight reduction. The proof of Proposition 2.3.3 is similar to a simpler reduction by Schoenmakers [342], but differs in the assumption on the distribution of the $g_i$'s (for reasons that will become clear in Chapter 3). Construction 2.3.2 is new.

The RSAREP function in Section 2.3.3 was introduced by Brands [54]. Previously only the case $l = 1$ appeared, in proofs of knowledge (see Okamoto [288]) and implicitly in some proofs of security (see, e.g., Guillou and Quisquater [201]). Construction 2.3.4 and Proposition 2.3.5 appear here for the first time.

The notion of secret-key certificates is due to Brands [56]; see also Brands [52, 53]. Definition 2.6.2 has not appeared previously. The possibility of delegation was first noted by Schoenmakers (personal communication, May 1995), in the context of the electronic coin system of Brands [49].