Lecture Notes
Cryptographic Protocols

Version 1.32
February 6, 2018

Berry Schoenmakers

Department of Mathematics and Computer Science,
Technical University of Eindhoven,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands.
l.a.m.schoenmakers@tue.nl
berry@win.tue.nl

Cryptographic Protocols (2DMI00)
www.win.tue.nl/~berry/2DMI00/

Spring 2018

Available online: updated lecture notes and matching lecture slides (printable handout).

Preface

As a motivating example for the cryptographic protocols covered in these lecture notes consider the Dutch tradition of "Sinterklaaslootjes trekken," internationally known as "Secret Santa," in which a group of people anonymously exchange small gifts—often accompanied by poems quite a few rhyming couplets long. A lot of websites are available to help people organize such drawings over the internet, see, e.g., sinterklaaslootjes.net, lootjestrekken.nl, and the "Secret Santa" service at elfster.com. The interesting question is how to do this securely! That is, without trusting the website or program providing this service, but with the guarantee (a) that indeed a *random* drawing is performed, corresponding to a random permutation without fixed points, and (b) such that each participant learns *nothing* except who he or she is a Secret Santa for.

More serious applications of such privacy-protecting cryptographic protocols are emerging in lots of places. For instance, over the last two decades many electronic elections using advanced cryptography have already been conducted. Other applications involve the use of anonymous cash, anonymous credentials, group signatures, secure auctions, etc., all the way to secure (multiparty) computation.

To this end we study cryptographic techniques that go beyond what we like to refer to as Crypto 1.0. Basically, Crypto 1.0 concerns encryption and authentication of data during communication, storage, and retrieval. Well-known Crypto 1.0 primitives are symmetric (secret-key) primitives such as stream ciphers, block ciphers, and message authentication codes; asymmetric (public-key) primitives such as public-key encryption, digital signatures, and key-exchange protocols; and, keyless primitives such as cryptographic hash functions. The common goal is to protect against malicious outsiders, say, attacking storage or communication media.

On the other hand, Crypto 2.0 additionally aims at protection against malicious insiders, that is, against attacks by the other parties engaged in the protocol that one is running. Thus, Crypto 2.0 concerns computing with encrypted data, partial information release of data, and hiding the identity of data owners or any link with them. Well-known Crypto 2.0 primitives are homomorphic encryption, secret sharing, oblivious transfer, blind signatures, zero-knowledge proofs, and secure two/multi-party computation, which will all be treated to a certain extent in these lecture notes.

The treatment throughout will be introductory yet precise at various stages. Familiarity with basic cryptography is assumed. We focus on asymmetric techniques for cryptographic protocols, also considering proofs of security for various constructions. The topic of zero-knowledge proofs plays a central role. In particular, Σ-protocols are treated in detail as a primary example of the so-called simulation paradigm, which forms the basis of much of modern cryptography.

The first and major version of these lecture notes was written in the period of December 2003 through March 2004. Many thanks to all the students and readers over the years for their feedback, both directly and indirectly, which helped to finally produce the first full version of this text.

Berry Schoenmakers

Contents

Preface iii

List of Figures vi

1 Introduction 1

1.1 Terminology 1

1.2 Preliminaries 2

Number Theory – Group Theory – Probability Theory – Complexity Theory

1.3 Assumptions 6

Discrete Log and Diffie-Hellman Assumptions – Indistinguishability – Random Self-Reducibility – Random Oracle Model

1.4 Bibliographic Notes 14

2 Key Exchange Protocols 16

2.1 Diffie-Hellman Key Exchange 16

Basic Protocol – Passive Attacks – A Practical Variant – Aside: ElGamal Encryption

2.2 Authenticated Key Exchange 20

Man-in-the-Middle Attacks – A Protocol Using Digital Signatures – A Protocol Using Password-Based Encryption

2.3 Bibliographic Notes 22

3 Commitment Schemes 23

3.1 Definition 23

3.2 Examples 24

Using a Cryptographic Hash Function – Using a Discrete Log Setting – Impossibility Result

3.3 Homomorphic Commitments 26

3.4 Bibliographic Notes 26

4 Identification Protocols 27

4.1 Definitions 27

4.2 Password-based Schemes 28

4.3 One-Way Hash Chains 29

4.4 Basic Challenge-Response Protocols 29

Using Symmetric Encryption – Using Symmetric Authentication – Using Asymmetric Encryption – Using Asymmetric Authentication

4.5 Zero-knowledge Identification Protocols 31

Schnorr Zero-knowledge Protocol – Schnorr Protocol – Guillou-Quisquater Protocol

4.6 Witness Hiding Identification Protocols 37

Okamoto Protocol

4.7 Bibliographic Notes 39

5 Zero-Knowledge Proofs **40**
5.1 Σ-Protocols 40
5.2 Composition of Σ-Protocols 44
Parallel Composition – AND-Composition – EQ-Composition – OR-Composition – NEQ-Composition
5.3 Miscellaneous Constructions 51
5.4 Non-interactive Σ-Proofs 54
Digital Signatures from Σ-Protocols – Proofs of Validity – Group Signatures
5.5 Bibliographic Notes 59

6 Threshold Cryptography **60**
6.1 Secret Sharing 60
Shamir Threshold Scheme
6.2 Verifiable Secret Sharing 62
Feldman VSS – Pedersen VSS
6.3 Threshold Cryptosystems 66
Threshold ElGamal Cryptosystem
6.4 Bibliographic Notes 68

7 Secure Multiparty Computation **69**
7.1 Electronic Voting 69
7.2 Based on Threshold Homomorphic Cryptosystems 72
7.3 Based on Oblivious Transfer 73
7.4 Bibliographic Notes 75

8 Blind Signatures **76**
8.1 Definition 76
8.2 Chaum Blind Signature Scheme 77
8.3 Blind Signatures from Σ-Protocols 78
8.4 Bibliographic Notes 79

Bibliography **80**

List of Figures

2.1 Three-pass encryption? 21

4.1 Four basic challenge-response schemes 30
4.2 Schnorr's zero-knowledge protocol 32
4.3 Schnorr's identification protocol 35
4.4 Guillou-Quisquater's identification protocol 36
4.5 Okamoto's identification protocol 38

5.1 Σ-protocol for relation R 41
5.2 Transformed Σ-protocol for relation R 42
5.3 Insecure variant of Schnorr's protocol 43
5.4 Parallel composition of Schnorr's protocol 44
5.5 AND-composition of Schnorr's protocol 45
5.6 Alternative to AND-composition of Schnorr's protocol? 46
5.7 EQ-composition of Schnorr's protocol 47
5.8 OR-composition of Schnorr's protocol 48
5.9 NEQ-composition of Schnorr's protocol 50
5.10 Σ-protocol for $\{(A, B; x, y, z) : A = g^x h^y \wedge B = g^{xy} h^{(1-x)z} \wedge x \in \{0, 1\}\}$ 52
5.11 Alternative to Okamoto's protocol? 54
5.12 Parametrized insecure variant of Schnorr's protocol 56

7.1 Matching without embarrassments 72
7.2 $\binom{2}{1}$-OT protocol 74

8.1 Chaum's blind signature protocol 78
8.2 Schnorr-based blind signature protocol 78

CHAPTER 1

Introduction

1.1 TERMINOLOGY

The field of cryptology is generally divided into the two mutually dependent fields of cryptography and cryptanalysis. Cryptography concerns the design of (mathematical) schemes related to information security which resist cryptanalysis, whereas cryptanalysis is the study of (mathematical) techniques for attacking cryptographic schemes.

The following distinction is commonly made between cryptographic algorithms, cryptographic protocols, and cryptographic schemes.

DEFINITION 1.1 *A **cryptographic algorithm** is a well-defined* transformation, *which on a given input value produces an output value, achieving certain security objectives. A **cryptographic protocol** is a distributed algorithm describing precisely the* interactions between two or more entities, *achieving certain security objectives. A **cryptographic scheme** is a suite of* related *cryptographic algorithms and cryptographic protocols, achieving certain security objectives.*

Entities interact in a cryptographic protocol by exchanging messages between each other over specific communication channels. A basic distinction can be made between *point-to-point channels* linking two entities, and *broadcast channels* connecting a sender to multiple receivers. Communication channels are often assumed to provide particular security guarantees. For example, a *private channel* (or, *secure channel*) is a point-to-point channel employing some form of encryption to protect against eavesdropping and, possibly, employing some form of authentication to protect against tampering with messages. A channel without any protection against eavesdropping and tampering is sometimes called a *public channel* (or, *insecure channel*). Another example is a *bulletin board* which is a public authenticated broadcast channel, allowing a sender to broadcast authenticated messages. A **communication model** describes what type of channels are available between which pairs or sets of entities. The communication model hides the implementation details of the channels, which may be far from trivial and which may incur substantial cost.

The following distinction is commonly made between passive and active attacks. Both types of attack are defined in terms of an adversary. The **adversary** represents the coalition formed by an attacker and/or one or more of the entities taking part in the cryptographic scheme. The entities under control of the adversary are said to be *corrupt* and the remaining entities are said to be *honest*. The attacker itself may be thought of as an *outsider*, while the corrupt entities can be viewed as *insiders*. It is essential that the attacker and the corrupt

entities may coordinate their efforts.

DEFINITION 1.2 *In a **passive attack**, the adversary does not interfere with the execution of the algorithms and protocols comprising a cryptographic scheme. A passive adversary merely eavesdrops on the communication between the entities, and records all information it has access to, including all private information of the corrupt entities.*[1] *In an **active attack**, the adversary may—in addition—interfere with the communication within a cryptographic scheme by deleting, injecting, or modifying messages; moreover, the adversary may have the corrupt entities deviate from their prescribed behavior in an arbitrary way.*[2]

Hence, a passive attack on an encryption scheme corresponds to the classical case of an eavesdropper. The classical man-in-the-middle attack on a key exchange protocol is an example of an active attack.

1.2 PRELIMINARIES

Throughout these lecture notes, basic familiarity with the following notions from mathematics and theoretical computer science is assumed: number theory, group theory, probability theory, and complexity theory. Particularly relevant aspects of these theories are highlighted below.

1.2.1 NUMBER THEORY

Throughout, n is a positive integer. We use $\mathbb{Z}_n$ to denote the set of integers modulo n (or, more formally, the set of residue classes modulo n), and we write $\mathbb{Z}_n^* = \{x \in \mathbb{Z}_n : \gcd(x, n) = 1\}$ for the set of integers that have a multiplicative inverse modulo n. Further, we write $\phi(n) = |\mathbb{Z}_n^*|$ for Euler's phi function, for which we have $\phi(n) = n \prod_{p|n}(1 - 1/p)$ as a basic fact, implying that $\phi(n)$ can be computed efficiently given the prime divisors of n. As another useful fact, we mention that $n/\phi(n) = O(\log \log n)$.

EXERCISE 1.3 Prove the elementary bound $n/\phi(n) = O(\log n)$, using that the number of distinct prime factors $\omega(n)$ of n is $O(\log n)$, and that the ith smallest prime factor of n is at least $i + 1$ for $i = 1, 2, \ldots, \omega(n)$.

1.2.2 GROUP THEORY

Throughout, G_n denotes a finite cyclic group of order n, written multiplicatively. Usually, but not necessarily, we assume the group order n to be prime. Recall that any group of prime order is cyclic, and that any cyclic group is abelian (commutative). For $h \in G_n$, we let $\mathrm{ord}(h)$ denote its order, hence $\mathrm{ord}(h) \mid n$. Any $g \in G_n$ with $\mathrm{ord}(g) = n$ is a generator of G_n, that is, $G_n = \langle g \rangle = \{g^x : x \in \mathbb{Z}_n\}$, or equivalently, the elements of G_n can be enumerated as $1, g, g^2, g^3, \ldots, g^{n-1}$, and $g^n = 1$.

The **discrete logarithm** (or, **discrete log**) of an element $h \in \langle g \rangle$ is defined as the unique integer $x \in \mathbb{Z}_n$ satisfying $h = g^x$. We write $x = \log_g h$. For the following groups it is commonly assumed that computing discrete logarithms is hard for appropriate values of n.

EXAMPLE 1.4 *Take $G_n = \mathbb{Z}_p^*$, for a prime p. Then G_n is a cyclic group of order $n = p - 1$. Clearly, n is not prime (unless $p = 3$, of course).*

[1] Passively corrupt entities are also known as *semi-honest* (or, *honest-but-curious*) entities.
[2] Actively corrupt entities are also known as *malicious* (or, *cheating*) entities.

More generally, one may take $G_n = \mathbb{F}_q^*$, the multiplicative group of a finite field of order $q = p^r$, for some positive integer r. Then G_n is a cyclic group of order $n = q - 1$.

EXAMPLE 1.5 *Take $G_n = \langle g \rangle$, where g denotes an element of order p' in $\mathbb{Z}_p^*$, with $p' \mid p-1$, p, p' prime. Then G_n is a cyclic group of order $n = p'$. Clearly, n is prime in this case.*

More generally, one may take $G_n = \langle g \rangle$, where g denotes an element of order p' in $\mathbb{F}_q^*$, with $p' \mid q-1$.

EXAMPLE 1.6 *Consider $E(\mathbb{F}_q)$, the finite group of points of an elliptic curve over $\mathbb{F}_q$, which is usually written additively, with the point at infinity $\mathcal{O}$ as identity element. Then $E(\mathbb{F}_q)$ is abelian, but not necessarily cyclic. In general, $E(\mathbb{F}_q)$ is isomorphic to $\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$, where $n_1 \mid n_2$ and $n_1 \mid q-1$. So, $E(\mathbb{F}_q)$ is cyclic if and only if $n_1 = 1$. Hence, one may take $G_n = E(\mathbb{F}_q)$ if $n_1 = 1$; otherwise, one may take a cyclic subgroup of $E(\mathbb{F}_q)$ for G_n.*

Finally, as a convenient notation, we let $\langle g \rangle^* = \{g^x : x \in \mathbb{Z}_n^*\}$ denote the **set of all generators** of $\langle g \rangle$. Clearly, $|\langle g \rangle^*| = \phi(n)$, which is the number of generators of any cyclic group of order n.

EXERCISE 1.7 Show that for any $h \in \langle g \rangle$, the following conditions are equivalent:

(i) $h \in \langle g \rangle^*$,

(ii) $\mathrm{ord}(h) = n$,

(iii) $\langle h \rangle = \langle g \rangle$,

(iv) $\langle h \rangle^* = \langle g \rangle^*$.

EXERCISE 1.8 Show that $a^{\log_h b} = b^{\log_h a}$ for any $a, b \in \langle g \rangle$ and $h \in \langle g \rangle^*$.

1.2.3 PROBABILITY THEORY

Throughout, we will use basic notions from probability theory, such as sample space, events, probability distributions, and random variables. We will only be concerned with discrete random variables. Specifically, for a (nonempty) finite set V, we write $X \in_R V$ to define X as a random variable distributed uniformly at random on V, that is, $\Pr[X = v] = 1/|V|$ for all $v \in V$. Furthermore, the notion of statistical distance plays a fundamental role.

DEFINITION 1.9 *The **statistical distance**[3] $\Delta(X;Y)$ between random variables X and Y is defined as*

$$\Delta(X;Y) = \frac{1}{2} \sum_{v \in V} \big| \Pr[X = v] - \Pr[Y = v] \big|,$$

where V denotes the set of possible values for X and Y.

For our purposes, it suffices to consider the case that V is finite.

Note that Definition 1.9 can also be read as the statistical distance between the *(probability) distributions* of X and Y. So $\Delta(X;Y) = 0$ if and only if the probability distributions of X and Y are identical, that is, $\Pr[X = v] = \Pr[Y = v]$ for all $v \in V$. In general, this is not

[3] This quantity is also known as, among other names, the *(total) variation(al) distance*, *Kolmogorov distance*, *trace distance*, *L_1-distance*, or *Manhattan distance*, possibly without the scaling factor $\frac{1}{2}$.

equivalent to equality of X and Y as random variables: for example, let $X \in_R \{0,1\}$ and let $Y = 1 - X$, then $\Delta(X;Y) = 0$ but clearly $X \neq Y$ (in fact, $\Pr[X = Y] = 0$).

The statistical distance is a bounded metric in the following sense.

PROPOSITION 1.10 *For random variables X, Y, and Z:*

(i) $0 \leq \Delta(X;Y) \leq 1$, "nonnegativity" *and* "boundedness"

(ii) $\Delta(X;Y) = 0$ *if and only if* $\forall_{v \in V} \Pr[X = v] = \Pr[Y = v]$, "identical distributions"

(iii) $\Delta(X;Y) = \Delta(Y;X)$, "symmetry"

(iv) $\Delta(X;Z) \leq \Delta(X;Y) + \Delta(Y;Z)$. "triangle inequality"

Clearly, $\Delta(X;X) = 0$. Note that $\Delta(X;Y) = 1$ if and only if X and Y are disjoint, that is, $\Pr[X = v]\Pr[Y = v] = 0$ for all $v \in V$.

Alternative ways to characterize or compute statistical distance are as follows.

PROPOSITION 1.11 *For random variables X and Y:*

(i) $\Delta(X;Y) = \sum_{v \in V^+} \big(\Pr[X = v] - \Pr[Y = v]\big)$, *with* $V^+ = \{v \in V\colon \Pr[X = v] > \Pr[Y = v]\}$,

(ii) $\Delta(X;Y) = \sum_{v \in V} \big(\Pr[X = v] \dot{-} \Pr[Y = v]\big)$, *with* $x \dot{-} y = \max(x - y, 0)$ *("x monus y")*,

(iii) $\Delta(X;Y) = 1 - \sum_{v \in V} \min\big(\Pr[X = v], \Pr[Y = v]\big)$,

(iv) $\Delta(X;Y) = \max_{W \subseteq V} \big|\Pr[X \in W] - \Pr[Y \in W]\big|$.

EXERCISE 1.12 (a) Prove Proposition 1.10. (b) Prove Proposition 1.11.

EXERCISE 1.13 Prove that $\Delta(f(X); f(Y)) \leq \Delta(X;Y)$ for any function f defined on V.

EXERCISE 1.14 For $n, d \geq 1$, consider distributions X and Y given by

$$\begin{aligned} X &= \{u : u \in_R \{0, \ldots, n-1\}, \\ Y &= \{u + d : u \in_R \{0, \ldots, n-1\}\}. \end{aligned}$$

Determine $\Delta(X;Y)$, assuming $d \leq n$. Also, what is $\Delta(X;Y)$ if $d > n$?

EXERCISE 1.15 For $n \geq 1$, consider distributions X, Y, Z given by

$$\begin{aligned} X &= \{u : u \in_R \{0, \ldots, n-1\}\}, \\ Y &= \{2u : u \in_R \{0, \ldots, n-1\}\}, \\ Z &= \{2u + 1 : u \in_R \{0, \ldots, n-1\}\}. \end{aligned}$$

Show that $\Delta(Y;Z) = 1$. Show that $\Delta(X;Y) = \Delta(X;Z) = 1/2$ for even n, and also determine $\Delta(X;Y)$ and $\Delta(X;Z)$ for odd n.

EXERCISE 1.16 For $n \geq 1$, let $X \in_R \mathbb{Z}_n$ and $Y \in_R \mathbb{Z}_n^*$. (a) Determine $\Delta(X;Y)$. (b) Show that $\Delta(X + Y; XY) = 0$, where addition and multiplication are done modulo n.

EXERCISE 1.17 For n prime, let h and M_0 be arbitrary, fixed elements of $G_n = \langle g \rangle$, $h \neq 1$. Consider distributions X, Y, and Z given by

$$\begin{aligned} X &= \{(A, B) : A \in_R \langle g \rangle, B \in_R \langle g \rangle\}, \\ Y &= \{(g^u, h^u M) : u \in_R \mathbb{Z}_n, M \in_R \langle g \rangle\}, \\ Z &= \{(g^u, h^u M_0) : u \in_R \mathbb{Z}_n\}. \end{aligned}$$

Show that $\Delta(X;Y) = 0$ and $\Delta(Y;Z) = 1 - 1/n$. Show that also $\Delta(X;Z) = 1 - 1/n$ (e.g., using triangle inequalities).

EXERCISE 1.18 For $n \geq d \geq 1$, let random variable X take on values in $\{0, \ldots, d-1\}$, and let $U \in_R \{0, \ldots, n-1\}$. Show that $\Delta(U; X+U) \leq (d-1)/n$, and that this bound is tight.

The result of Exercise 1.18 implies that $\Delta(U; X+U)$ is small if $d \ll n$. For instance, if one sets $n = d2^k$, we see that the statistical distance between U and $X+U$ is less than $1/2^k$, hence approaches 0 exponentially fast as a function of k. In other words, one can mask an integer value X from a bounded range $\{0, \ldots, d-1\}$ by adding a uniformly random integer U from an enlarged range $\{0, \ldots, n-1\}$. This way one can do one-time pad encryption with integers, where X is the plaintext, U is the one-time pad, and $X+U$ is the ciphertext.

1.2.4 COMPLEXITY THEORY

Familiarity with basic notions from (computational) complexity theory is assumed. Examples are the notion of a Turing machine and the complexity classes P and NP. Below, we briefly discuss the essential role of *randomized* complexity in cryptology.

Taking Church's thesis for granted, Turing machines are powerful enough to describe any kind of algorithm. A basic **Turing machine** consists of a (finite) control part and an (infinite) tape used to store data. Many variations exist, e.g., Turing machines with multiple tapes (possibly divided into input tapes, work tapes, output tapes, etc.), with one-way infinite vs. two-way infinite tapes, tapes with several heads instead of just one, and so on. The basic Turing machine, however, already captures the essence of an algorithm, and thereby defines the borderline between problems which are computable and which are not (and, which languages are decidable or not, acceptable or not). In particular, the computability of a problem does not depend on whether a Turing machine is **deterministic**, hence on each step moves from its current configuration to a *unique* successor configuration, or **non-deterministic**, hence moves on each step to *one of several possible* successor configurations.

When efficiency is taken into account, the situation changes quite dramatically. To define *efficient* algorithms one uses some form of time-bounded Turing machine. A Turing machine is said to be **polynomial time** if it halts within $p(|x|)$ steps on any input string x, where p denotes some polynomial and $|x|$ denotes the length of string x. The **complexity class P** is then defined as the set of all problems which can be solved by a **deterministic polynomial time** Turing machine. In terms of class P one may say that a problem is feasible (can be handled efficiently) exactly when it is in P.

The issue with characterizing efficiency in terms of P is that the feasibility of a problem is thus considered in terms of its *worst-case* complexity only. For cryptographic and for cryptanalytic purposes, however, it is much more relevant to consider *average case* complexity instead. The critical notion is that of a **probabilistic** Turing machine, which behaves similarly to a non-deterministic Turing machine, except that on each step it chooses the successor

configuration *uniformly at random* from the possible ones. Hence, an operational way of viewing a probabilistic Turing machine is to think of one of its tapes as the "random tape," which is a read-only tape containing uniformly random bits. A problem is now considered feasible if it can be solved by a **probabilistic polynomial time (p.p.t.)** Turing machine.[4]

The algorithms constituting a cryptographic scheme are thus all understood to be p.p.t. algorithms. This is important because efficient algorithms for tasks such as primality testing[5] and prime number generation are probabilistic (randomized). More importantly, the adversary is also viewed as a p.p.t. algorithm, which means that it does not suffice to show that the security of a cryptographic scheme is guaranteed if an underlying problem (e.g., the discrete log problem) is not in P. Instead we need problems that are not in the **complexity class BPP**, which are those problems that can be solved with bounded-error by a p.p.t. Turing machine. Without going into details, we mention that a decision problem is said to be solved with bounded-error by a p.p.t. Turing machine if for every YES-instance x, the probability (over all internal random choices) of accepting it is at least $2/3$, and if for every NO-instance x, this probability is at most $1/3$.[6] Throughout, we will thus say that a problem is "easy" (or, "feasible") if it is in BPP, hence can be solved by a p.p.t. algorithm, and we say it is "hard" (or, "infeasible") otherwise. Thus, "hard" means that any efficient algorithm only succeeds with negligibly small probability in computing the correct output.

1.3 ASSUMPTIONS

1.3.1 DISCRETE LOG AND DIFFIE-HELLMAN ASSUMPTIONS

The following three hardness assumptions are commonly used in cryptographic schemes based on a discrete log setting.

DEFINITION 1.19 *The **Discrete Logarithm (DL) assumption** for group $\langle g \rangle$ states that it is hard to compute x given a random group element g^x.*

DEFINITION 1.20 *The **Diffie-Hellman (DH) assumption** for group $\langle g \rangle$ states that it is hard to compute g^{xy} given random group elements g^x and g^y.*

DEFINITION 1.21 *The **Decisional Diffie-Hellman (DDH) assumption** for group $\langle g \rangle$ states that it is hard to distinguish g^{xy} from a random group element g^z given random group elements g^x and g^y.*

[4] Throughout these lecture notes we ignore the distinction between *strict* polynomial time and *expected* polynomial time. For strict polynomial time, a probabilistic Turing machine must halt on each input after a polynomial number of steps (as a function of the input size), whereas for expected polynomial time this only needs to hold for each input on the average (averaged over all random choices made by the Turing machine).

[5] The Miller-Rabin test is a well-known example. Note that until the discovery of the AKS algorithm in 2002, it was not known that primality testing is actually in P.

[6] The role of BPP should not be confused with the role of the **complexity class NP**, which is defined as the set of all problems which can be solved by a **non-deterministic polynomial time** Turing machine. Here, the time needed by a non-deterministic Turing machine on a given input string is defined as the *least* number of steps before it halts (as if one—by magic—always makes the right choice to solve a problem instance as quickly as possible). The big open question is whether P = NP, or not. If indeed P $\neq$ NP, then it would follow that the so-called NP-complete problems take more than polynomial time to solve in the worst case—but, this says nothing about the average case.

The DH assumption is sometimes called the Computational Diffie-Hellman (CDH) assumption to stress the difference with the DDH assumption.

Evidently, these assumptions satisfy: DL $\Leftarrow$ DH $\Leftarrow$ DDH. Therefore it is better if a scheme can be proved secure under just the DL assumption. It turns out, however, that in many cases the security can only be proved under the DH assumption, or even only under the DDH assumption.[7]

1.3.2 INDISTINGUISHABILITY

DEFINITION 1.22 *A nonnegative function $f : \mathbb{N} \to \mathbb{R}$ is called* ***negligible*** *if for every $\gamma \in \mathbb{N}$ there exists a $k_0 \in \mathbb{N}$ such that for all $k \geq k_0$, $f(k) \leq 1/k^\gamma$.*

A typical example of a negligible function is 2^{-k}, which converges exponentially fast to 0. Other examples are $2^{-\sqrt{k}}$ and $k^{-\log k}$. However, $1/k$ and $1/k^{100}$ are clearly not negligible. Note that $\lim_{k\to\infty} f(k) = 0$ if f is negligible; the converse does not hold in general.

DEFINITION 1.23 *Let $X = \{X_i\}_{i\in I}$ and $Y = \{Y_i\}_{i\in I}$ be two families of random variables, or probability distributions, indexed by I. (Suppose that $|X_i| = |Y_i|$ for all $i \in I$, and that these sizes are both polynomial in $|i|$.) Then X and Y are said to be:*

(i) ***perfectly indistinguishable*** *if $\Delta(X_i; Y_i) = 0$ (hence* identically *distributed);*

(ii) ***statistically indistinguishable*** *if $\Delta(X_i; Y_i)$ is negligible as a function of $|i|$;*

(iii) ***computationally indistinguishable*** *if $\Delta(D(X_i); D(Y_i))$ is negligible as a function of $|i|$ for every p.p.t. algorithm D with output in $\{0, 1\}$ (**Boolean distinguisher**).*

Equivalently, X and Y are computationally indistinguishable if for every distinguisher D, the **advantage** $\mathrm{Adv}_D(X_i, Y_i)$ is negligible,[8] where

$$\mathrm{Adv}_D(X_i, Y_i) = \big|\Pr[D(X_i) = 1] - \Pr[D(Y_i) = 1]\big|.$$

This equivalence follows from Proposition 1.11(iv): clearly, $\mathrm{Adv}_D(X_i, Y_i) \leq \Delta(D(X_i); D(Y_i))$; moreover, for any distinguisher D, one obtains a distinguisher D' with $\mathrm{Adv}_{D'}(X_i, Y_i) = \Delta(D(X_i); D(Y_i))$ by defining $D'(v) = 1$ if and only if $D(v) \in W$, where W is any set maximizing $|\Pr[D(X_i) \in W] - \Pr[D(Y_i) \in W]|$.

In practice, one often uses Boolean distinguishers D with output $D(v) \in \{0, 1\}$.

EXERCISE 1.24 Show that $\mathrm{Adv}_D(X_i, Y_i) = \Delta(D(X_i); D(Y_i))$ for any Boolean distinguisher D.

EXERCISE 1.25 Show that computational indistinguishability is implied by statistical indistinguishability. Hint: use Proposition 1.11(iv), cf. Exercise 1.13.

---

[7] In some cases one can prove that the DH assumption is equivalent to the DL assumption. The DDH assumption, though, appears to be stronger than the DH assumption. In fact, there exist groups based on elliptic curves for which the DDH assumption is false, while the DH problem is considered hard. For these particular groups the DDH problem is actually easy.

[8] A nonnegative family $\{f_i \in \mathbb{R}\}_{i\in I}$ of values is called *negligible (as a function of $|i|$)* if for every $\gamma \in \mathbb{N}$ there exists a $k_0 \in \mathbb{N}$ such that for all $i \in I$ with $|i| \geq k_0$, $f_i \leq 1/|i|^\gamma$, cf. Definition 1.22.

A more formal version of the DDH assumption, stated in terms of computationally indistinguishable distributions, can now be rendered as follows. The DDH assumption is obtained by letting index set I correspond to groups $G_n = \langle g \rangle$. For $i = \langle g \rangle$, the size $|i|$ is defined as the length of the bit string representing $\langle g \rangle$. The distributions are given by

$$\begin{aligned} X_{\langle g \rangle} &= \{(g^x, g^y, g^{xy}) : x, y \in_R \mathbb{Z}_n\}, && \text{“DH triples”} \\ Y_{\langle g \rangle} &= \{(g^x, g^y, g^z) : x, y, z \in_R \mathbb{Z}_n, z \neq xy\}. && \text{“non-DH triples”} \end{aligned}$$

The DDH assumption is then that $X_{\langle g \rangle}$ and $Y_{\langle g \rangle}$ are computationally indistinguishable.

Note that it does not matter whether we take distribution $Y_{\langle g \rangle}$ or the related distribution $Y'_{\langle g \rangle}$ given by

$$Y'_{\langle g \rangle} = \{(g^x, g^y, g^z) : x, y, z \in_R \mathbb{Z}_n\}, \qquad \text{“random triples”}$$

since these two distributions are statistically indistinguishable. This can be verified as follows, using Definition 1.9:

$$\begin{aligned} \Delta(Y; Y') &= \tfrac{1}{2}\Big(\textstyle\sum_{x,y,z \in \mathbb{Z}_n} \big|\Pr[Y = (g^x, g^y, g^z)] - \Pr[Y' = (g^x, g^y, g^z)]\big|\Big) \\ &= \tfrac{1}{2}\Big(\textstyle\sum_{x,y,z \in \mathbb{Z}_n, z \neq xy} \big|\Pr[Y = (g^x, g^y, g^z)] - \Pr[Y' = (g^x, g^y, g^z)]\big| \\ &\quad + \textstyle\sum_{x,y,z \in \mathbb{Z}_n, z = xy} \big|\Pr[Y = (g^x, g^y, g^z)] - \Pr[Y' = (g^x, g^y, g^z)]\big|\Big) \\ &= \tfrac{1}{2}\Big((n^3 - n^2)\big|\tfrac{1}{n^3 - n^2} - \tfrac{1}{n^3}\big| + n^2\big|0 - \tfrac{1}{n^3}\big|\Big) \\ &= 1/n. \end{aligned}$$

Since n is exponentially large, the statistical distance is negligible. By saying that n is exponentially large, we mean that $\log_2 n \approx k$ for some security parameter k. Typically, $k = 256$ is recommended for discrete log based cryptography. (For RSA based cryptography, see also Exercise 1.34, k is typically the bit length of the RSA modulus m, e.g., $k = 2048$ or larger. Note that one cannot simply compare these security parameters across different cryptosystems.)

Note that the statistical distance between $X_{\langle g \rangle}$ and $Y_{\langle g \rangle}$ is given by

$$\begin{aligned} \Delta(X; Y) &= \tfrac{1}{2}\Big(\textstyle\sum_{x,y,z \in \mathbb{Z}_n} \big|\Pr[X = (g^x, g^y, g^z)] - \Pr[Y = (g^x, g^y, g^z)]\big|\Big) \\ &= \tfrac{1}{2}\Big(\textstyle\sum_{x,y,z \in \mathbb{Z}_n, z = xy} \big|\Pr[X = (g^x, g^y, g^z)] - \Pr[Y = (g^x, g^y, g^z)]\big| \\ &\quad + \textstyle\sum_{x,y,z \in \mathbb{Z}_n, z \neq xy} \big|\Pr[X = (g^x, g^y, g^z)] - \Pr[Y = (g^x, g^y, g^z)]\big|\Big) \\ &= \tfrac{1}{2}\Big(n^2\big|\tfrac{1}{n^2} - 0\big| + (n^3 - n^2)\big|0 - \tfrac{1}{n^3 - n^2}\big|\Big) \\ &= 1. \end{aligned}$$

Hence, $\Delta(X; Y)$ is maximal, which is no surprise as $X_{\langle g \rangle}$ and $Y_{\langle g \rangle}$ are disjoint. Yet, the distributions cannot be distinguished from each other, under the DDH assumption. By the triangle inequality for statistical distance, it follows that

$$\Delta(X; Y') \geq \Delta(X; Y) - \Delta(Y; Y') = 1 - 1/n,$$

hence also the distance between $X_{\langle g \rangle}$ and $Y'_{\langle g \rangle}$ is non-negligible.

EXERCISE 1.26 Check that actually $\Delta(X; Y') = 1 - 1/n$.

1.3.3 RANDOM SELF-REDUCIBILITY

The following notion of random self-reducible problems is sufficiently general for our purposes. Recall that one commonly refers to the input of a problem and a corresponding output as a **(problem) instance** and its **solution**.

DEFINITION 1.27 *A problem is called* ***(perfectly) random self-reducible*** *if any instance I of the problem can be solved by these three steps:*

1. *Transform instance I into a* uniformly random *instance I'.*
2. *Solve instance I'.*
3. *Extract the solution for I from the solution for I'.*

Only steps 1 and 3 are required to run in polynomial time.

For cryptographic purposes, it is a good sign if a presumably hard problem is random self-reducible. In that case, it is excluded that even though the problem is hard in the worst-case, the problem is actually easy on the average. To see why, consider a random self-reducible problem and suppose that the problem is easy on the average. Then there cannot be any hard instances at all, since any such instance can be solved by solving a transformed, uniformly random instance due to random self-reducibility.

PROPOSITION 1.28 *Any random self-reducible problem that is hard in the worst-case is also hard on the average.*

It is reassuring that the standard discrete log and Diffie-Hellman problems are all random self-reducible, where these problems are formulated as follows, for any group $\langle g \rangle$ of order n:

DL problem: compute x, given g^x with $x \in \mathbb{Z}_n$.

DH problem: compute g^{xy}, given g^x, g^y with $x, y \in \mathbb{Z}_n$.

DDH problem: distinguish g^{xy} from g^z, given g^x, g^y with $x, y, z \in \mathbb{Z}_n$, $z - xy \in \mathbb{Z}_n^*$.

Note that for n prime the DDH problem corresponds to distinguishing the (disjoint) distributions $X_{\langle g \rangle} = \{(g^x, g^y, g^{xy}) : x, y \in_R \mathbb{Z}_n\}$ and $Y_{\langle g \rangle} = \{(g^x, g^y, g^z) : x, y, z \in_R \mathbb{Z}_n, z \neq xy\}$, as explained in Section 1.3.2.

The above problem formulations allow for *arbitrary*—potentially, worst-case—problem instances. In cryptography, however, we generally need to allow for *random* problem instances, cf. the hardness assumptions in Definitions 1.19–1.21. The following proposition implies that these assumptions are not stronger than their worst-case cousins.

PROPOSITION 1.29 *The DL, DH, and DDH problems are random self-reducible.*

PROOF For the DL problem, any given instance $h = g^x$ with $x \in \mathbb{Z}_n$ is solved as follows:

1. Transform h into a uniformly random instance $h' = hg^u$ with $u \in_R \mathbb{Z}_n$.
2. Solve instance h' yielding $x' = \log_g h'$.
3. Extract the solution as $x = \log_g h = x' - u \bmod n$.

Clearly, h' is distributed uniformly on $\langle g \rangle$.

For the DH problem, any given instance $I = (g^x, g^y)$ with $x, y \in \mathbb{Z}_n$ is solved as follows:

1. Transform I into a uniformly random $I' = (g^{x'}, g^{y'}) = (g^x g^t, g^y g^u)$ with $t, u \in_R \mathbb{Z}_n$.
2. Solve instance I' yielding $g^{x'y'} = g^{(x+t)(y+u)} = g^{xy+xu+ty+tu}$.
3. Extract the solution as $g^{xy} = g^{x'y'}/((g^x)^u (g^y)^t g^{tu})$.

Note that indeed the pair $(g^{x'}, g^{y'})$ is distributed uniformly on $\langle g \rangle \times \langle g \rangle$.

For the DDH problem, random self-reducibility amounts to transforming each triple given as input to the Boolean distinguisher into a uniformly-random triple of the "same type." Given any instance $I = (g^x, g^y, g^z)$, we do so as follows:

1. Transform I into $I' = ((g^x)^s g^t, g^y g^u, (g^z)^s (g^x)^{su} (g^y)^t g^{tu})$ with $s \in_R \mathbb{Z}_n^*$ and $t, u \in_R \mathbb{Z}_n$.
2. Solve instance I' yielding bit b'.
3. Output $b = b'$.

First, note that $s \in_R \mathbb{Z}_n^*$ can be generated in polynomial time by testing on average $n/\phi(n) = O(\log \log n)$ uniformly random values $s \in \mathbb{Z}_n$ until $\gcd(s, n) = 1$; see also Section 1.2.1.

Next, let $I' = (g^{x'}, g^{y'}, g^{z'})$. Then instances I and I' are of the same type as

$$z' - x'y' = zs + xsu + yt + tu - (xs + t)(y + u) = (z - xy)s,$$

hence $z = xy$ if and only if $z' = x'y'$, using that $s \in \mathbb{Z}_n^*$.

Furthermore, if $z = xy$, triple I' is uniform among all DH triples $(g^{x'}, g^{y'}, g^{x'y'})$, independent of I. That is, for any values $v, w \in \mathbb{Z}_n$:

$$\Pr[x' = v, y' = w, z' = vw] = \frac{1}{n^2}.$$

Similarly, if $z - xy \in \mathbb{Z}_n^*$, then s, t, u are determined uniquely by $s = (z' - x'y')/(z - xy)$, $t = x' - xs$, and $u = y' - y$, implying that I' is uniform among all non-DH triples $(g^{x'}, g^{y'}, g^{z'})$ with $z' - x'y' \in \mathbb{Z}_n^*$, independent of I as well. That is,

$$\Pr[x' = v, y' = w, z' = r] = \Pr[s = \frac{r - vw}{z - xy}, t = v - xs, u = w - y] = \frac{1}{\phi(n) n^2},$$

for any values $r, v, w \in \mathbb{Z}_n$ with $r - vw \in \mathbb{Z}_n^*$. □

We will often use the following variants of the discrete log and Diffie-Hellman problems:

DL* problem: compute x, given g^x with $x \in \mathbb{Z}_n^*$.

DH* problem: compute g^{xy}, given g^x, g^y with $x \in \mathbb{Z}_n^*$ and $y \in \mathbb{Z}_n$.

DH** problem: compute g^{xy}, given g^x, g^y with $x, y \in \mathbb{Z}_n^*$.

DDH* problem: distinguish g^{xy} from g^z, given g^x, g^y with $x \in \mathbb{Z}_n^*$, $y, z \in \mathbb{Z}_n$, $z - xy \in \mathbb{Z}_n^*$.

DDH** problem: distinguish g^{xy} from g^z, given g^x, g^y with $x, y, z \in \mathbb{Z}_n^*$, $z - xy \in \mathbb{Z}_n^*$.

Since $\mathbb{Z}_n^* = \mathbb{Z}_n \setminus \{0\}$ in the common case that the group order n is prime, the only difference with the basic problems is that $1 \in \langle g \rangle$ is not a valid input anymore in some cases. Using slightly different transformations, each of these variants can be seen to be random self-reducible as well, except that for the DDH** problem we need to assume that the prime factorization of n is given.

EXAMPLE 1.30 *It is easy to see that the DL* problem is random self-reducible as follows. Given any instance $h = g^x$ with $x \in \mathbb{Z}_n^*$ (hence, $h \in \langle g \rangle^*$):*

1. *Transform h into a uniformly random instance $h' = h^u$ with $u \in_R \mathbb{Z}_n^*$.*
2. *Solve instance h' yielding $x' = \log_g h'$.*
3. *Extract the solution as $x = \log_g h = x'/u \bmod n$.*

This time h' is distributed uniformly on $\langle g \rangle^*$.

As an alternative approach, the transformation used for the DL problem in the proof of Proposition 1.29 can also be used here. In this case, one tests in addition in step 1 if $\mathrm{ord}(h') = n$ to make sure h' is a valid input in step 2 satisfying $h' \in \langle g \rangle^*$, cf. Exercise 1.7. However, to ensure that the test $\mathrm{ord}(h') = n$ can be evaluated efficiently, we assume that we know the prime factorization of n. Then, given any instance $h = g^x$ with $x \in \mathbb{Z}_n^*$:

1. *Transform h into a uniformly random instance $h' = hg^u$ with $u \in_R \mathbb{Z}_n$ until $\mathrm{ord}(h') = n$.*
2. *Solve instance h' yielding $x' = \log_g h'$.*
3. *Extract the solution as $x = \log_g h = x' - u \bmod n$.*

In step 2, h' is distributed uniformly on $\langle g \rangle^*$ as required. To see that the (expected) running time of step 1 is polynomial in the size of n, we note that on average $n/\phi(n) = O(\log \log n)$ uniformly random $h' \in \langle g \rangle$ are needed to find one $h' \in \langle g \rangle^*$.

EXERCISE 1.31 Show that (a) the DH* problem is random self-reducible, and (b) the DH** problem is random self-reducible.

EXERCISE 1.32 Show that (a) the DDH* problem is random self-reducible, and (b) given the prime factorization of n, the DDH** problem is random self-reducible. Hints: for both parts use three random numbers for the transformation in step 1, $s, t \in_R \mathbb{Z}_n^*$ and $u \in_R \mathbb{Z}_n$; for part (b), also test for invalid inputs, in the same way as at the end of Example 1.30.

EXERCISE 1.33 Show that the following problems are random self-reducible for any group $\langle g \rangle$ of order n:

(a) given g^x with $x \in \mathbb{Z}_n$, compute g^{x^2};

(b) given g^x with $x \in \mathbb{Z}_n^*$, compute $g^{1/x}$;

(c) given g^x, g^y with $x, y \in \mathbb{Z}_n^*$, compute $g^{x/y}$;

(d) given g^x, g^y with $x \in \mathbb{Z}_n$, $y \in \mathbb{Z}_n^*$, compute $g^{x/y}$;

(e) given g^x with $x \in \mathbb{Z}_n^*$, compute g^{x^3};

(f) given g^x, g^{x^2} with $x \in \mathbb{Z}_n$, compute g^{x^3};

(g) given g^x, g^y with $x, y \in \mathbb{Z}_n$, compute $g^{(x+y)^2}$;

(h) given g^x, g^y with $x, y \in \mathbb{Z}_n, x - y \in \mathbb{Z}_n^*$, compute $g^{1/(x-y)}$.

EXERCISE 1.34 Let $m = pq$ be an RSA modulus, that is, p and q are large, distinct primes of bit length k, for some integer k. Let integer $e > 1$ satisfy $\gcd(e, \phi(m)) = 1$, where $\phi(m) = (p-1)(q-1)$. The RSA problem is to compute $x = y^{1/e} \bmod m$ given $y \in \mathbb{Z}_m^*$. Show that the RSA problem is random self-reducible.

EXERCISE 1.35 Let m be an RSA modulus, as in the previous exercise. Let $J_m = \{y \in \mathbb{Z}_m^* : (y/m) = 1\}$, the set of all integers in $\mathbb{Z}_m^*$ with Jacobi symbol 1. The Quadratic Residuosity (QR) problem is to decide whether a given $y \in J_m$ is a quadratic residue modulo m or not, that is, whether $y \in QR_m$, where $QR_m = \{y \in \mathbb{Z}_m^* : \exists_{x \in \mathbb{Z}_m^*} y = x^2 \bmod m\}$. Show that the QR problem is random self-reducible.

1.3.4 RANDOM ORACLE MODEL

DEFINITION 1.36 *A function $H : \{0,1\}^* \to \{0,1\}^k$, mapping bit strings of arbitrary length to bit strings of a fixed length k, $k \geq 0$, is called a* ***hash function****. Function H is called a* ***cryptographic*** *hash function, if it is easy to compute $H(x)$ given any string x, and one or more of the following requirements are satisfied:*

- ***preimage resistance (onewayness):*** *given a k-bit string y, it is hard to find a bit string x such that $H(x) = y$.*
- ***2nd-preimage resistance (weak collision resistance):*** *given a bit string x, it is hard to find a bit string $x' \neq x$ such that $H(x') = H(x)$.*
- ***collision resistance (strong collision resistance):*** *it is hard to find a pair of bit strings (x, x') with $x \neq x'$ such that $H(x) = H(x')$.*

In general, collision resistance implies 2nd-preimage resistance, but collision resistance need not imply preimage resistance. In practice, however, cryptographic hash functions usually satisfy all three requirements.

Practical examples of cryptographic hash functions are MD5, SHA-1, SHA-256, with output lengths $k = 128$, $k = 160$, and $k = 256$, respectively.[9] If collision resistance is not required, one may *truncate* the outputs by discarding, e.g., the last $k/2$ bits; the resulting hash function is still preimage resistant.

Many protocols make use of a cryptographic hash function. In order to be able to prove anything useful on the security of such protocols one commonly uses the so-called **random oracle model**. In this model, a cryptographic hash function is viewed as a random oracle, which when queried will behave as a black box containing a *random function* $\mathcal{H} : \{0,1\}^* \to \{0,1\}^k$, say. If the oracle is queried on an input value x, it will return the output value $\mathcal{H}(x)$. As a consequence, if the oracle is queried multiple times on the same input, it will return the same output value, as $\mathcal{H}$ is a function. Moreover, if one observes the distribution of the output value for different input values, the distribution will be uniform, as $\mathcal{H}$ is a random function.

Note that the use of the random oracle model is a heuristic! If we prove a protocol secure in the random oracle model, it does not follow that the same protocol using, e.g., SHA-256 as its hash function is secure, since SHA-256 is simply not a random function.[10]

Thus, the practical upshot of the random oracle model is that a protocol proved secure

[9] In August 2004, collisions for MD5 have (finally) been shown; furthermore, collisions for SHA-0 (a previous version of SHA-1) have also been found. See this Mathematica notebook for some example collisions.

[10] It is even possible to construct protocols that can be proved secure in the random oracle model, but are insecure when used with some concrete hash function. Yet, these "counterexamples" are not realistic. See also Exercise 5.22.

in it can only be broken if the attacker takes into account specific properties of the concrete hash function used.

The following proposition confirms that finding collisions is indeed easier than finding (2nd-)preimages.

PROPOSITION 1.37 *Let H be a cryptographic hash function, which we view as a random oracle. Let $\mathcal{E}$ be an (unlimitedly powerful) adversary that makes at most t hash queries.*

(i) *Let $y \in \{0,1\}^k$ be given. The probability that $\mathcal{E}$ finds a preimage x such that $H(x) = y$ is at most $t/2^k$.*

(ii) *Let $x \in \{0,1\}^*$ be given. The probability that $\mathcal{E}$ finds a 2nd-preimage x', $x' \neq x$, such that $H(x') = H(x)$ is at most $t/2^k$.*

(iii) *The probability that $\mathcal{E}$ finds a collision (x, x'), $x \neq x'$, such that $H(x) = H(x')$ is at most $t^2/2^k$.*

PROOF Without loss of generality, assume that all hash queries are distinct.

(i) For each hash query, the probability of an output equal to y is exactly 2^{-k}.

(ii) Let $y = H(x)$. For each hash query on inputs different from x, the probability of an output equal to y is again exactly 2^{-k}.

(iii) Write $N = 2^k$. If $t \geq \sqrt{N}$, then the bound is immediate. So assume $t < \sqrt{N}$. For hash queries on inputs $x_1, \ldots, x_t$, the probability of at least one collision is equal to

$$1 - N(N-1)(N-2)\cdots(N-t+1)/N^t.$$

This probability is bounded above by

$$\begin{array}{rl} & 1 - (1-1/N)(1-2/N)\cdots(1-(t-1)/N) \\ \leq & 1 - e^{-2/N}e^{-4/N}\ldots e^{-2(t-1)/N} \\ = & 1 - e^{-t(t-1)/N} \\ \leq & 1 - (1 - t(t-1)/N) \\ = & t(t-1)/N \\ \leq & t^2/N, \end{array}$$

using that $1 - x \geq e^{-2x}$ for $0 \leq x \leq 3/4$ and that $e^x \geq 1 + x$ for all $x \in \mathbb{R}$.

□

EXERCISE 1.38 See the proof of Proposition 1.37(iii). Show that the upper bound for the probability of finding at least one collision is almost tight by showing that this probability is bounded below by $t(t-1)/4N$, for $t < \sqrt{N}$.

There are numerous further requirements that can be demanded of a cryptographic hash function beyond what is stated in Definition 1.36. In general, any deviation from what can be statistically expected of a random function should be absent or unlikely, and in any case it should be infeasible to exploit such statistical weaknesses. By definition, the random oracle model is robust w.r.t. such additional requirements. For example, *partial preimage resistance*

(or, *local onewayness*) basically states that given a k-bit string y it is hard to find (partial) information about any input x satisfying $H(x) = y$. Many applications of hash functions, such as the bit commitment scheme of Section 3.2.1, rely on this requirement rather than the (much weaker) requirement of preimage resistance. Clearly, partial preimage resistance also holds in the random oracle model, in which each hash value is, by definition, statistically independent of the input value.

EXERCISE 1.39 Let H be a preimage resistant hash function. Show that partial preimage resistance is strictly stronger than preimage resistance, by constructing a preimage resistant hash function H' (from H) which is not partial preimage resistant.

EXERCISE 1.40 Suppose one demands of a hash function H that it is hard to find a pair of bit strings (x, x') satisfying $H(x) = \overline{H(x')}$, where $\overline{s}$ denotes the bitwise complement of bit string s. Analyze the probability that an adversary $\mathcal{E}$ making at most t hash queries finds such a pair, where H is viewed as a random oracle.

EXERCISE 1.41 Let $y = H^2(x)$, where $x \in \{0,1\}^*$, and assume that $y \neq H(x)$. Let $\mathcal{E}$ be an adversary that, given y only, makes t hash queries on distinct inputs $x_1, \ldots, x_t \in \{0,1\}^k \setminus \{y\}$. View $H : \{0,1\}^* \to \{0,1\}^k$ as a random oracle to show that $\mathcal{E}$ finds a preimage of y with probability *exactly* equal to $\epsilon(2-\epsilon)$, with $\epsilon = t/2^k$. Also, argue why there is no contradiction with Proposition 1.37(i) even though $\epsilon(2-\epsilon) \geq \epsilon$.

1.4 BIBLIOGRAPHIC NOTES

Most of the notions covered in this chapter have become standard, and are widely used in cryptography. We highlight a few of the more recent developments.

The DL assumption has been known for a long time, and nowadays many groups have been identified for which the DL problem is assumed to be hard. As for the examples given in the text, we note that $\mathbb{Z}_p^*$ and $\mathbb{F}_q^*$ (Example 1.4) are classical instances. The particular advantages of using a (prime) order p' subgroup of $\mathbb{Z}_p^*$ (Example 1.5) were identified by Schnorr [Sch91], noting that p' can be much smaller than p (e.g., p should be of length at least 2048 bits to counter index-calculus methods, whereas for p' a length of 256 bits suffices to counter Pollard's rho method). The use of elliptic curves for cryptography (Example 1.6) was proposed independently by Koblitz [Kob87] and Miller [Mil86].

The DH assumption was part of the seminal paper by Diffie and Hellman [DH76] introducing the basic ideas of asymmetric cryptography. The explicit statement of the DDH assumption was only identified years later (first in [Bra93]), and is now known to be equivalent to the semantic security of the ElGamal cryptosystem (see Section 2.1.4). The fact that the DDH problem is random self-reducible was found independently by [Sta96] and [NR97]. See also [Bon98] for an overview of the DDH assumption.

The notions of indistinguishability and their use in cryptography date back to the early 1980s, with the papers by Yao [Yao82b] and Goldwasser and Micali [GM84] playing a major role in this respect.

The notion of random self-reducibility was introduced in the context of various cryptographic primitives [AL83, TW87]; it also plays a role in complexity theory. We have used a limited form of random self-reducibility, which suffices for our purposes (see Definition 1.27). More generally, random self-reducibility may comprise solving multiple (polynomially many) random instances $I'_1, I'_2, \ldots$ in order to solve a given instance I, where these instances may

even be determined adaptively (e.g., when I_2' is computed as a function of I and the answer for I_1'); see, e.g., [FF93].

The idea and use of the random oracle model was first elaborated in [BR93], thereby formalizing the use of hash functions in common constructions such as the Fiat-Shamir heuristic (see Section 5.4). Many papers have employed the random oracle model since. Additional requirements for hash functions such as partial preimage resistance are considered throughout the literature; e.g., see [MOV97, p.331], which also mentions non-correlation (of input bits and output bits) and near-collision resistance as specific requirements.

CHAPTER 2

Key Exchange Protocols

2.1 DIFFIE-HELLMAN KEY EXCHANGE

2.1.1 BASIC PROTOCOL

The Diffie-Hellman key exchange protocol enables two parties $\mathcal{A}$ and $\mathcal{B}$ to arrive at a shared key K by exchanging messages over a public channel. Key K remains unknown to any eavesdropper.

The protocol runs as follows. Suppose parties $\mathcal{A}$ and $\mathcal{B}$ have agreed upon a group $G_n = \langle g \rangle$, where we require n to be prime. Party $\mathcal{A}$ picks a value $x_{\mathcal{A}} \in \mathbb{Z}_n^*$ uniformly at random, and sends $h_{\mathcal{A}} = g^{x_{\mathcal{A}}}$ to party $\mathcal{B}$. Similarly, party $\mathcal{B}$ picks a value $x_{\mathcal{B}} \in \mathbb{Z}_n^*$ uniformly at random, and sends $h_{\mathcal{B}} = g^{x_{\mathcal{B}}}$ to party $\mathcal{A}$. Upon receipt of $h_{\mathcal{B}}$, party $\mathcal{A}$ computes key $K_{\mathcal{AB}} = h_{\mathcal{B}}^{x_{\mathcal{A}}}$. Similarly, party $\mathcal{B}$ computes key $K_{\mathcal{BA}} = h_{\mathcal{A}}^{x_{\mathcal{B}}}$.

Clearly, $K = K_{\mathcal{AB}} = K_{\mathcal{BA}}$ is a *shared key for* $\mathcal{A}$ *and* $\mathcal{B}$, which means (i) that it is the same for $\mathcal{A}$ and $\mathcal{B}$, and (ii) that it is a *private* key (only known to $\mathcal{A}$ and $\mathcal{B}$), and (iii) that the key is actually equal to $g^{x_{\mathcal{A}} x_{\mathcal{B}}}$, hence uniformly distributed.

EXERCISE 2.1 Explain what happens if the secret exponents $x_{\mathcal{A}}$ and $x_{\mathcal{B}}$ are chosen from $\mathbb{Z}_n$ instead of $\mathbb{Z}_n^*$. Confirm your findings by computing the statistical distance $\Delta(K; K')$, where $K = g^{x_{\mathcal{A}} x_{\mathcal{B}}}$ with $x_{\mathcal{A}}, x_{\mathcal{B}} \in_R \mathbb{Z}_n^*$ and $K' = g^{x'_{\mathcal{A}} x'_{\mathcal{B}}}$ with $x'_{\mathcal{A}}, x'_{\mathcal{B}} \in_R \mathbb{Z}_n$.

2.1.2 PASSIVE ATTACKS

A passive attacker (eavesdropper) learns the values $h_{\mathcal{A}} = g^{x_{\mathcal{A}}}$ and $h_{\mathcal{B}} = g^{x_{\mathcal{B}}}$. Under the DL assumption (Definition 1.19) it is hard to determine $x_{\mathcal{A}}$ and $x_{\mathcal{B}}$ from $h_{\mathcal{A}}$ and $h_{\mathcal{B}}$, respectively. However, this does not guarantee that the value of $K = h_{\mathcal{A}}^{x_{\mathcal{B}}}$ cannot be determined given just $h_{\mathcal{A}}$ and $h_{\mathcal{B}}$. To exclude this possibility we need the DH assumption (Definition 1.20).

A stronger assumption is needed to ensure that an eavesdropper does not learn *any* information whatsoever on K. In general, an eavesdropper may learn some *partial* information on K, while full recovery of K is infeasible. For example, an eavesdropper might be able to determine the parity of K, viewing K as an integer, which would mean that the eavesdropper learns one bit of information. To exclude such possibilities we need the DDH assumption (Definition 1.21). We will now make this more precise.

First we argue why we require n to be prime. Suppose n is composite, say $n = 2p'$, where p' is an odd prime. For any element $h \in \langle g \rangle$, we have $\mathrm{ord}(h) \in \{1, 2, p', 2p'\}$ and $\mathrm{ord}(h)$ is easily computed. We have the following table, where each case occurs approximately with

probability 1/4:

$\mathrm{ord}(h_{\mathcal{A}})$	$\mathrm{ord}(h_{\mathcal{B}})$	$\mathrm{ord}(K)$
p'	p'	p'
p'	$2p'$	p'
$2p'$	p'	p'
$2p'$	$2p'$	$2p'$

Hence, the order of the key K is biased, as $\Pr[\mathrm{ord}(K) = p'] \approx 3/4$ and $\Pr[\mathrm{ord}(K) = 2p'] \approx 1/4$. If K would be generated uniformly at random in $\langle g \rangle$, then we would have $\Pr[\mathrm{ord}(K) = p'] \approx \Pr[\mathrm{ord}(K) = 2p'] \approx 1/2$.

Such a slight deviation in the distribution of K seems innocent. However, suppose key K is used to encrypt a 1-bit plaintext, say $M \in_R \{0,1\}$, using $C = g^M K$ as ciphertext (similar to one-time pad encryption). In that case, an eavesdropper would compute $\mathrm{ord}(C)$. If $\mathrm{ord}(C) = p'$ then $M = 0$ is most likely, and if $\mathrm{ord}(C) = 2p'$ then $M = 1$ is most likely.

EXAMPLE 2.2 *Take $\langle g \rangle = \mathbb{Z}_p^*$ as in Example 1.4, and suppose $p - 1 = 2p'$, where p' is prime. Recall that the even powers of g are quadratic residues modulo p and that the odd powers of g are quadratic non-residues modulo p. Hence, $\mathbb{Z}_p^* = QR_p \cup \overline{QR_p}$, where*

$$QR_p = (\mathbb{Z}_p^*)^2 = \{1, g^2, \ldots, g^{2p'-2}\}, \qquad \overline{QR_p} = \{g, g^3, \ldots, g^{2p'-1}\}.$$

Note that $1 \in QR_p$ and $-1 = g^{p'} \in \overline{QR_p}$. Furthermore, all elements of $QR_p \setminus \{1\}$ are of order p' and all elements of $\overline{QR_p} \setminus \{-1\}$ are of order $2p'$.

We then have the following table:

$h_{\mathcal{A}}$	$h_{\mathcal{B}}$	K
QR_p	QR_p	QR_p
QR_p	$\overline{QR_p}$	QR_p
$\overline{QR_p}$	QR_p	$\overline{QR_p}$
$\overline{QR_p}$	$\overline{QR_p}$	$\overline{QR_p}$

If a plaintext $M \in_R \{0,1\}$ is encrypted as $C = g^M K$, then we get that $\Pr[g^M \in QR_p \mid C \in QR_p] = \Pr[K \in QR_p] = 3/4$ and also that $\Pr[g^M \in \overline{QR_p} \mid C \in \overline{QR_p}] = \Pr[K \in QR_p] = 3/4$. Hence, $M = 0$ is most likely if $C \in QR_p$, and $M = 1$ is most likely if $C \in \overline{QR_p}$.

By actually computing $\mathrm{ord}(K)$ from $\mathrm{ord}(h_{\mathcal{A}})$ and $\mathrm{ord}(h_{\mathcal{B}})$, one can see that $\mathrm{ord}(g^M)$ hence M itself can be determined from $\mathrm{ord}(C)$ in case $n = 2p'$. See also Exercise 2.6.

EXERCISE 2.3 Argue that the DDH assumption is false when n contains a small prime factor.

The simplest and most efficient way to guarantee that n contains no small prime factors is to require that n itself is a sufficiently large prime. As an additional benefit, we have that $\mathbb{Z}_n$ is a field (rather than a ring).

We now wish to show that if an eavesdropper would be able to determine any partial information on key K, then we would get a contradiction with the DDH assumption. For the scope of these lecture notes, we show this for a rather limited scenario only.

We consider *balanced* functions $f : \langle g \rangle \to \{0,1\}$, for which the *a priori* probabilities satisfy $\Pr[f(u) = 0] = \Pr[f(u) = 1] = 1/2$ for $u \in_R \langle g \rangle$. Now, suppose there exists an

attacker $\mathcal{E}$ (which is a p.p.t. algorithm) and a balanced function f for which $\Pr[\mathcal{E}(h_{\mathcal{A}}, h_{\mathcal{B}}) = f(K)] > 1/2+\epsilon$, for some non-negligible value ϵ. Then, the claim is that we have the following distinguisher D for DDH:

$$D(g^x, g^y, g^z) = \begin{cases} 1, & \text{if } \mathcal{E}(g^x, g^y) = f(g^z), \\ 0, & \text{if } \mathcal{E}(g^x, g^y) \neq f(g^z). \end{cases}$$

By the above assumption on $\mathcal{E}$, we have that $\Pr[D(h_{\mathcal{A}}, h_{\mathcal{B}}, K) = 1] = \Pr[\mathcal{E}(h_{\mathcal{A}}, h_{\mathcal{B}}) = f(K)] > 1/2 + \epsilon$. On the other hand, we have that $\Pr[D(h_{\mathcal{A}}, h_{\mathcal{B}}, g^z) = 1] = \Pr[\mathcal{E}(h_{\mathcal{A}}, h_{\mathcal{B}}) = f(g^z)] = 1/2$, since for random z the value of $f(g^z)$ will be distributed uniformly on $\{0,1\}$, independently of the values of $h_{\mathcal{A}}, h_{\mathcal{B}}$. Hence, the advantage for D is bounded below by $1/2 + \epsilon - 1/2 = \epsilon$, which is non-negligible. This contradicts the DDH assumption (Definition 1.21).

EXERCISE 2.4 Extend the above analysis to the case that the *a priori* probabilities are given by $\Pr[f(u) = 0] = p_0$ and $\Pr[f(u) = 1] = p_1 = 1 - p_0$.

2.1.3 A PRACTICAL VARIANT

We now consider the Diffie-Hellman protocol as above, except that the key K is defined as follows:

$$K = H(g^{x_{\mathcal{A}} x_{\mathcal{B}}}),$$

where H is a cryptographic hash function. Clearly, both parties are still able to compute K by first computing $g^{x_{\mathcal{A}} x_{\mathcal{B}}}$ and then applying H. A practical choice for H is the standardized SHA-256 hash function.

The reason for using a hash function H is that even though $g^{x_{\mathcal{A}} x_{\mathcal{B}}}$ will have a sufficient amount of entropy, it cannot be simply used as an AES key, for example. The value of $g^{x_{\mathcal{A}} x_{\mathcal{B}}}$ will be uniformly distributed on the group elements in $\langle g \rangle \setminus \{1\}$, but usually a redundant representation is used for the group elements. For example, if $\langle g \rangle$ is a subgroup of $\mathbb{Z}_p^*$, then elements of $\langle g \rangle$ are usually represented as bit strings of length $\lceil \log_2(p+1) \rceil$, while the order of $\langle g \rangle$ can be much smaller than p. The use of a cryptographic hash function is a practical way to remove this redundancy. The result will be a random bit string as long as the order of the group n sufficiently exceeds 2^k, where k denotes the output length of H, i.e., $H : \{0,1\}^* \to \{0,1\}^k$.

We now argue that the resulting protocol is secure against passive attacks, assuming the DH assumption (Definition 1.20) and the random oracle model (Section 1.3.4). As above, we do so for a limited case, where we show that for any balanced function $f : \{0,1\}^k \to \{0,1\}$ and for any p.p.t. attacker $\mathcal{E}$ that $\Pr[\mathcal{E}(h_{\mathcal{A}}, h_{\mathcal{B}}) = f(K)] \leq 1/2 + \epsilon$, for some negligible value ϵ.

Without loss of generality, we assume that $\mathcal{E}$ queries the random oracle $\mathcal{H}$ on unique inputs only. Let L denote the event that $\mathcal{E}$ queries $\mathcal{H}$ on $g^{x_{\mathcal{A}} x_{\mathcal{B}}}$.

$$\begin{aligned} & \Pr[\mathcal{E}(h_{\mathcal{A}}, h_{\mathcal{B}}) = f(K)] \\ = \; & \Pr[\mathcal{E}(h_{\mathcal{A}}, h_{\mathcal{B}}) = f(K) \mid L] \Pr[L] + \Pr[\mathcal{E}(h_{\mathcal{A}}, h_{\mathcal{B}}) = f(K) \mid \neg L] \Pr[\neg L] \\ \leq \; & \Pr[L] + \Pr[\mathcal{E}(h_{\mathcal{A}}, h_{\mathcal{B}}) = f(K) \mid \neg L] \Pr[\neg L] \\ = \; & \Pr[L] + \tfrac{1}{2}(1 - \Pr[L]) \\ = \; & \tfrac{1}{2} + \tfrac{1}{2} \Pr[L], \end{aligned}$$

using that $\Pr[\mathcal{E}(h_{\mathcal{A}}, h_{\mathcal{B}}) = f(K) \mid \neg L] = 1/2$, since $\mathcal{E}$ has *no information* on $K = \mathcal{H}(g^{x_{\mathcal{A}} x_{\mathcal{B}}})$ if it does not query $\mathcal{H}$ on $g^{x_{\mathcal{A}} x_{\mathcal{B}}}$.

It remains to bound $\Pr[L]$. We show that $\Pr[L]$ is negligible under the DH assumption. Let $\mathcal{E}'$ be an algorithm that takes $h_{\mathcal{A}}$ and $h_{\mathcal{B}}$ as input and runs $\mathcal{E}$ as a subroutine on these inputs, while recording all $\mathcal{H}$ queries made by $\mathcal{E}$. When $\mathcal{E}$ halts, $\mathcal{E}'$ picks one of the inputs used by $\mathcal{E}$ in an $\mathcal{H}$ query at random, and returns this value as output. Clearly, $\mathcal{E}'$ is a probabilistic polynomial time algorithm. We then have

$$\Pr[\mathcal{E}'(h_{\mathcal{A}}, h_{\mathcal{B}}) = g^{x_{\mathcal{A}} x_{\mathcal{B}}}] = \Pr[L]/N,$$

where N denotes the total number of $\mathcal{H}$ queries. Since the running time of $\mathcal{E}$ is polynomial, N is polynomial as well. Then, by the DH assumption, we have that $\Pr[\mathcal{E}'(h_{\mathcal{A}}, h_{\mathcal{B}}) = g^{x_{\mathcal{A}} x_{\mathcal{B}}}]$ is negligible, hence that $\Pr[L]$ is negligible.

EXERCISE 2.5 Extend the above analysis to the case that the *a priori* probabilities are given by $\Pr[f(u) = 0] = p_0$ and $\Pr[f(u) = 1] = p_1 = 1 - p_0$.

2.1.4 ASIDE: ELGAMAL ENCRYPTION

Recall that the ElGamal cryptosystem is defined as follows, given a security parameter k.

Key generation. Pick a group $\langle g \rangle$ at random among all groups of "size" k. Let n denote the order of g. Next, pick $x \in_R \mathbb{Z}_n^*$. The private key is x, the public key is $h = g^x$.

Encryption. Given a plaintext $M \in \langle g \rangle$, pick $u \in_R \mathbb{Z}_n$. The ciphertext for a public key h is the pair $(g^u, h^u M)$.

Decryption. Given a ciphertext (A, B), the plaintext is recovered as $M = B/A^x$, using private key x.

Note that key generation and encryption are randomized, while decryption is deterministic. In practice, the group $\langle g \rangle$ may be shared between many users.

The ElGamal cryptosystem is *semantically secure* under the DDH assumption. That is, the ciphertext does not leak any (partial) information on the plaintext.

EXERCISE 2.6 Show how to break the ElGamal cryptosystem for $\langle g \rangle = \mathbb{Z}_p^*$, with $p = 2p' + 1$, p, p' both prime. Focus on the case that $M \in \{1, g\}$, and show how to recover M.

A practical variant of the ElGamal cryptosystem is obtained as follows, using a cryptographic hash function:

Key generation. As above.

Encryption. Given a plaintext $M \in \{0,1\}^k$, pick $u \in_R \mathbb{Z}_n$. The ciphertext for a public key h is the pair $(g^u, H(h^u) \oplus M)$.

Decryption. Given a ciphertext (A, B), the plaintext is recovered as $M = H(A^x) \oplus B$, using private key x.

This variant is secure under the DH assumption, in the random oracle model. One may think of $H(A^x)$ as a one-time pad.

2.2 AUTHENTICATED KEY EXCHANGE

2.2.1 MAN-IN-THE-MIDDLE ATTACKS

Under the DDH assumption, the Diffie-Hellman protocol is secure against passive attackers. Against active attackers, however, the protocol is completely insecure. While a passive attacker is restricted to eavesdropping on the communication between $\mathcal{A}$ and $\mathcal{B}$, an active attacker is allowed to manipulate the messages exchanged between $\mathcal{A}$ and $\mathcal{B}$ at its own liking. That is, an attacker may delete, inject, and modify messages.

In the context of key exchange protocols, the most prominent type of active attack is a so-called man-in-the-middle attack. Here, the idea is that when $\mathcal{A}$ and $\mathcal{B}$ engage, say, in an execution of the Diffie-Hellman protocol, the attacker will replace the values $h_{\mathcal{A}}$ and $h_{\mathcal{B}}$ by values $h'_{\mathcal{A}}$ and $h'_{\mathcal{B}}$ of its own choice, respectively. Below, we consider a few examples.

Attack 1. The attacker uses $h'_{\mathcal{A}} = g$ and $h'_{\mathcal{B}} = g$, which results in $K = h_{\mathcal{A}}$ as the "common" key for $\mathcal{A}$ and $K = h_{\mathcal{B}}$ as the "common" key for $\mathcal{B}$. These keys will be known to any passive eavesdropper as well. Note that an event such as $h_{\mathcal{A}} = g$ does not happen in practice, as such a particular event occurs with a negligible probability of $1/n$ only.

Attack 2. The attacker uses $h'_{\mathcal{A}} = g^{x'_{\mathcal{A}}}$ and $h'_{\mathcal{B}} = g^{x'_{\mathcal{B}}}$, with $x'_{\mathcal{A}}, x'_{\mathcal{B}} \in_R \mathbb{Z}_n^*$. This will go completely unnoticed to $\mathcal{A}$ and $\mathcal{B}$ as $h'_{\mathcal{A}}$ and $h'_{\mathcal{B}}$ follow exactly the same distribution as $h_{\mathcal{A}}$ and $h_{\mathcal{B}}$. At the end of the protocol, $\mathcal{A}$ will compute $g^{x_{\mathcal{A}} x'_{\mathcal{B}}}$ as its "common" key while $\mathcal{B}$ will compute $g^{x'_{\mathcal{A}} x_{\mathcal{B}}}$ as its "common" key.

This is the standard man-in-the-middle attack, and it constitutes a complete break of the protocol. The keys used by $\mathcal{A}$ and $\mathcal{B}$ are not even equal to each other, and both are fully known to the attacker.

Attack 3. The attacker uses $h'_{\mathcal{A}} = h_{\mathcal{A}} g^u$ and $h'_{\mathcal{B}} = h_{\mathcal{B}}$, with $u \in_R \mathbb{Z}_n$. Then $\mathcal{A}$ will compute $K_{\mathcal{AB}} = g^{x_{\mathcal{A}} x_{\mathcal{B}}}$, while $\mathcal{B}$ computes $K'_{\mathcal{AB}} = g^{(x_{\mathcal{A}}+u) x_{\mathcal{B}}}$. Clearly, these keys are uncorrelated (hence different from each other, except with negligible probability). However, the attacker does not know anything about the keys $K_{\mathcal{AB}}$ and $K'_{\mathcal{AB}}$, except that $K'_{\mathcal{AB}} = K_{\mathcal{AB}} h_{\mathcal{B}}^u$.

Attack 4. The attacker uses $h'_{\mathcal{A}} = 1/h_{\mathcal{A}}$ and $h'_{\mathcal{B}} = 1/h_{\mathcal{B}}$. Then $\mathcal{A}$ and $\mathcal{B}$ compute as common key $g^{-x_{\mathcal{A}} x_{\mathcal{B}}}$. Then $\mathcal{A}$ and $\mathcal{B}$ agree on the common key and the attacker does not know any information on the shared key. However, the key is different than intended or expected.

For instance, suppose $\langle g \rangle$ is a group of points on an elliptic curve (see Example 1.6). To perform the attack, the attacker only needs to negate the y-coordinate of the points exchanged by $\mathcal{A}$ and $\mathcal{B}$ (assuming the curve equation is of the form $Y^2 = f(X)$). As a result, the y-coordinate of the resulting common key is negated.

EXERCISE 2.7 Consider the two protocols of Figure 2.1 between parties $\mathcal{A}$ and $\mathcal{B}$ connected by an insecure communication channel: Protocol I for sending a plaintext $M \in \langle g \rangle$, $M \neq 1$, securely from party $\mathcal{A}$ to party $\mathcal{B}$, and Protocol II for sending a plaintext $b \in \{0, 1\}$ securely from party $\mathcal{A}$ to party $\mathcal{B}$. The object of both protocols is that the plaintext remains completely hidden from other parties than $\mathcal{A}$ and $\mathcal{B}$, and that the plaintext cannot be modified by other parties than $\mathcal{A}$ or $\mathcal{B}$.

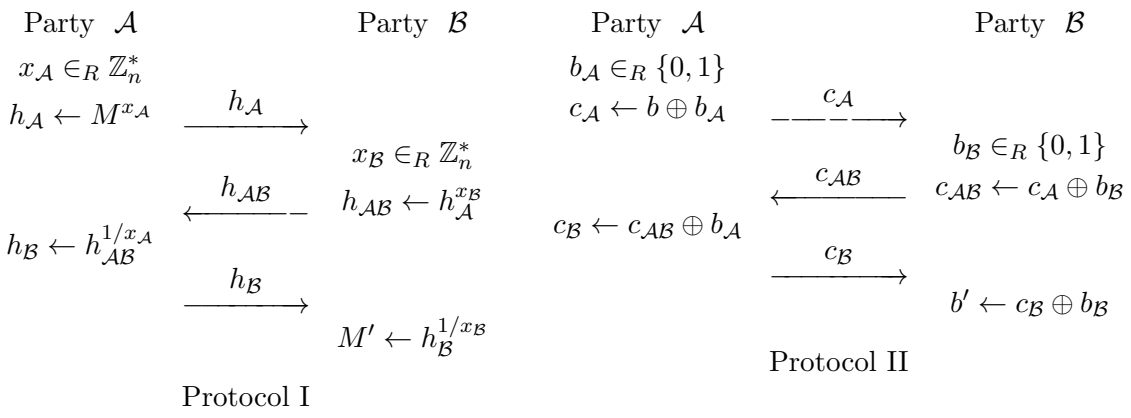


FIGURE 2.1: Three-pass encryption?

First, verify that $M' = M$ and $b' = b$ if $\mathcal{A}$ and $\mathcal{B}$ follow protocols I and II, respectively. Next, determine for protocol I whether it is secure against passive attacks, and whether it is secure against active attacks. If secure, describe the relevant computational assumption (if any); if insecure, show an attack. Then, do the same for protocol II.

2.2.2 A PROTOCOL USING DIGITAL SIGNATURES

The Diffie-Hellman protocol only withstands passive attacks. A first, but general, idea to obtain a key exchange protocol withstanding active attacks is to authenticate the communication between $\mathcal{A}$ and $\mathcal{B}$. For instance, we may assume that $\mathcal{A}$ and $\mathcal{B}$ know each other's public keys in a digital signature scheme.

There are many solutions to this problem, but only few have been proved correct. We will not give a formal security analysis at this point.

The protocol is as follows. Party $\mathcal{A}$ picks $x_{\mathcal{A}} \in \mathbb{Z}_n^*$ uniformly at random, and sends $h_{\mathcal{A}} = g^{x_{\mathcal{A}}}$ along with a signature on $(h_{\mathcal{A}}, \mathcal{B})$ to party $\mathcal{B}$. Similarly, party $\mathcal{B}$ picks $x_{\mathcal{B}} \in \mathbb{Z}_n^*$ uniformly at random, and replies with $h_{\mathcal{B}} = g^{x_{\mathcal{B}}}$ along with a signature on $(h_{\mathcal{A}}, h_{\mathcal{B}}, \mathcal{A})$ to party $\mathcal{A}$. As before, the agreed upon key is $K = g^{x_{\mathcal{A}} x_{\mathcal{B}}}$.

A protocol of this type is secure under the DDH assumption, also assuming that the digital signature scheme is secure.

2.2.3 A PROTOCOL USING PASSWORD-BASED ENCRYPTION

Suppose no digital signature scheme is available. However, suppose parties $\mathcal{A}$ and $\mathcal{B}$ share a password, which is a relatively small secret.

Let $E : \{0,1\}^* \times \langle g \rangle \to \langle g \rangle$ be an encryption algorithm that takes as input a password w and a plaintext M and produces as output a ciphertext $E_w(M)$. Let $D : \{0,1\}^* \times \langle g \rangle \to \langle g \rangle$ be the corresponding decryption algorithm such that $D_w(E_w(M)) = M$ for all passwords w and plaintexts M.

The protocol is as follows. Party $\mathcal{A}$ picks $x_{\mathcal{A}} \in \mathbb{Z}_n^*$ uniformly at random, and sends $E_w(h_{\mathcal{A}})$, where $h_{\mathcal{A}} = g^{x_{\mathcal{A}}}$, to party $\mathcal{B}$. Similarly, party $\mathcal{B}$ picks $x_{\mathcal{B}} \in \mathbb{Z}_n^*$ uniformly at random, and replies with $E_w(h_{\mathcal{B}})$, where $h_{\mathcal{B}} = g^{x_{\mathcal{B}}}$. Party $\mathcal{A}$ decrypts $E_w(h_{\mathcal{B}})$ to recover the value

of $h_{\mathcal{B}} = D_w(E_w(h_{\mathcal{B}}))$. Similarly, party $\mathcal{B}$ recovers the value of $h_{\mathcal{A}}$, using w. As before, the agreed upon key is $K = g^{x_{\mathcal{A}} x_{\mathcal{B}}}$.

An eavesdropper who wants to guess the password sees $E_w(h_{\mathcal{A}})$ and $E_w(h_{\mathcal{B}})$. These values do not give any information on the password w, since $h_{\mathcal{A}}$ and $h_{\mathcal{B}}$ are random and unknown to the eavesdropper.

An active attacker may try a candidate password w' by sending a value $E_{w'}(h_{\mathcal{A}})$ on behalf of $\mathcal{A}$. When the attacker succeeds, it guessed the password correctly. However, the best the attacker can do is trying all passwords one by one. The attacker cannot do an off-line dictionary attack.

2.3 BIBLIOGRAPHIC NOTES

The Diffie-Hellman key exchange protocol is from the seminal paper by Diffie and Hellman [DH76], in which the other fundamental notions of public-key cryptography, namely asymmetric cryptosystems and digital signature schemes, were also introduced. The intimately related ElGamal cryptosystem—sometimes called Diffie-Hellman encryption for that reason—was introduced by ElGamal in [ElG85], together with a secure digital signature scheme based on a discrete log assumption. Originally, these results were formulated for the group $\mathbb{Z}_p^*$ only (cf. Example 1.4), but the generalization to any finite cyclic group is easy.

Many ways have been proposed to extend the basic Diffie-Hellman protocol to an authenticated key exchange protocol. The approach of Section 2.2.2 is rather generic, and leads to provably secure protocols for various settings (see, e.g., Shoup's paper [Sho99], which introduces several models for (authenticated) key exchange protocols and formally analyzes various instances of such protocols). The approach of Section 2.2.3, which allows for password-based authentication rather than strong authentication, is known as EKE (Encrypted Key Exchange) and is due to Bellovin and Merritt [BM92].

Key exchange and similar protocols are probably the most widely studied and applied type of protocol in cryptography. For instance, the book by Boyd and Mathuria [BM03] contains over 150 protocols for authentication and key establishment—which is still far from exhaustive as noted by the authors.

Protocol I in Figure 2.1 is known as Shamir's no-key protocol and also as Shamir's three-pass protocol (cf. [MOV97, Section 12.3]).

CHAPTER 3

Commitment Schemes

The functionality of a commitment scheme is commonly introduced by means of the following analogy. Suppose you need to commit to a certain value, but you do not want to reveal it right away. For example, the committed value is a sealed bid in some auction scheme. One way to do this is to write the value on a piece of paper, put it in a box, and lock the box with a padlock. The locked box is then given to the other party, but you keep the key. At a later time, you present the key to the other party who may then open the box, and check its contents.

An immediate application of commitment schemes is known as "coin flipping by telephone." Two parties, say $\mathcal{A}$ and $\mathcal{B}$, determine a mutually random bit as follows. Party $\mathcal{A}$ commits to a random bit $b_{\mathcal{A}} \in_R \{0,1\}$ by sending a commitment on $b_{\mathcal{A}}$ to party $\mathcal{B}$. Party $\mathcal{B}$ then replies by sending a random bit $b_{\mathcal{B}} \in_R \{0,1\}$ to $\mathcal{A}$. Finally, party $\mathcal{A}$ opens the commitment and sends $b_{\mathcal{A}}$ to $\mathcal{B}$. Both parties take $b = b_{\mathcal{A}} \oplus b_{\mathcal{B}}$ as the common random bit.

If at least one of the parties is honest, the resulting bit b is distributed uniformly at random, assuming that $\mathcal{A}$ and $\mathcal{B}$ cannot cheat when revealing their bits. Note that party $\mathcal{B}$ sees the commitment of $\mathcal{A}$ before choosing its bit $b_{\mathcal{B}}$, so no information on bit $b_{\mathcal{A}}$ should leak from the commitment on $b_{\mathcal{A}}$. Similarly, party $\mathcal{A}$ could try to influence the value of the resulting bit b (after seeing the bit $b_{\mathcal{B}}$) by opening the commitment on $b_{\mathcal{A}}$ as a commitment on $1 - b_{\mathcal{A}}$. Clearly, party $\mathcal{A}$ should not be able to "change its mind" in such a way!

Generating mutually random bits is a basic part of many protocols. Commitments are used as an auxiliary tool in many cryptographic applications, such as zero-knowledge proofs and secure multi-party computation.

3.1 DEFINITION

A commitment scheme consists of two protocols, called *commit* and *reveal*, between two parties, usually called the sender and the receiver. In many cases, the protocols commit and reveal can be defined in terms of a single algorithm, requiring no interaction between the sender and receiver at all. Such commitment schemes are called non-interactive.

DEFINITION 3.1 *Let* commit : $\{0,1\}^k \times \{0,1\}^* \to \{0,1\}^*$ *be a deterministic polynomial time algorithm, where k is a security parameter. A (non-interactive)* ***commitment scheme*** *consists of two protocols between a sender and a receiver:*

Commit Phase. *A protocol in which the sender commits to a value $x \in \{0,1\}^*$ by com-*

puting $C = \text{commit}(u, x)$*, where* $u \in_R \{0,1\}^k$*, and sending* C *to the receiver. The receiver stores* C *for later use.*

Reveal Phase. *A protocol in which the sender opens commitment* $C = \text{commit}(u, x)$ *by sending* u *and* x *to the receiver. The receiver computes* $\text{commit}(u, x)$ *and verifies that it is equal to the previously received commitment.*

In the special case that the committed value is a bit, that is, $x \in \{0,1\}$, one speaks of a **bit commitment scheme**. The security requirements for a bit commitment scheme are the following.

The commitment must be **binding**, i.e., for any adversary $\mathcal{E}$, the probability of generating $u, u' \in \{0,1\}^k$ satisfying $\text{commit}(u, 0) = \text{commit}(u', 1)$ should be negligible (as a function of k). Furthermore, the commitment must be **hiding**, i.e., the distributions induced by $\text{commit}(u, 0)$ and $\text{commit}(u, 1)$ (with $u \in_R \{0,1\}^k$) are indistinguishable.

Moreover, one makes the following distinctions. A commitment scheme is called **computationally binding** if the adversary $\mathcal{E}$ is restricted to be a p.p.t. algorithm. If no such restriction is made (in other words, the adversary may be unlimitedly powerful), the scheme is called **information-theoretically binding**. Similarly, if the distributions induced by $\text{commit}(u, 0)$ and $\text{commit}(u, 1)$ are computationally indistinguishable the scheme is called **computationally hiding** and the scheme is called **information-theoretically hiding** if these distributions are statistically (or even perfectly) indistinguishable.

The security properties are easily extended to the case that x is an arbitrary bit string.

Note that the above security requirements only cover attacks by either the sender or the receiver. For example, suppose party $\mathcal{A}$ acts as the sender and party $\mathcal{B}$ acts as the receiver, and $\mathcal{A}$ sends a commitment C to $\mathcal{B}$. Then there is no guarantee that $\mathcal{B}$ will notice if an attacker replaces C by a commitment $C' = \text{commit}(u', x')$ during the commit protocol, and replaces u, x by u', x' during the reveal protocol. Such attacks may be stopped by using an authenticated channel between $\mathcal{A}$ and $\mathcal{B}$.

3.2 EXAMPLES

3.2.1 USING A CRYPTOGRAPHIC HASH FUNCTION

Given a cryptographic hash function H, one obtains a bit commitment scheme simply by setting

$$\text{commit}_0(u, x) = H(u, x),$$

where $x \in \{0,1\}$ and $u \in_R \{0,1\}^k$.

Collision-resistance of H guarantees that the committer cannot (efficiently) prepare u, x and $u', 1 - x$ with $H(u, x) = H(u', 1 - x)$. Hence, the scheme is binding.

Preimage resistance of H is necessary but not sufficient to guarantee that the value of x remains hidden (as well as the value of u). Rather, one needs to assume partial preimage resistance of H, as briefly discussed at the end of Section 1.3.4. In the random oracle model, the scheme is hiding as long as guessing a bit string of $k + 1$ bits is infeasible.

Thus, we conclude that the scheme is computationally binding and computationally hiding. Can we do better?

3.2.2 USING A DISCRETE LOG SETTING

Let $\langle g \rangle$ be a group of order n, where n is a large prime. Let $h \in_R \langle g \rangle \setminus \{1\}$ denote a random group element (such that $\log_g h$ is not known to any party).

We define the following bit commitment scheme (known as "Pedersen's commitment scheme"):

$$\text{commit}_1(u, x) = g^u h^x,$$

where $u \in_R \mathbb{Z}_n$. This scheme is computationally binding (under the DL assumption), which can be seen as follows. Suppose it is computationally feasible to compute $u, u' \in \mathbb{Z}_n$ such that $\text{commit}_1(u, x) = \text{commit}_1(u', 1 - x)$. That means that $g^u h^x = g^{u'} h^{1-x}$, hence that $\log_g h = (u - u')/(1 - 2x)$. Since u, u', and x are all known, this means that the discrete log of h with respect to g would be computed.

On the other hand, the scheme is information-theoretically hiding, since the distribution of $g^u h^x$ is statistically independent of the value of x (and hence g^u and $g^u h$ are perfectly indistinguishable).

As a complementary bit commitment scheme, one may use the following ElGamal-like scheme:

$$\text{commit}_2(u, x) = (g^u, h^{u+x}),$$

where $u \in_R \mathbb{Z}_n$. This scheme is information-theoretically binding, since it is easily seen that there cannot even exist $u, u' \in \mathbb{Z}_n$ such that $\text{commit}_2(u, x) = \text{commit}_2(u', 1-x)$, for $x \in \{0, 1\}$.

On the other hand, this scheme is computationally hiding, since x can be computed as follows. Assuming that the DL problem is feasible, one may compute x from a given commitment $\text{commit}_2(u, x) = (A, B)$, using the formula $x = \log_h B - \log_g A$. Note, however, that the DL assumption is not sufficient to guarantee that the scheme is hiding. We need to use the DDH assumption to ensure the hiding property, as solving for x given $\text{commit}_2(u, x)$ is equivalent to solving the DDH problem.

Both commitment schemes remain secure when used for $x \in \mathbb{Z}_n$ instead of $x \in \{0, 1\}$.

EXERCISE 3.2 Analyze the security properties of the schemes commit_1 and commit_2 for the case that $x \in \mathbb{Z}_n$.

EXERCISE 3.3 What happens if the receiver knows $\log_g h$ in scheme commit_1? Same question for scheme commit_2.

EXERCISE 3.4 Discuss the security of the following commitment scheme for values $x \in \langle g \rangle$:

$$\text{commit}(u, x) = g^u x,$$

where $u \in_R \mathbb{Z}_n$. Is it binding? Is it hiding?

3.2.3 IMPOSSIBILITY RESULT

A natural question is whether there exists a commitment scheme which is both information-theoretically binding and information-theoretically hiding. The following informal argument shows that such a scheme cannot exist.

Consider any bit commitment scheme which is information-theoretically binding. For such a scheme there cannot exist any u, u' such that $\text{commit}(u, 0) = \text{commit}(u', 1)$, because then the (unlimitedly powerful) sender would be able to compute both u and u', and open the

commitment at its liking. However, if the sender commits to 0, say, using $C = \text{commit}(u, 0)$ for some u, the (unlimitedly powerful) receiver will notice, by exhausting the finite set of possibilities, that there does not exist any u' with $C = \text{commit}(u', 1)$, hence the receiver knows that the committed bit must be 0.

3.3 HOMOMORPHIC COMMITMENTS

The basic security requirement for a commitment scheme are that it must be binding and hiding. Another relevant property of a commitment scheme is that it may be homomorphic. For the moment we will introduce the homomorphic property by means of an example. Consider Pedersen's commitment scheme, given by $\text{commit}_1(u, x) = g^u h^x$, where $u \in_R \mathbb{Z}_n$ and $x \in \mathbb{Z}_n$. This scheme is *additively* homomorphic in the sense that:

$$\text{commit}_1(u, x)\,\text{commit}_1(u', x') = \text{commit}_1(u + u', x + x'),$$

where the multiplication on the left-hand side is in the group $\langle g \rangle$ and the additions on the right-hand side are in $\mathbb{Z}_n$. So, the product of two commitments is a commitment to the *sum* of the committed values.

Homomorphic properties turn out to be very useful, e.g., for achieving secure multiparty computation, as we will see later on. As a concrete example, homomorphic commitments can be used as a building block for secure election schemes: very roughly, during the voting stage, voters put their votes into homomorphic commitments, and during the tallying stage, the votes are counted in a verifiable manner by taking the product of all commitments.

EXERCISE 3.5 Assume the setting of Exercise 1.35. The Quadratic Residuosity (QR) assumption states that the QR problem is hard.

Let $y \in J_m$ denote a quadratic non-residue modulo m (e.g., $y = -1$ is a quadratic non-residue modulo m when m is a Blum integer, that is, $m = pq$ with $p \equiv q \equiv 3 \pmod 4$). Consider the following bit commitment scheme:

$$\text{commit}(u, x) = y^x u^2 \bmod m,$$

where $u \in_R \mathbb{Z}_m^*$ and $x \in \{0, 1\}$. In what sense is the scheme binding? In what sense is the scheme hiding? In what sense is the scheme homomorphic?

3.4 BIBLIOGRAPHIC NOTES

The concept of a commitment scheme emerged in the early 1980s, most notably in the paper by Blum [Blu82] on "coin flipping by telephone," which was also used as a motivating example in the beginning of this chapter. Commitment schemes have been an important cryptographic primitive ever since.

Pedersen's commitment scheme first appeared in [Ped92], where it is presented as part of a non-interactive verifiable secret sharing scheme (see Section 6.2.2).

The impossibility result for a both information theoretically binding and hiding commitment scheme is folklore.

The commitments of Exercise 3.5 correspond to ciphertexts in the Goldwasser-Micali public-key cryptosystem, which was actually the first *probabilistic* cryptosystem (see [GM84], where also the notion of semantic security is introduced and proved to hold for this cryptosystem under the QR assumption).

CHAPTER 4

Identification Protocols

We consider identification protocols between two parties, where one party, called the **verifier**, needs to get convinced that the other party, called the **prover**, is as claimed. A typical example is when a user wants to gain access to a computer account (secure login), or when someone needs to enter a secured room.

In general, identification protocols may be based on one or more of the following factors.

- *What you are.* Biometrics, such as fingerprints, iris scans, etc.
- *What you have.* Smart cards, SIM cards, or similar hardware tokens.
- *What you know.* Passwords, PIN codes, secret keys.

In this chapter, we focus on purely cryptographic identification protocols, in which a successful prover only needs to know some secret key. There are many, many cryptographic constructions of identification protocols. A general goal is to minimize the computational effort for the prover and/or the verifier. The security ranges from rather weak for password-based protocols to strong for zero-knowledge protocols and witness-hiding protocols.

We note that the problem addressed by identification protocols is related to message authentication (e.g., by means of digital signatures) and also to authenticated key exchange.

Compared to message authentication, an important difference is that there is some notion of *freshness* to be fulfilled. On the other hand, compared to digital signatures, there is no such thing as non-repudiation: it is not required that the verifier is able to convince an outsider at a later point in time that a prover indeed successfully identified itself to the verifier. In other words, it is not a requirement that the prover gets an alibi from engaging in an identification protocol.

Compared to authenticated key exchange, the problem is easier as there is no requirement for actually establishing a secure (session) key.

4.1 DEFINITIONS

An identification protocol is actually part of an identification scheme. An **identification scheme** consists of two protocols, called *registration* and *identification*, between two parties, called the prover and the verifier.

In a basic symmetric identification scheme, registration will end with both parties sharing a secret key, which both of them need to store securely. In a basic asymmetric identification

scheme, registration will end with both parties sharing a public key, for which only the prover knows the private key. (In more advanced schemes, also the verifier may have a private key.) A major advantage of asymmetric schemes is that the prover may use its public key with several, possibly many, verifiers.

We consider attacks on the identification protocol only. Hence, we assume that the registration protocol is performed in a secure environment. Furthermore, we consider only cryptographic attacks.[1]

The basic security requirement for an identification protocol is that it stops **impersonation attacks**, that is, it should be impossible for an attacker to successfully identify itself as another party. We distinguish several passive and active impersonation attacks.

The main type of passive impersonation attack is *eavesdropping* on communication between a prover and a verifier in legal executions of the identification protocol. Another type of passive attack is a *key-only attack* for asymmetric schemes, in which the attacker tries to find the private key from the public key. However, we will not be concerned with key-only attacks.

A simple form of active impersonation attack is a *guessing attack*, in which the attacker poses as the prover and hopes to make the right guesses, without knowing the prover's secret key or private key. The success rate of a guessing attack may be increased considerably by combining it with a *cheating verifier* attack, in which the attacker poses as a verifier and hopes to extract some useful information from the prover by deviating from the protocol.

Finally, the attacker may apply a *man-in-the-middle attack*: an honest prover $\mathcal{P}$ thinks it runs the identification protocol with verifier $\mathcal{V}^*$ but actually $\mathcal{V}^*$ relays all messages to a verifier $\mathcal{V}$ who thinks it runs the protocol with $\mathcal{P}$. For example, one may identify itself to open a certain door X but the attacker will have you open another door Y (while you get the message that there was some malfunctioning at door X).

The man-in-the-middle attack is reminiscent of the so-called grandmaster chess attack, in which an amateur chess player tries to increase his or her rating by playing correspondence chess with two grandmasters at the same time. The amateur chess player will engage in a chess game with both grandmasters, playing white in one game and black in the other one. Once started, the amateur simply copies all moves from one grandmaster to the other one. As a result, the amateur will either win one game and lose the other one, or play two draws. In any event, the amateur's rating will increase considerably.

We will be concerned mostly with cheating verifier attacks.

4.2 PASSWORD-BASED SCHEMES

The conventional way to login to a computer is to provide a user-id and a password. Upon registration it is ensured that the prover gets a unique user-id. The prover is also allowed to pick a password. During identification, the prover sends the user-id and password to the verifier.

A password scheme is a symmetric identification scheme, supposed to withstand guessing attacks. One may think of the password as a random bit string in $\{0,1\}^k$. If the password is human-memorizable, the security parameter k is usually rather small, say $k \leq 20$. (We will

[1] Non-cryptographic attacks in which one breaks into the prover's or verifier's computer to steal or modify a key are not considered, as such attacks cannot be stopped by purely cryptographic means. Note that for an asymmetric scheme it is indeed important that the prover's public key, as stored by the verifier, is authentic.

not discuss dictionary attacks and related issues.) Clearly, it is possible to withstand guessing attacks by taking $k = 80$, but then the password will be harder to memorize.

A password scheme does not withstand eavesdropping attacks at all. Once the password is intercepted, the scheme is broken.

4.3 ONE-WAY HASH CHAINS

Lamport's identification scheme provides a relatively easy way to stop eavesdropping attacks by using so-called (one-way) hash chains. A hash chain of length ℓ is a sequence of values x_i, $0 \leq i \leq \ell$, satisfying $x_{i+1} = H(x_i)$, for $0 \leq i < \ell$, where $H : \{0,1\}^* \to \{0,1\}^k$ is a cryptographic hash function.

For registration, the prover picks $x_0 \in_R \{0,1\}^k$, computes $x_\ell = H^\ell(x_0)$, and sends x_ℓ to the verifier. Both the prover and the verifier keep a counter i, initially $i = 0$. The prover stores x_0 for later use. The verifier keeps a variable v, which is initially set to x_ℓ.

For identification, the prover increments counter i, computes $x_{\ell-i} = H^{\ell-i}(x_0)$ and sends this value to the verifier. The verifier tests whether $H(x_{\ell-i}) = v$. If so, identification is successful and the verifier increments i and sets $v = x_{\ell-i}$; otherwise, the verifier discards the identification attempt.

Lamport's identification scheme thus requires the prover and the verifier to remain "in sync" (cf. counter i), but unlike a completely symmetric scheme, the verifier does not need to store a secret key.

A key-only attack is infeasible (cf. the random oracle model, Section 1.3.4). The scheme withstands eavesdropping attacks as interception of a value $x_{\ell-i}$ does not help the attacker in succeeding in any of the subsequent runs of the identification protocol.

We do not consider active attacks for this scheme.

REMARK 4.1 *Hash chains have been considered for applications in which the length of the hash chain is very large. For such "ultra long" hash chains one takes, for instance,* $\ell = 2^{32}$.

For ultra-long hash chains, the naive implementation in which $x_{\ell-i}$ *is computed from* x_0 *in each run of the identification protocol is not acceptable. Similarly, it is not feasible to store the entire sequence* x_i, *for* $i = 0, \ldots, \ell - 1$.

However, using so-called pebbling algorithms *it is possible to achieve the following performance. The space complexity is measured as the number of* k*-bit strings stored, whereas the time complexity is measured as the number of applications of the hash function* H. *The prover uses* $O(\log \ell)$ *storage, and the verifier uses* $O(1)$ *storage. For registration the prover spends* $O(\ell)$ *time, and the verifier needs* $O(1)$ *time. For each run of the identification protocol, the prover needs* $O(\log \ell)$ *time, whereas the verifier needs* $O(1)$ *time.*

4.4 BASIC CHALLENGE-RESPONSE PROTOCOLS

We consider four basic challenge-response protocols. In each of these identification protocols, the verifier starts by sending a random challenge, which the prover answers by sending a response, which is then checked by the verifier. The schemes are summarized in Figure 4.1.

We will consider eavesdropping attacks and cheating verifier attacks for each of the schemes.

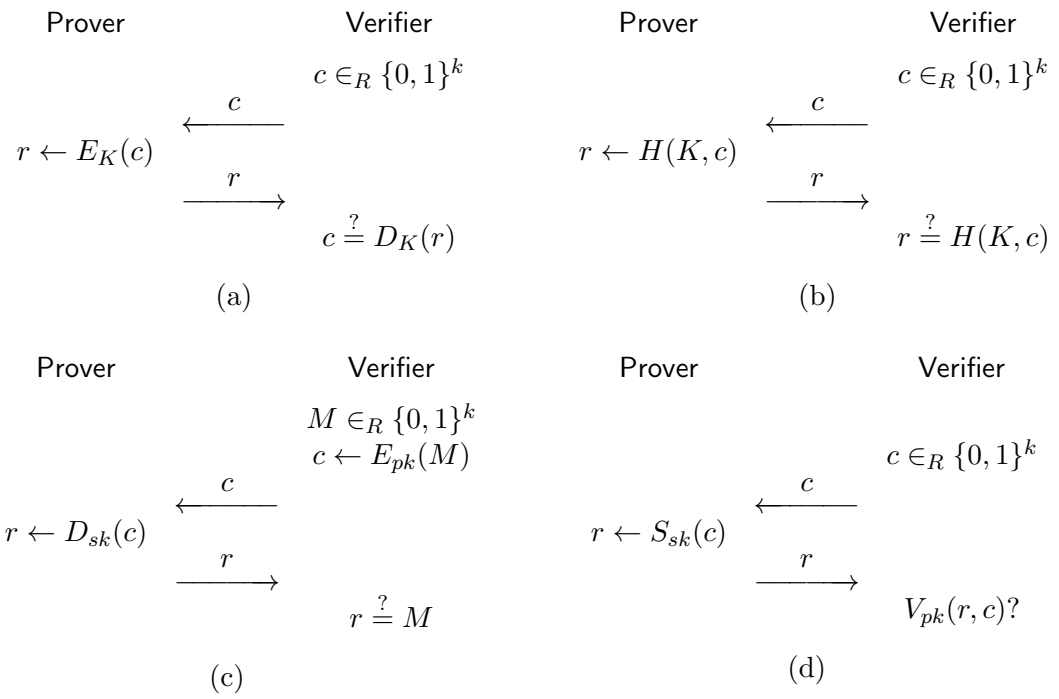


FIGURE 4.1: Four basic challenge-response schemes

4.4.1 USING SYMMETRIC ENCRYPTION

Assume that prover and verifier share a symmetric key $K \in_R \{0,1\}^k$. Let E_K denote an encryption algorithm using key K, and let D_K denote the corresponding decryption algorithm. Assume for simplicity that $E_K, D_K : \{0,1\}^k \to \{0,1\}^k$.

See Figure 4.1(a). The identification protocol starts with the verifier sending a challenge $c \in_R \{0,1\}^k$, for which the prover is supposed to return the response $r = E_K(c)$. The verifier checks that indeed $D_K(r) = c$.

To withstand eavesdropping attacks an encryption scheme withstanding known-plaintext attacks must be used. To withstand cheating verifier attacks, the encryption scheme must withstand adaptive chosen-plaintext attacks.

EXERCISE 4.2 Consider the alternative protocol in which the verifier challenges the prover with $c = E_K(M)$, where $M \in_R \{0,1\}^k$, and for which the prover is supposed to produce response $r = M$. Discuss eavesdropping attacks and cheating verifier attacks for this protocol.

4.4.2 USING SYMMETRIC AUTHENTICATION

Assume that prover and verifier share a symmetric key $K \in_R \{0,1\}^k$. Let $H : \{0,1\}^* \to \{0,1\}^k$ denote a cryptographic hash function.

See Figure 4.1(b). The identification protocol starts with the verifier sending a challenge $c \in_R \{0,1\}^k$, for which the prover is supposed to return the response $r = H(K,c)$. The verifier checks that indeed $r = H(K,c)$.

In the random oracle model, the scheme withstands both eavesdropping and cheating verifier attacks.

4.4.3 USING ASYMMETRIC ENCRYPTION

Assume that prover and verifier share a public key pk for which the prover knows the private key sk. Let E_{pk} denote an encryption algorithm using key pk, and let D_{sk} denote the corresponding decryption algorithm using key sk.

See Figure 4.1(c). The verifier challenges the prover with $c = E_{pk}(M)$ with $M \in_R \{0,1\}^k$, for which the prover is supposed to produce response $r = D_{sk}(c)$. The verifier checks that indeed $r = M$ holds.

To withstand eavesdropping attacks the encryption scheme must be semantically secure. To withstand cheating verifier attacks, the encryption scheme must withstand adaptive chosen-ciphertext attacks.

4.4.4 USING ASYMMETRIC AUTHENTICATION

Assume that prover and verifier share a public key pk for which the prover knows the private key sk. Let S_{sk} denote a signing algorithm using key sk, and let V_{pk} denote the corresponding verification algorithm using key pk.

See Figure 4.1(d). The identification protocol starts with the verifier sending a challenge $c \in_R \{0,1\}^k$, for which the prover is supposed to return the response $r = S_{sk}(c)$. The verifier checks that indeed $V_{pk}(c,r)$ holds, that is, whether r is indeed a signature on message c under public key pk.

To withstand eavesdropping attacks the digital signature scheme must withstand known-message attacks. To withstand cheating verifier attacks, the digital signature scheme must withstand adaptive chosen-message attacks.

4.5 ZERO-KNOWLEDGE IDENTIFICATION PROTOCOLS

The schemes of the previous section are secure when used with sufficiently strong encryption or authentication schemes. The cost of a digital signature scheme withstanding adaptive chosen-message attack is quite high, though. In addition, the work for the prover for computing the response may be costly, in particular considering the work the prover needs to do strictly *after* receiving the challenge.

In this section we will consider zero-knowledge identification protocols, which will have the property that no matter what a cheating verifier does, it will not extract any *useful* information from the (honest) prover. More precisely, the term "zero-knowledge" refers to the fact that whatever information the cheating verifier learns from (interacting with) the prover, that information could have been generated by the cheating verifier on its own—without interacting with the prover at all. In other words, it is possible to show that the messages sent by the prover can be (efficiently) *simulated* without actually involving the prover. An honest verifier, however, will be convinced that the prover knows the private key, as required.

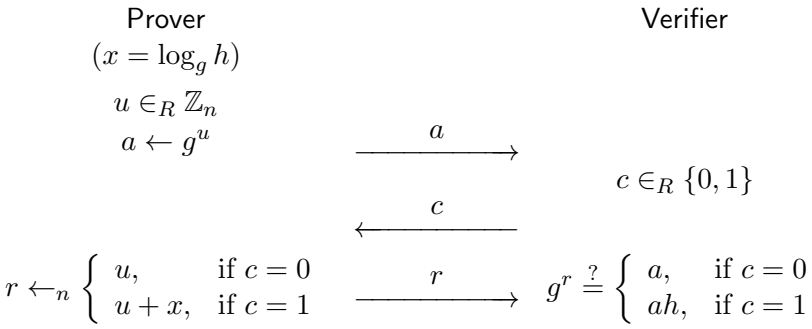


FIGURE 4.2: Schnorr's zero-knowledge protocol

4.5.1 SCHNORR ZERO-KNOWLEDGE PROTOCOL

As a first example of a zero-knowledge protocol we consider Schnorr's protocol for proving knowledge of a discrete logarithm. We will describe the protocol somewhat informally at this point, as Schnorr's protocol and similar protocols will be revisited extensively in the next chapter.

Let $\langle g \rangle$ be a group of order n, where n is a large prime. Let $x \in_R \mathbb{Z}_n$ be the prover's private key, and let $h = g^x$ be the prover's public key. The verifier gets the public key h during the registration protocol. One iteration of Schnorr's identification protocol is given in Figure 4.2.[2] In total, k iterations are executed between the prover and the verifier, one after the other, where k is a security parameter. The three-flow structure of (one iteration of) Schnorr's protocol is typical of many zero-knowledge protocols; we will refer to the first message a as the *announcement*,[3] the second message c is called the *challenge*, and the third message r is called the *response*.

We first discuss why Schnorr's protocol convinces the verifier that the prover indeed knows $x = \log_g h$. This property is called the **soundness** property. If the prover does not know x, the best the prover can do is prepare announcement a such that it knows response r either in the case $c = 0$ or in the case $c = 1$. To prepare for answering challenge $c = 0$, a cheating prover sets $a = g^u$, and sends $r = u$ as response. And, to prepare for answering challenge $c = 1$, a cheating prover sets $a = g^u/h$, and sends the response $r = u$; then the verification $g^r = ah$ will hold.

The point is that the prover cannot prepare for answering both cases $c = 0$ and $c = 1$, without knowing the private key x. This follows from the following observation. Suppose that after sending announcement a, a prover is able to respond to both challenges $c = 0$ and $c = 1$ correctly. That is, the prover is able to produce two responses r_0 and r_1, which are correct for challenges $c = 0$ and $c = 1$, respectively. Then it follows that a, r_0, and r_1 satisfy

$$g^{r_0} = a, \quad g^{r_1} = ah,$$

which implies that

$$h = g^{r_1 - r_0}.$$

[2] We use $x \leftarrow_n E$ to indicate that x is assigned the value of expression E modulo n.

[3] Technically, announcement a is a *commitment to a nonce* u. The nonce u is a secret random value generated for ephemeral (short-term, one-time) use—in contrast with the private key x, which is a secret random value for persistent (long-term, repeated) use.

But this means that the prover actually knows x, since $x = r_1 - r_0 \bmod n$ holds!

Consequently, at each iteration of Schnorr's protocol a cheating prover "survives" with probability at most 50% essentially. Thus after k iterations, a cheating prover succeeds with probability at most 2^{-k} essentially.[4]

Now, we discuss why Schnorr's protocol is **zero-knowledge**. A cheating verifier may engage many times in the identification protocol, obtaining a conversation $(a; c; r)$ for each run of the protocol. Here, "many times" means at most $O(k^\gamma)$ times for some constant $\gamma \in \mathbb{N}$ (polynomially bounded in the security parameter k). The cheating verifier thus obtains many conversations $(a; c; r)$. However, it turns out that the verifier may generate these conversations completely on its own, without interacting with the prover at all: the verifier may generate *simulated* conversations $(a; c; r)$ that follow exactly the same distribution as the conversations $(a; c; r)$ that occur in executions of the identification protocol with a real prover.

We first consider the zero-knowledge property for the case of an honest verifier $\mathcal{V}$, that is, the verifier picks c uniformly at random in $\{0, 1\}$ as prescribed by the protocol. Below, we present two p.p.t. algorithms, one for generating real conversations (following the protocol), and one for generating simulated conversations (deviating from the protocol).

Real conversations
Input: private key x
Output: conversation $(a; c; r)$

1. $u \in_R \mathbb{Z}_n$
2. $a \leftarrow g^u$
3. $c \in_R \{0, 1\}$
4. $r \leftarrow_n u + cx$
5. output $(a; c; r)$

Simulated conversations
Input: public key h
Output: conversation $(a; c; r)$

1. $c \in_R \{0, 1\}$
2. $r \in_R \mathbb{Z}_n$
3. $a \leftarrow g^r h^{-c}$
4. output $(a; c; r)$

Both algorithms generate accepting conversations $(a; c; r)$ uniformly at random, that is, $\Pr[(a; c; r) = (A; C; R)] = \frac{1}{2n}$ for any triple $(A; C; R) \in \langle g \rangle \times \{0, 1\} \times \mathbb{Z}_n$ satisfying $g^R = Ah^C$. The crux is that the real conversations are generated given access to the private key x, whereas the simulated conversations are generated given access to the public key h only.

REMARK 4.3 *The fact that an identification protocol is zero-knowledge against an* honest *verifier essentially reduces any* passive *impersonation attack to a key-only attack (see Section 4.1). In particular, eavesdropping conversations between honest provers and verifiers does not yield anything about a prover's private key beyond what can be deduced from the corresponding public key already.*

Next, we consider the zero-knowledge property for the general case of any p.p.t. cheating verifier $\mathcal{V}^*$. We will use probabilistic Turing machine $\mathcal{V}^*$ as a *rewindable black-box*, which means (i) that we access $\mathcal{V}^*$ in a black-box manner only, restricted to exchanging messages with $\mathcal{V}^*$ through its input and output tapes, and (ii) that we can rewind $\mathcal{V}^*$ to any prior configuration. Recall from Section 1.2.4 that the configuration of a probabilistic Turing machine is determined by the state of its finite control part, the contents of its tapes (incl. the random tape) as well as the positions of its tape heads. By rewinding $\mathcal{V}^*$ we can test it on several input values until a desired output value is obtained.

[4] It is possible to show in a rigorous way that any p.p.t. cheating prover only succeeds with probability 2^{-k} plus a further negligible term representing the success probability of finding $x = \log_g h$ directly.

Real conversations
Input: private key x
Output: conversation $(a; c; r)$

1. $u \in_R \mathbb{Z}_n$
2. $a \leftarrow g^u$
3. send a to $\mathcal{V}^*$
4. receive $c \in \{0, 1\}$ from $\mathcal{V}^*$
5. $r \leftarrow_n u + cx$
6. send r to $\mathcal{V}^*$
7. output $(a; c; r)$

Simulated conversations
Input: public key h
Output: conversation $(a; c; r)$

1. $c \in_R \{0, 1\}$
2. $r \in_R \mathbb{Z}_n$
3. $a \leftarrow g^r h^{-c}$
4. send a to $\mathcal{V}^*$
5. receive $c' \in \{0, 1\}$ from $\mathcal{V}^*$
6. if $c \neq c'$ rewind $\mathcal{V}^*$ to point prior to receiving a and go to step 1
7. send r to $\mathcal{V}^*$
8. output $(a; c; r)$

At step 6 of the simulation, the probability that $c = c'$ is exactly $1/2$, since $c \in_R \{0, 1\}$. Hence, on average two iterations are required to generate a simulated conversation $(a; c; r)$.

The conclusion is that no matter what algorithm (or "strategy") a cheating verifier $\mathcal{V}^*$ follows in trying to extract useful information from the prover, the same algorithm can be used to generate identically distributed conversations without needing the cooperation of the prover. Whereas the real conversations are generated using the private key x as input, the simulated conversations are generated using only the public key h as input.

4.5.2 SCHNORR PROTOCOL

The protocol of Figure 4.2 is a zero-knowledge protocol. However, as we have argued informally, the probability that a cheating prover succeeds is 50%. By using k sequential iterations of this protocol the zero-knowledge property is preserved, but the probability that a cheating prover succeeds becomes 2^{-k}, which is negligible as a function of k.

The computational complexity of the resulting protocol is rather high, since both the prover and the verifier need to compute $O(k)$ exponentiations in the group $\langle g \rangle$. Therefore, Schnorr also proposed to use the variant given in Figure 4.3 (which is actually known as "Schnorr's protocol"). In this protocol, the verifier picks its challenge from a large range, say $c \in \mathbb{Z}_n$.

The soundness property of Schnorr's protocol can be analyzed similarly as above. Suppose that a prover is able to answer correctly at least two challenges c and c', with $c \neq c'$, after sending announcement a. That is, the prover is able to produce two valid conversations $(a; c; r)$ and $(a; c'; r')$. Then it follows as before that the prover actually knows the discrete $\log x = \log_g h$, since

$$g^r = ah^c, \quad g^{r'} = ah^{c'}$$

implies that

$$h = g^{(r-r')/(c-c')}.$$

Therefore, intuitively, after sending announcement a, the prover can answer at most one challenge correctly, if the prover does not know the private key x. Since there are n possible values for the challenge, the probability of success is basically bounded above by $1/n$, which is negligibly small.

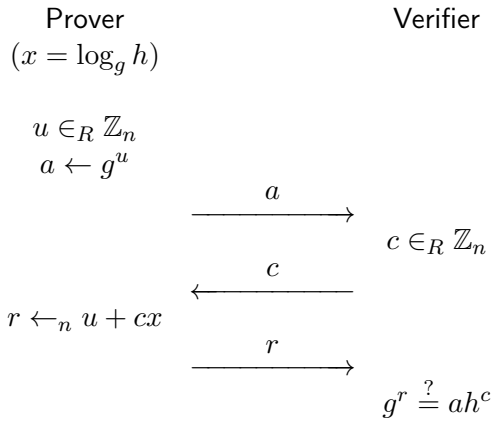


FIGURE 4.3: Schnorr's identification protocol

The zero-knowledge property can also be proved for Schnorr's protocol similarly as above for the case of an honest verifier $\mathcal{V}$. The distributions of the real conversations (generated using private key $x \in \mathbb{Z}_n$) and of the simulated conversations (generated using public key $h \in \langle g \rangle$ only) are, respectively:

$$\{(a; c; r) : u, c \in_R \mathbb{Z}_n; a \leftarrow g^u; r \leftarrow_n u + cx\},$$
$$\{(a; c; r) : c, r \in_R \mathbb{Z}_n; a \leftarrow g^r h^{-c}\}.$$

These distributions are identical, as each valid conversation $(a; c; r)$ (satisfying $c, r \in \mathbb{Z}_n$ and $g^r = ah^c$) occurs with probability $1/n^2$ in both distributions.

In trying to simulate conversations for an arbitrary verifier $\mathcal{V}^*$, we run into a problem. We may use the same algorithm as before, first picking $c, r \in_R \mathbb{Z}_n$, setting $a = g^r h^{-c}$, feeding a to $\mathcal{V}^*$ and then hoping that the c' returned by $\mathcal{V}^*$ matches c. However, $\Pr[c = c'] = 1/n$ which is negligibly small, and it will take n tries on average to find a valid conversation this way. In other words, the running time of the simulator is $O(n)$, which is exponentially large.

In conclusion, Schnorr's protocol is sound and honest-verifier zero-knowledge. Although it cannot be proved zero-knowledge in general, no attacks are known for this protocol, hence it can be used as an identification protocol, if so desired.

Later, we will see how to obtain so-called Schnorr signatures from Schnorr's identification protocol. At this point, we remark that if one could prove that Schnorr's protocol is zero-knowledge then it would follow that Schnorr signatures can be forged. In a way it is therefore good that Schnorr's protocol is not zero-knowledge.

4.5.3 GUILLOU-QUISQUATER PROTOCOL

The identification protocol by Guillou and Quisquater is similar to Schnorr's protocol, except that it is defined in an RSA setting instead of a DL setting.

Let $m = pq$ be an RSA modulus, that is, p and q are large, distinct primes of bit length k, for security parameter k. Let e be a positive integer satisfying $\gcd(e, \phi(m)) = 1$, where $\phi(m) = (p-1)(q-1)$. (See Exercise 1.34.) As an additional requirement for e we have that e is a large prime such that $1/e$ is negligible in security parameter k. For example, e may be a 128-bit prime.

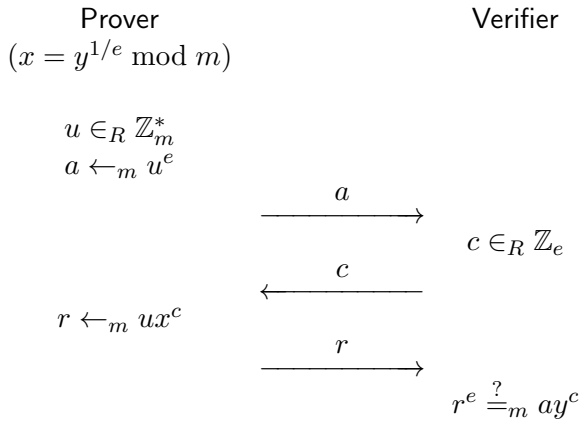


FIGURE 4.4: Guillou-Quisquater's identification protocol

Recall that the RSA problem is to compute $x = y^{1/e} \bmod m$ given $y \in \mathbb{Z}_m^*$, which is assumed to be hard for sufficiently large values of k. For Guillou-Quisquater's protocol (Figure 4.4), the private key of the prover is therefore a number $x \in \mathbb{Z}_m^*$, and the corresponding public key is $y = x^e \bmod m$. One can easily verify that the verifier indeed accepts if the prover follows the protocol, as we have (modulo m):

$$r^e = (ux^c)^e = u^e(x^e)^c = ay^c.$$

The security properties of Guillou-Quisquater's protocol are as follows. The soundness property holds as the success probability of a cheating prover is basically bounded by $1/e$ for the following reason. Suppose that a prover is able to answer two distinct challenges $c, c' \in \mathbb{Z}_e$ correctly, after sending announcement a to the verifier. In other words, suppose a prover is able to produce two accepting conversations $(a; c; r)$ and $(a; c'; r')$, with $c \neq c'$. Then we have (modulo m):

$$r^e = ay^c, \quad r'^e = ay^{c'},$$

which implies

$$(r/r')^e = y^{c-c'}.$$

To isolate y in this equation, we note that $\gcd(e, c - c') = 1$, since e is prime and $c, c' \in \mathbb{Z}_e, c \neq c'$. By the extended Euclidean algorithm integers s, t can thus be computed efficiently satisfying $se + t(c - c') = 1$. Raising both sides of the equation to the power t we get:

$$(r/r')^{te} = y^{t(c-c')} = y^{1-se},$$

hence

$$y = (y^s(r/r')^t)^e.$$

Summarizing, given accepting conversations $(a; c; r)$, $(a; c'; r')$, where $c \neq c'$, the private key x can be computed as $x = y^s(r/r')^t \bmod m$, where s, t satisfy $se + t(c - c') = 1$ (as obtained by the extended Euclidean algorithm).

The protocol is honest-verifier zero-knowledge, since the distributions of the real conversations (generated using private key $x \in \mathbb{Z}_m^*$) and of the simulated conversations (generated

using public key $y \in \mathbb{Z}_m^*$ only) are identical:

$$\{(a; c; r) : u \in_R \mathbb{Z}_m^*; c \in_R \mathbb{Z}_e; a \leftarrow_m u^e; r \leftarrow_m ux^c\},$$
$$\{(a; c; r) : c \in_R \mathbb{Z}_e; r \in_R \mathbb{Z}_m^*; a \leftarrow_m r^e y^{-c}\}.$$

Each valid conversation $(a; c; r)$ (satisfying $c \in \mathbb{Z}_e$, $r \in \mathbb{Z}_m^*$ and $r^e =_m ay^c$) occurs with probability $1/(e\phi(m))$.

REMARK 4.4 *Note that the prover does not need to know the factorization of m. This fact can be exploited by letting several users share the same modulus m. It is even possible to make the identification scheme* identity-based, *which means that the public keys of the users can be computed as a (deterministic) function of their identities. For example, the public key $y_{\mathcal{A}}$ of a user $\mathcal{A}$ whose identity is represented by a string $ID_{\mathcal{A}}$, can be computed as $y_{\mathcal{A}} = H(ID_{\mathcal{A}})$, where H is an appropriate cryptographic hash function. The string $ID_{\mathcal{A}}$ may consist of the user's name and/or email address.*

A trusted third party $\mathcal{T}$ is required to compute the private key of each user. Only $\mathcal{T}$ will know the factorization of the modulus m. Hence, $\mathcal{T}$ is able to compute user $\mathcal{A}$'s private key as $x_{\mathcal{A}} = H(ID_{\mathcal{A}})^{1/e} \bmod m$.

The advantage of an identity-based scheme is that the public keys of users need not be certified anymore by a digital signature from $\mathcal{T}$. A disadvantage is that $\mathcal{T}$ is able to compute the private key of every user. However, this problem may be alleviated by distributing the role of $\mathcal{T}$ between many parties, using threshold cryptography *(see Chapter 6).*

4.6 WITNESS HIDING IDENTIFICATION PROTOCOLS

Schnorr's identification protocol is quite efficient, but it can be proved zero-knowledge only for an honest verifier. By using k iterations of Schnorr's protocol (with binary challenges), the resulting protocol is zero-knowledge for arbitrary verifiers, but, clearly, the computational complexity of the protocol also increases by a factor of k.

Witness hiding identification protocols strike a nice balance between security and efficiency. Consider a protocol that satisfies the soundness property as described above. It follows that a prover can only be successful if it actually knows the *complete* private key. So, the only problem we need to care about is that a cheating verifier is not able to learn the complete private key. Therefore, an identification protocol is called **witness hiding** if a cheating verifier is not able to obtain the prover's private key by interacting with the prover.

If a protocol is witness hiding, it is not necessarily zero-knowledge (but the converse is always true). A cheating verifier may be able to extract some *partial* information on the private key, but the amount of information it is able to get is not sufficient for successful impersonation of the prover.

4.6.1 OKAMOTO PROTOCOL

As before, let $\langle g \rangle$ be a group of order n, where n is a large prime. In addition, let $g_1, g_2 \in \langle g \rangle$ be given such that $\log_{g_1} g_2$ is not known to anybody. (If g_1, g_2 are picked uniformly at random in $\langle g \rangle$ then $\log_{g_1} g_2$ is not known under the DL assumption.) Let $x_1, x_2 \in_R \mathbb{Z}_n$ be the prover's private key, and let $h = g_1^{x_1} g_2^{x_2}$ be the prover's public key. Okamoto's protocol, see Figure 4.5, may be viewed as a variation of Schnorr's protocol. The computational complexity of Okamoto's protocol and of Schnorr's protocol only differ by a constant factor.

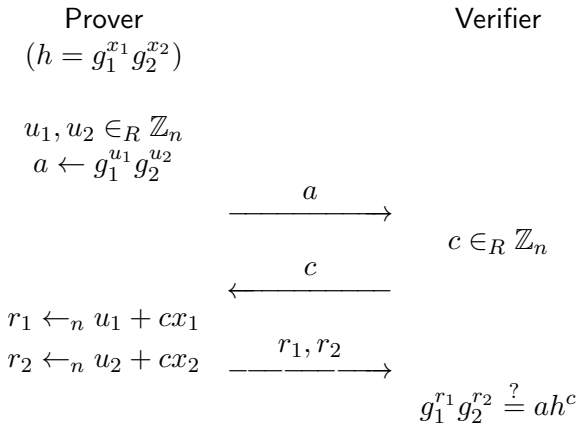


FIGURE 4.5: Okamoto's identification protocol

Also, Okamoto's protocol satisfies the same properties of soundness and honest-verifier zero-knowledgeness as Schnorr's protocol. The important difference is that Okamoto's protocol can be proved to be witness hiding.

We first note the following property. For a given public key h (and given generators g_1, g_2) there are exactly n possible pairs $(x_1, x_2) \in \mathbb{Z}_n \times \mathbb{Z}_n$ satisfying $h = g_1^{x_1} g_2^{x_2}$. Since if one fixes any $x_1 \in \mathbb{Z}_n$, the corresponding x_2 is uniquely defined by $x_2 = \log_{g_2}(h/g_1^{x_1})$. Such a pair (x_1, x_2) is referred to as a **witness**.

The prover's private key is one such witness (x_1, x_2). A crucial property of Okamoto's protocol is that it is **witness indistinguishable**, as the conversations seen by an arbitrary, possibly cheating, verifier are (statistically) independent of the particular witness used by the prover.

To see that Okamoto's protocol is witness indistinguishable we argue as follows. Let $(a; c; r_1, r_2)$ be a conversation between an (honest) prover $\mathcal{P}_{(x_1,x_2)}$ using witness (x_1, x_2) and a possibly cheating verifier $\mathcal{V}^*$, and let u_1, u_2 be the corresponding random numbers used by $\mathcal{P}_{(x_1,x_2)}$. Now, consider another witness (x_1', x_2'). Then there exist unique values $u_1', u_2' \in \mathbb{Z}_n$ that yield the same conversation for a prover $\mathcal{P}_{(x_1',x_2')}$ using witness (x_1', x_2'):

$$\begin{aligned} u_1' &\leftarrow_n u_1 + c(x_1 - x_1'), \\ u_2' &\leftarrow_n u_2 + c(x_2 - x_2'). \end{aligned}$$

Indeed,

$$a' = g_1^{u_1'} g_2^{u_2'} = g_1^{u_1} g_2^{u_2} (g_1^{x_1} g_2^{x_2})^c / (g_1^{x_1'} g_2^{x_2'})^c = a,$$

and (modulo n)

$$\begin{aligned} r_1' &= u_1' + cx_1' = u_1 + c(x_1 - x_1') + cx_1' = r_1, \\ r_2' &= u_2' + cx_2' = u_2 + c(x_2 - x_2') + cx_2' = r_2. \end{aligned}$$

Phrased slightly differently: for each combination of a conversation $(a; c; r_1, r_2)$ between an honest prover $\mathcal{P}$ and a possibly cheating verifier $\mathcal{V}^*$, and a possible witness (x_1', x_2') satisfying $h = g_1^{x_1'} g_2^{x_2'}$, there exist unique u_1', u_2' satisfying $a = g_1^{u_1'} g_2^{u_2'}$, $r_1 =_n u_1' + cx_1'$, and $r_2' =_n u_2' + cx_2'$. This implies that Okamoto's protocol is witness indistinguishable.

Now, suppose that a cheating verifier $\mathcal{V}^*$ is able to find a witness (x_1', x_2') after interacting with a given prover $\mathcal{P}_{(x_1,x_2)}$ polynomially many times. Since Okamoto's protocol is witness indistinguishable, it follows that the witness (x_1', x_2') found by $\mathcal{V}^*$ will be equal to the witness used by $\mathcal{P}_{(x_1,x_2)}$ with probability exactly equal to $1/n$. In other words, with probability close to 1, the two witnesses will be different.

However, now viewing the prover $\mathcal{P}_{(x_1,x_2)}$ and the verifier $\mathcal{V}^*$ as one "big" p.p.t. algorithm $\mathcal{E}$, it follows that $\mathcal{E}$ is able to compute two pairs $(x_1, x_2) \neq (x_1', x_2')$ satisfying

$$h = g_1^{x_1} g_2^{x_2}, \quad h = g_1^{x_1'} g_2^{x_2'}.$$

But this implies that

$$g_2 = g_1^{(x_1' - x_1)/(x_2' - x_2)},$$

hence that $\mathcal{E}$ computed $\log_{g_1} g_2$, which we assumed to be infeasible.

In conclusion, under the DL assumption, Okamoto's protocol is witness hiding, which means that no p.p.t. verifier is able to extract a prover's private key.

REMARK 4.5 *As mentioned above, there are n possible witnesses (x_1, x_2) for a given public key $h = g_1^{x_1} g_2^{x_2}$ in Okamoto's protocol. Interestingly, the protocol remains witness hiding even if the number of possible witnesses used by the prover is limited to just two. For instance, if either $x_1 = 0$ or $x_2 = 0$ (hence either $h = g_1^{x_1}$ or $h = g_2^{x_2}$), the same analysis applies, except that the probability for two different witnesses is now bounded below by $1/2$.*

4.7 BIBLIOGRAPHIC NOTES

Many identification schemes have been proposed throughout the cryptographic literature.

Lamport's identification scheme using hash chains is from [Lam81], building on the idea of iterated hash functions which is attributed to Winternitz (see [Mer87, Mer89], where Merkle refers to a personal communication with Winternitz). The use of ultra long hash chains (see Remark 4.1) is from Jakobsson [Jak02] (see also [CJ02]), building on the pebbling algorithm of [IR01] for efficient key updates in a forward-secure digital signature scheme. See [Sch16] for optimal binary pebbling algorithms, for which the prover stores at most $\log_2 \ell$ hash values and uses at most $\frac{1}{2} \log_2 \ell$ hashes in each run of Lamport's identification protocol. Also, see [Szy04] for extended techniques to generate the successive authentication paths when traversing Merkle trees ("hash trees").

The notion of zero-knowledge was formally introduced by Goldwasser, Micali, and Rackoff in [GMR89], soon after the first such protocol was presented at Eurocrypt 1984 by Fischer, Micali, and Rackoff [FMR96], and is now more broadly referred to as the *simulation paradigm*. A practical scheme for zero-knowledge identification was first presented by Fiat and Shamir [FS87], followed by the slightly improved scheme of Feige, Fiat, and Shamir [FFS88]. Guillou-Quisquater's protocol is from [GQ88], and Schnorr's protocol is from [Sch91]. For more on zero-knowledge proofs, see the next chapter.

The notions of witness indistinguishable and witness hiding protocols are due to Feige and Shamir [FS90], who also gave applications to identification. The elegant and efficient variation of Schnorr's identification protocol is due to Okamoto [Oka93].

CHAPTER 5

Zero-Knowledge Proofs

Zero-knowledge proofs are a general class of protocols between two parties, called the prover and the verifier. By means of a zero-knowledge proof, the prover convinces the verifier of the validity of a given statement without revealing any information beyond the truth of the statement.

In zero-knowledge identification protocols, the statement proved is something like "I know the private key for this public key." But much more involved statements are possible, such as "I know the private key for this public key *or* for that public key, and that in any case the private keys are different." In fact, the theory of zero-knowledge proofs tells us that any NP-statement can be proved efficiently in zero-knowledge (see also Exercise 5.20 and Exercise 5.21).

5.1 Σ-PROTOCOLS

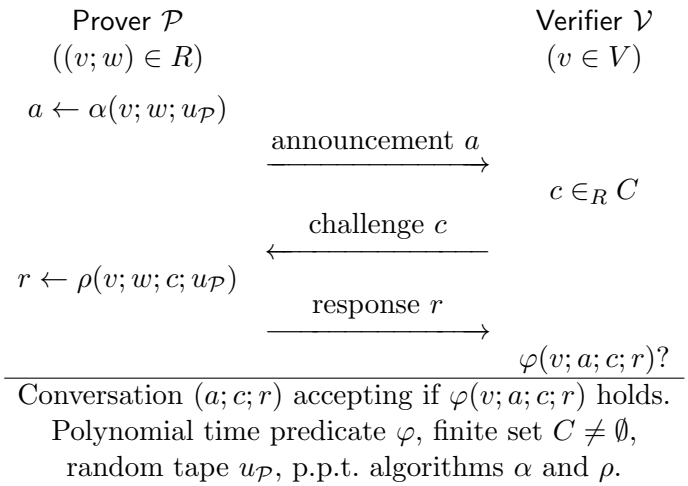
The notion of a Σ-protocol generalizes the identification protocols by Schnorr, Guillou-Quisquater, and Okamoto covered in the previous chapter, as well as many other ones known from the literature (such as Fiat-Shamir's and Feige-Fiat-Shamir's protocols, see Section 4.7). A Σ-protocol is required to be zero-knowledge against an honest verifier only. Despite this limitation, Σ-protocols will turn out to be versatile building blocks, and their use will become apparent later on.

For technical reasons, a Σ-protocol is actually required to be *special* honest-verifier zero-knowledge, which means that the simulator will take a challenge c, say, as an additional input and then produce conversations with the specified challenge c. It is easily checked that the protocols mentioned above are all special honest-verifier zero-knowledge.

Let $R = \{(v; w)\} \subseteq V \times W$ be a binary relation, where $v \in V$ denotes the common input to the prover and the verifier, and $w \in W$ denotes a **witness**, which is the private input to the prover. Let $L_R = \{v \in V : \exists_{w \in W}(v; w) \in R\}$ denote the language corresponding to relation R.[1]

[1]Language L_R will usually be an NP language, hence relation R will be an NP-relation. This means that determining whether $(v; w) \in R$ holds for given $(v; w) \in V \times W$ is in P, that is, in deterministic polynomial time in the size of v one can determine whether $(v; w) \in R$ holds.

More generally, L_R can be an MA language, where MA is the complexity class of Merlin-Arthur languages. MA contains both NP and BPP. To decide if $v \in L_R$ holds, Merlin (the prover) sends w to Arthur (the verifier), where the size of w is bounded by a polynomial in the size of v for all $(v; w) \in R$, and the verifier is a p.p.t. algorithm.



Conversation $(a;c;r)$ accepting if $\varphi(v;a;c;r)$ holds.
Polynomial time predicate φ, finite set $C \neq \emptyset$,
random tape $u_{\mathcal{P}}$, p.p.t. algorithms α and ρ.

FIGURE 5.1: Σ-protocol for relation R

DEFINITION 5.1 *A* ***Σ-protocol for relation R*** *is a protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ of the form given in Figure 5.1 satisfying the following three properties.*

Completeness. *If $\mathcal{P}$ and $\mathcal{V}$ follow the protocol, then $\mathcal{V}$ always accepts.*

Special soundness. *There exists a p.p.t. algorithm E (**extractor**) which given any $v \in V$ and any pair of accepting conversations $(a;c;r)$ and $(a;c';r')$ with $c \neq c'$ always computes a witness w satisfying $(v;w) \in R$.*

Special honest-verifier zero-knowledgeness. *There exists a p.p.t. algorithm S (**simulator**) which given any $v \in L_R$ and any challenge $c \in C$ produces conversations $(a;c;r)$ with the same probability distribution as conversations between honest $\mathcal{P}$ and $\mathcal{V}$ on common input v and challenge c, where $\mathcal{P}$ uses any witness w satisfying $(v;w) \in R$. Furthermore, given any $v \in V \backslash L_R$, simulator S is just required to produce arbitrary accepting conversations $(a;c;r)$, for any given challenge $c \in C$.*

If C consists of a single element, the Σ-protocol is said to be ***trivial****.*

It can be proved rigorously that special soundness implies that a cheating prover succeeds with probability at most $1/n$ essentially, where n denotes the cardinality of the challenge space C. Hence, assuming that n is sufficiently large, a Σ-protocol proves knowledge of a witness w for a public input v.

As can be seen from the following proposition, the special honest-verifier zero-knowledge property is not much stronger than (plain) honest-verifier zero-knowledgeness. For convenience, we let $(C,+)$ be an additive finite group.

PROPOSITION 5.2 *The transformed protocol in Figure 5.2 is a Σ-protocol for relation R, provided that the original protocol as given in Figure 5.1 satisfies completeness, special soundness, and* plain *honest-verifier zero-knowledgeness.*

PROOF Completeness holds because α, ρ, and φ are applied in the same way as in the original protocol with c replaced by $c_{\mathcal{P}} + c_{\mathcal{V}}$.

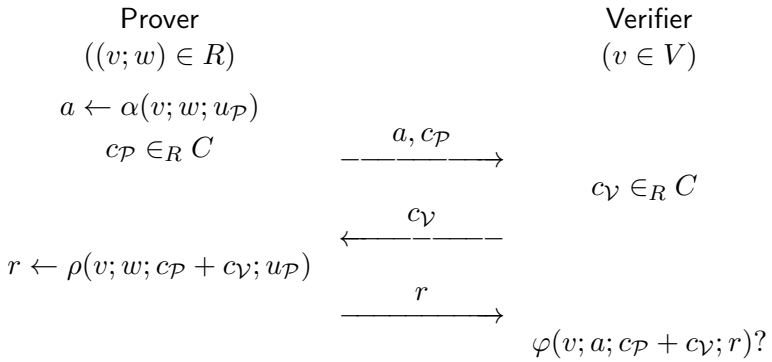


FIGURE 5.2: Transformed Σ-protocol for relation R

For special soundness, consider two accepting conversations $(a, c_{\mathcal{P}}; c_{\mathcal{V}}; r)$ and $(a, c_{\mathcal{P}}; c'_{\mathcal{V}}; r')$ with $c_{\mathcal{V}} \neq c'_{\mathcal{V}}$. Define $c = c_{\mathcal{P}} + c_{\mathcal{V}}$ and $c' = c_{\mathcal{P}} + c'_{\mathcal{V}}$. Then $(a; c; r)$ and $(a; c'; r')$ are two accepting conversations for the original protocol with $c \neq c'$, hence special soundness implies that a witness w can be obtained efficiently.

Finally, for special honest-verifier zero-knowledgeness, let S be a simulator for the original protocol. The simulator for the transformed protocol proceeds as follows. For any given challenge $c_{\mathcal{V}}$, generate an accepting conversation $(a; c; r)$ using simulator S on input v, and put $c_{\mathcal{P}} = c - c_{\mathcal{V}}$. The simulated conversation is defined as $(a, c_{\mathcal{P}}; c_{\mathcal{V}}; r)$, which is accepting by construction. Moreover, if $v \in L_R$, honest-verifier zero-knowledgeness of the original protocol implies that $(a; c; r)$ follows the distribution of conversations of the original protocol. Hence, in particular, $c \in C$ is distributed uniformly at random. Therefore, $c_{\mathcal{P}}$ is also uniformly random, and $(a, c_{\mathcal{P}}; c_{\mathcal{V}}; r)$ exactly follows the distribution of conversations of the transformed protocol. □

The particular way in which Definition 5.1 treats the case $v \in V \backslash L_R$ is chosen in order to streamline OR-composition covered in the next section. For special soundness, extractor E is required to output a witness w for *any* $v \in V$, given any pair of accepting conversations with identical announcements but different challenges—which implies that such pairs of conversations cannot exist (or cannot be found efficiently under some computational assumption) if $v \in V \backslash L_R$. Similarly, for special honest-verifier zero-knowledgeness, simulator S is required to output an accepting conversation for *any* $v \in V$—without any constraints on the distribution of these conversations if $v \in V \backslash L_R$.

As a simple illustration we show that Schnorr's protocol (see Figure 4.3) is a Σ-protocol for proving knowledge of a witness $x \in \mathbb{Z}_n$ satisfying $h = g^x$. Note that from now on we will not explicitly indicate anymore that all calculations involving elements of the finite field $\mathbb{Z}_n$ are actually done modulo n (e.g., "$c \neq c'$" is short for "$c \neq c' \bmod n$," and hence division by $c - c'$ is well-defined modulo n).

PROPOSITION 5.3 *The protocol in Figure 4.3 is a Σ-protocol for relation $\{(h; x) : h = g^x\}$.*

PROOF Completeness clearly holds, as

$$g^r = g^{u+cx} = g^u (g^x)^c = a h^c,$$

using that the prover computes r such that $r = u + cx$.

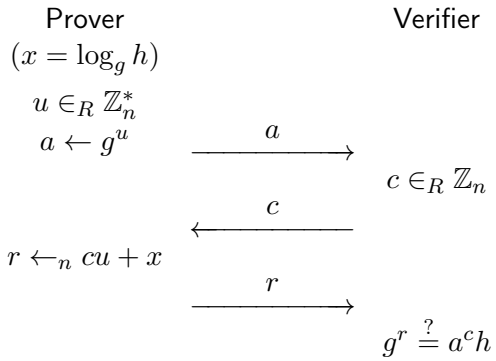


FIGURE 5.3: Insecure variant of Schnorr's protocol

For special soundness, we note that given accepting conversations $(a; c; r)$ and $(a; c'; r')$ with $c \neq c'$, we have:

$$\begin{aligned} & g^r = ah^c, \quad g^{r'} = ah^{c'} \\ \Rightarrow \quad & g^{r-r'} = h^{c-c'} \\ \Leftrightarrow \quad & h = g^{\frac{r-r'}{c-c'}} \end{aligned}$$

Hence, the witness is obtained as $x = (r - r')/(c - c')$.

Finally, to show special honest-verifier zero-knowledgeness, the distributions for the conversations with an honest verifier (following the protocol, using witness x as input) and the simulated conversations (deviating from the protocol, using only common input h) are, respectively:

$$\{(a; c; r) : u \in_R \mathbb{Z}_n; a \leftarrow g^u; r \leftarrow_n u + cx\},$$
$$\{(a; c; r) : r \in_R \mathbb{Z}_n; a \leftarrow g^r h^{-c}\},$$

given an arbitrary challenge c. These distributions are identical (each conversation occurs exactly with probability $1/n$). □

EXERCISE 5.4 To understand why honest-verifier zero-knowledgeness does not imply zero-knowledgeness for arbitrarily cheating verifiers, consider the protocol given in Figure 5.3. Show that the protocol is complete, special sound, and honest-verifier zero-knowledge. Also, show that the protocol is completely insecure against a cheating verifier.

EXERCISE 5.5 The protocol in Figure 5.3 avoids the case $u = 0$, but this is not essential. Show that the protocol remains complete, special sound, and honest-verifier zero-knowledge (and insecure against a cheating verifier) if one uses $u \in_R \mathbb{Z}_n$ instead of $u \in_R \mathbb{Z}_n^*$, by exhibiting a slightly more involved simulation.

REMARK 5.6 *By applying the transformation of Proposition 5.2 to the protocol in Figure 5.3, we obtain a Σ-protocol which is completely* insecure *against a cheating verifier.*

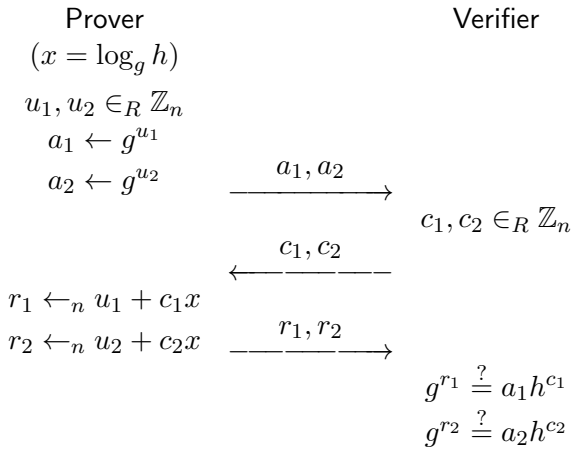


FIGURE 5.4: Parallel composition of Schnorr's protocol

5.2 COMPOSITION OF Σ-PROTOCOLS

By means of examples we introduce five forms of composition of Σ-protocols: parallel composition,[2] AND-composition, EQ-composition, OR-composition, and NEQ-composition. We will use Schnorr's protocol as the basic Σ-protocol for constructing more advanced Σ-protocols.

5.2.1 PARALLEL COMPOSITION

Running two instances of a non-trivial Σ-protocol for relation R in parallel results in a Σ-protocol for R with a larger challenge space.

Figure 5.4 shows the parallel composition of two instances of Schnorr's protocol for proving knowledge of $x = \log_g h$.

PROPOSITION 5.7 *The protocol in Figure 5.4 is a Σ-protocol for relation* $\{(h; x) : h = g^x\}$.

PROOF Completeness follows immediately from the completeness of Schnorr's protocol. More concretely, we have:

$$\begin{aligned} g^{r_1} &= g^{u_1 + c_1 x} = g^{u_1}(g^x)^{c_1} = a_1 h^{c_1}, \\ g^{r_2} &= g^{u_2 + c_2 x} = g^{u_2}(g^x)^{c_2} = a_2 h^{c_2}, \end{aligned}$$

using that the prover computes r_1 and r_2 such that $r_1 = u_1 + c_1 x$ and $r_2 = u_2 + c_2 x$.

To show special soundness, let $(a_1, a_2; c_1, c_2; r_1, r_2)$ and $(a_1, a_2; c_1', c_2'; r_1', r_2')$ be two accepting conversations with $(c_1, c_2) \neq (c_1', c_2')$. Then $c_1 \neq c_1'$ and/or $c_2 \neq c_2'$. If $c_1 \neq c_1'$, it follows from the special soundness of Schnorr's protocol that the witness is obtained as $x = (r_1 - r_1')/(c_1 - c_1')$. More concretely, we have:

$$\begin{aligned} & g^{r_1} = a_1 h^{c_1}, \quad g^{r_1'} = a_1 h^{c_1'} \\ \Rightarrow \quad & g^{r_1 - r_1'} = h^{c_1 - c_1'} \\ \Leftrightarrow \quad & h = g^{\frac{r_1 - r_1'}{c_1 - c_1'}}. \end{aligned}$$

[2] Parallel composition is also known as parallel repetition.

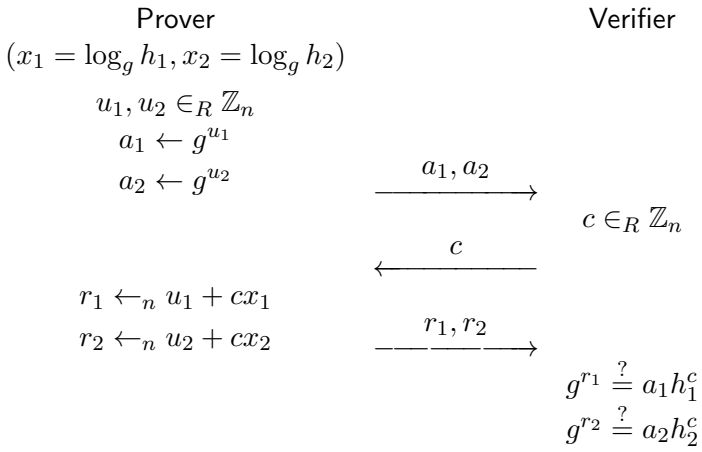


FIGURE 5.5: AND-composition of Schnorr's protocol

Similarly, if $c_2 \neq c_2'$, the (same) witness is obtained as $x = (r_2 - r_2')/(c_2 - c_2')$. Hence, the witness x satisfying $h = g^x$ can be recovered in either case.

Finally, special honest-verifier zero-knowledgeness follows from the fact that the Schnorr protocol is so, and parallel composition uses two independent runs of Schnorr's protocol. More concretely, we have that for an arbitrary (fixed) challenge (c_1, c_2), the distribution of the real conversations is identical to the distribution of the simulated conversations, which are respectively given by

$$\{(a_1, a_2; c_1, c_2; r_1, r_2) : u_1, u_2 \in_R \mathbb{Z}_n; a_1 \leftarrow g^{u_1}; a_2 \leftarrow g^{u_2}; r_1 \leftarrow_n u_1 + c_1 x; r_2 \leftarrow_n u_2 + c_2 x\},$$
$$\{(a_1, a_2; c_1, c_2; r_1, r_2) : r_1, r_2 \in_R \mathbb{Z}_n; a_1 \leftarrow g^{r_1} h^{-c_1}; a_2 \leftarrow g^{r_2} h^{-c_2}\}.$$

Note that in both cases each conversation occurs exactly with probability $1/n^2$. □

5.2.2 AND-COMPOSITION

Given two relations $R_1 = \{(v_1; w_1)\}$ and $R_2 = \{(v_2; w_2)\}$, a Σ-protocol is obtained for relation $R_1 \wedge R_2 := \{(v_1, v_2; w_1, w_2) : (v_1; w_1) \in R_1, (v_2; w_2) \in R_2\}$ by running a Σ-protocol for R_1 and a Σ-protocol for R_2 in parallel, using a *common* challenge (assuming that both protocols use the same challenge space).

Given two public keys h_1 and h_2, a proof of knowledge of both $\log_g h_1$ and $\log_g h_2$ is obtained, by running two instances of Schnorr's protocol in parallel, using a common challenge, as shown in Figure 5.5.

PROPOSITION 5.8 *The protocol in Figure 5.5 is a Σ-protocol for relation*

$$\{(h_1, h_2; x_1, x_2) : h_1 = g^{x_1}, h_2 = g^{x_2}\}.$$

PROOF Completeness is easily seen to hold, as

$$g^{r_1} = g^{u_1 + cx_1} = g^{u_1}(g^{x_1})^c = a_1 h_1^c,$$
$$g^{r_2} = g^{u_2 + cx_2} = g^{u_2}(g^{x_2})^c = a_2 h_2^c.$$

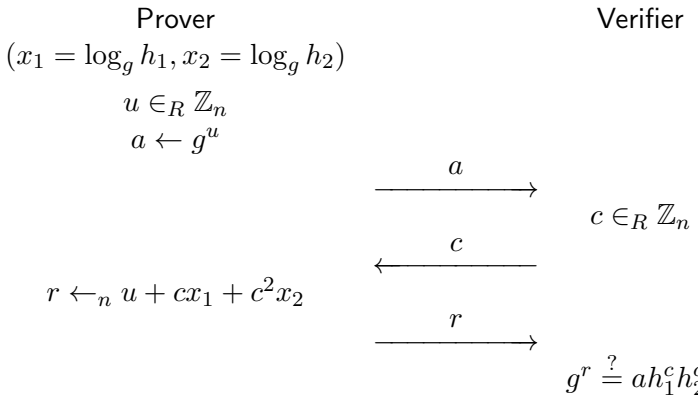


FIGURE 5.6: Alternative to AND-composition of Schnorr's protocol?

Special soundness is proved as follows. Given accepting conversations $(a_1, a_2; c; r_1, r_2)$ and $(a_1, a_2; c'; r_1', r_2')$ with $c \neq c'$, we have:

$$\begin{aligned} & g^{r_1} = a_1 h_1^c, \quad g^{r_1'} = a_1 h_1^{c'} \\ \Rightarrow \quad & g^{r_1 - r_1'} = h_1^{c-c'} \\ \Leftrightarrow \quad & h_1 = g^{\frac{r_1 - r_1'}{c-c'}}. \end{aligned}$$

By symmetry, we also have:

$$h_2 = g^{\frac{r_2 - r_2'}{c-c'}}.$$

The witness (x_1, x_2) is thus obtained as $x_1 = (r_1 - r_1')/(c - c')$ and $x_2 = (r_2 - r_2')/(c - c')$.

Finally, for special honest-verifier zero-knowledgeness, let c be a given challenge. The honest-verifier distribution and simulated distribution are, respectively:

$$\{(a_1, a_2; c; r_1, r_2) : u_1, u_2 \in_R \mathbb{Z}_n; a_1 \leftarrow g^{u_1}; a_2 \leftarrow g^{u_2}; r_1 \leftarrow_n u_1 + cx_1; r_2 \leftarrow_n u_2 + cx_2\},$$
$$\{(a_1, a_2; c; r_1, r_2) : r_1, r_2 \in_R \mathbb{Z}_n; a_1 \leftarrow g^{r_1} h_1^{-c}; a_2 \leftarrow g^{r_2} h_2^{-c}\}.$$

These distributions are identical, each conversation occurring with probability $1/n^2$. □

EXERCISE 5.9 By considering the special soundness property, explain why running Schnorr's protocol for h_1 in parallel to Schnorr's protocol for h_2 does not yield a Σ-protocol for relation $\{(h_1, h_2; x_1, x_2) : h_1 = g^{x_1}, h_2 = g^{x_2}\}$. Hint: consider a prover who knows $x_1 = \log_g h_1$ but does not know $x_2 = \log_g h_2$.

EXERCISE 5.10 Consider the protocol in Figure 5.6 as a possible Σ-protocol for relation $\{(h_1, h_2; x_1, x_2) : h_1 = g^{x_1}, h_2 = g^{x_2}\}$. (i) Show that the protocol is complete and special honest-verifier zero-knowledge. (ii) Why does special soundness not hold for this protocol? Hint: consider a prover who knows $x_1 = \log_g h_1$ but does not know $x_2 = \log_g h_2$. (iii) Show that soundness holds in the following sense: for any $(h_1, h_2) \in \langle g \rangle \times \langle g \rangle$, given three accepting conversations $(a; c; r)$, $(a; c'; r')$, $(a; c''; r'')$ with $c \neq c'$, $c \neq c''$, $c' \neq c''$ one can efficiently compute witness (x_1, x_2) satisfying $h_1 = g^{x_1}$ and $h_2 = g^{x_2}$.

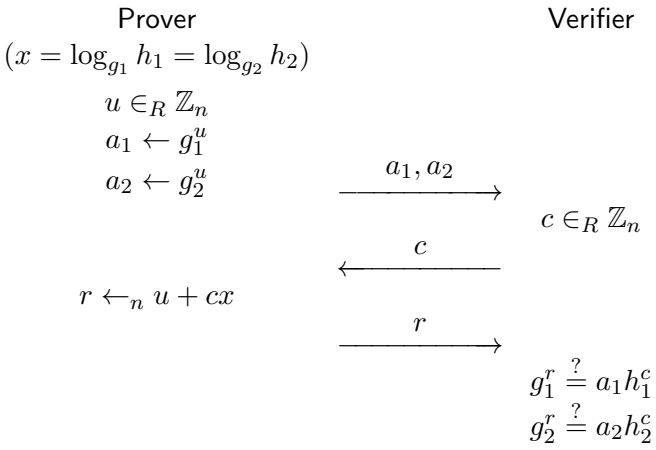


FIGURE 5.7: EQ-composition of Schnorr's protocol

5.2.3 EQ-COMPOSITION

As another important example of composition of Σ-protocols, we consider a special case of AND-composition, in which the prover uses a *common* witness for two instances of a relation. That is, we give a Σ-protocol for relation $\{(v_1, v_2; w) : (v_1; w) \in R, (v_2; w) \in R\}$, given a Σ-protocol for relation R.

The basic idea is to use AND-composition of the Σ-protocol for R, but this time the prover uses the *same* random tape $u_{\mathcal{P}}$ (see Figure 5.1) in both cases (and the same witness w).

We give an example based on Schnorr's protocol. Note that it does not make much sense to consider a single generator g and two public keys h_1 and h_2, for which we prove knowledge of an x such that $x = \log_g h_1 = \log_g h_2$: this would imply that $h_1 = h_2$. Therefore, we will work with two generators g_1, g_2.

Given two public keys g_1, h_1 and g_2, h_2, one proves knowledge of $x = \log_{g_1} h_1 = \log_{g_2} h_2$, by running two instances of the Schnorr protocol in parallel, using a *common* random choice, a *common* challenge and a *common* response.

PROPOSITION 5.11 *The protocol in Figure 5.7 is a Σ-protocol for relation*

$$\{(g_1, h_1, g_2, h_2; x) : h_1 = g_1^x, h_2 = g_2^x\}.$$

PROOF Completeness is easily checked:

$$g_1^r = g_1^{u+cx} = g_1^u (g_1^x)^c = a_1 h_1^c,$$
$$g_2^r = g_2^{u+cx} = g_2^u (g_2^x)^c = a_2 h_2^c.$$

To show special soundness, consider accepting conversations $(a_1, a_2; c; r)$ and $(a_1, a_2; c'; r')$ with $c \neq c'$. Then we have:

$$\begin{aligned} & g_1^r = a_1 h_1^c, \quad g_2^r = a_2 h_2^c, \quad g_1^{r'} = a_1 h_1^{c'}, \quad g_2^{r'} = a_2 h_2^{c'} \\ \Rightarrow \quad & g_1^{r-r'} = h_1^{c-c'}, \quad g_2^{r-r'} = h_2^{c-c'} \\ \Leftrightarrow \quad & h_1 = g_1^{\frac{r-r'}{c-c'}}, \quad h_2 = g_2^{\frac{r-r'}{c-c'}}. \end{aligned}$$

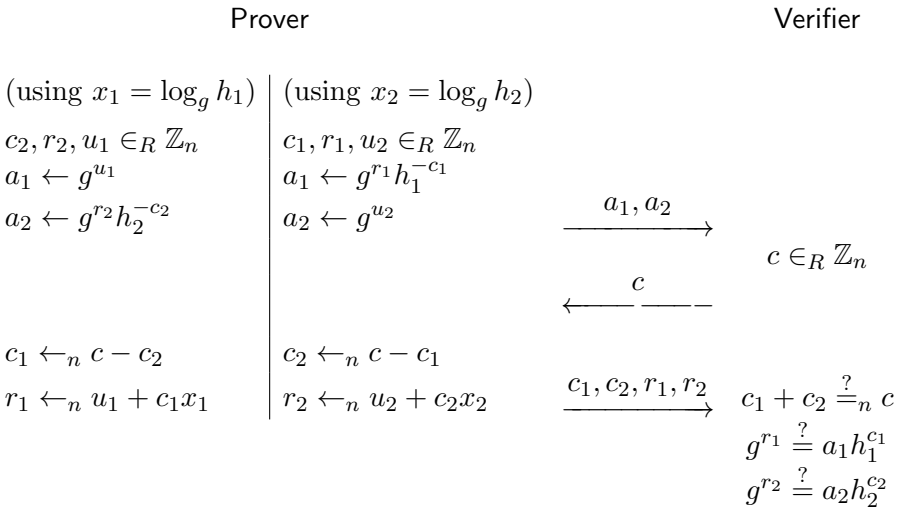


FIGURE 5.8: OR-composition of Schnorr's protocol

Hence, the witness is obtained as $x = (r - r')/(c - c')$.

Finally, for special honest-verifier zero-knowledgeness, let c be any given challenge. The distributions for the conversations with an honest verifier and the simulated conversations are, respectively:

$$\{(a_1, a_2; c; r) : u \in_R \mathbb{Z}_n; a_1 \leftarrow g_1^u; a_2 \leftarrow g_2^u; r \leftarrow_n u + cx\},$$
$$\{(a_1, a_2; c; r) : r \in_R \mathbb{Z}_n; a_1 \leftarrow g_1^r h_1^{-c}; a_2 \leftarrow g_2^r h_2^{-c}\},$$

These distributions are identical provided $\log_{g_1} h_1 = \log_{g_2} h_2$, cf. Definition 5.1; furthermore, if $\log_{g_1} h_1 \neq \log_{g_2} h_2$, the simulated conversations are accepting, as required. □

5.2.4 OR-COMPOSITION

Our next goal is to construct a Σ-protocol for relation $R_1 \vee R_2 = \{(v_1, v_2; w_1, w_2) : (v_1; w_1) \in R_1 \vee (v_2; w_2) \in R_2\}$, given Σ-protocols for relations R_1 and R_2. This turns out to be possible, using a Σ-protocol of similar complexity as used for AND-composition.

Suppose that the prover knows a witness (w_1, w_2) with $(v_1; w_1) \in R_1$. Hence, the prover knows a witness for $R_1 \vee R_2$. The prover is able to run the Σ-protocol for R_1. However, the prover need not be able to do so for R_2 as it need not know a witness w_2 such that $(v_2; w_2) \in R_2$. Of course, if the prover knows a witness (w_1, w_2) with $(v_2; w_2) \in R_2$ and possibly $(v_1; w_1) \notin R_1$ the same problem arises.

The way out is that the verifier may let the prover "cheat" a little bit by allowing the prover to use the *simulator* of the Σ-protocol for the relation R_i for which the prover does not know witness w_i ($i = 1$ or $i = 2$). The verifier will do so by providing a single challenge c which the prover is allowed to split into two challenges c_1, c_2 provided c_1, c_2 satisfy a *linear* constraint in terms of c. For example, the constraint may be $c = c_1 \oplus c_2$ if the challenges happen to be bit strings. Essentially, the prover gets one degree of freedom to cheat.

Given two public keys h_1 and h_2, a proof of knowledge of either $x_1 = \log_g h_1$ or $x_2 = \log_g h_2$ (or, possibly, both) is obtained, by composing one run of Schnorr's protocol with one run of the

simulator for Schnorr's protocol, as shown in Figure 5.8. The respective challenges $c_1, c_2 \in \mathbb{Z}_n$ must sum to c (modulo n). If the prover knows x_1 it follows the steps on the left-hand side of the vertical bar. The values $(a_1; c_1; r_1)$ are generated as in a normal run of Schnorr's protocol (note that $c_1 \in_R \mathbb{Z}_n$ since $c \in_R \mathbb{Z}_n$). The values $(a_2; c_2; r_2)$ are simulated. If the prover knows x_2, it is the other way around.

PROPOSITION 5.12 *The protocol in Figure 5.8 is a Σ-protocol for relation*

$$\{(h_1, h_2; x_1, x_2) : h_1 = g^{x_1} \vee h_2 = g^{x_2}\}.$$

PROOF To show completeness, consider the case that the prover uses x_1. Then we have (modulo n):

$$c_1 + c_2 = c - c_2 + c_2 = c,$$

and also, by inspection of the protocol,

$$\begin{aligned} g^{r_1} &= g^{u_1 + c_1 x_1} = g^{u_1}(g^{x_1})^{c_1} = a_1 h_1^{c_1}, \\ g^{r_2} &= a_2 h_2^{c_2}. \end{aligned}$$

Similarly, completeness also holds if the prover uses x_2.

Next, we show special soundness. Suppose accepting conversations $(a_1, a_2; c; c_1, c_2, r_1, r_2)$ and $(a_1, a_2; c'; c_1', c_2', r_1', r_2')$ are given, with $c \neq c'$. Since $c = c_1 + c_2 \neq c_1' + c_2' = c'$, it follows that $c_1 \neq c_1'$ or $c_2 \neq c_2'$ (or, possibly, both). We also have:

$$\begin{aligned} & g^{r_1} = a_1 h_1^{c_1}, \quad g^{r_2} = a_2 h_2^{c_2}, \quad g^{r_1'} = a_1 h_1^{c_1'}, \quad g^{r_2'} = a_2 h_2^{c_2'} \\ \Rightarrow \quad & g^{r_1 - r_1'} = h_1^{c_1 - c_1'}, \quad g^{r_2 - r_2'} = h_2^{c_2 - c_2'}. \end{aligned}$$

If $c_1 \neq c_1'$, we obtain witness $x_1 = (r_1 - r_1')/(c_1 - c_1')$ satisfying $h_1 = g^{x_1}$; otherwise $c_2 \neq c_2'$, and we obtain witness $x_2 = (r_2 - r_2')/(c_2 - c_2')$ satisfying $h_2 = g^{x_2}$. So, we either obtain a correct witness x_1 or a correct witness x_2 (or both).

Finally, we show that special honest-verifier zero-knowledgeness holds. The honest-verifier distribution and simulated distribution are, respectively (again considering the case of a prover using x_1):

$$\begin{aligned} &\{(a_1, a_2; c; c_1, c_2, r_1, r_2) : u_1, r_2, c_2 \in_R \mathbb{Z}_n; a_1 \leftarrow g^{u_1}; a_2 \leftarrow g^{r_2} h_2^{-c_2}; c_1 \leftarrow_n c - c_2; \\ &\qquad\qquad r_1 \leftarrow_n u_1 + c_1 x_1\}, \\ &\{(a_1, a_2; c; c_1, c_2, r_1, r_2) : c_1, r_1, r_2 \in_R \mathbb{Z}_n; c_2 \leftarrow_n c - c_1; a_1 \leftarrow g^{r_1} h_1^{-c_1}; a_2 \leftarrow g^{r_2} h_2^{-c_2}\}, \end{aligned}$$

for any given challenge c. These distributions are identical (uniform distribution on all accepting conversations, each one occurring with a probability of $1/n^3$). □

EXERCISE 5.13 As a slight optimization of the protocol of Figure 5.8, note that the prover may omit sending the value of c_2, in which case the verifier must replace the test $c_1 + c_2 \stackrel{?}{=}_n c$ by the assignment $c_2 \leftarrow_n c - c_1$. (Thus, the prover omits sending c_2 independent of whether it knows x_1 and/or x_2.) Prove that the resulting protocol is a Σ-protocol for the same relation as before.

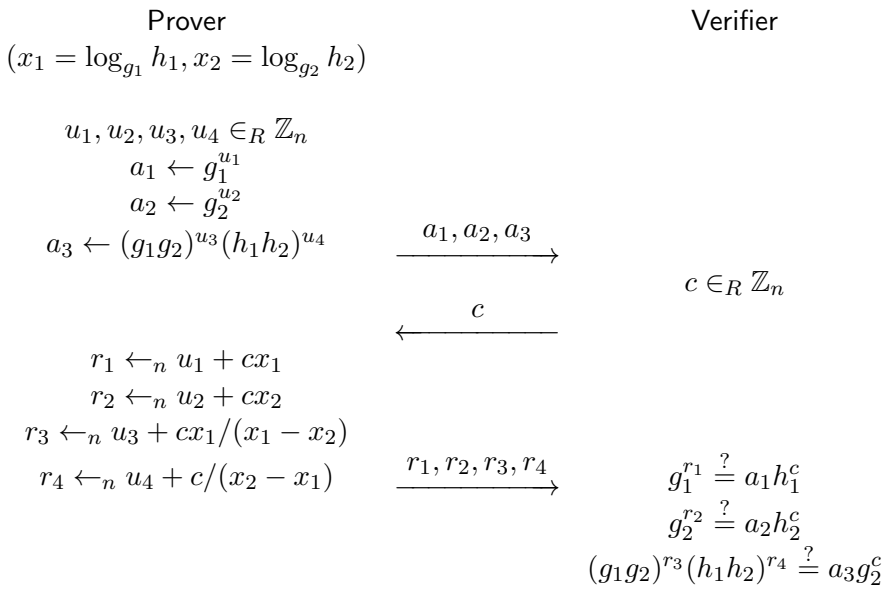


FIGURE 5.9: NEQ-composition of Schnorr's protocol

5.2.5 NEQ-COMPOSITION

Finally, as the counterpart of EQ-composition, we consider NEQ-composition, which is a form of AND-composition with the additional property that the two witnesses are different from each other. We consider NEQ-composition for two instances of a given relation. That is, we give a Σ-protocol for relation $\{(v_1, v_2; w_1, w_2) : (v_1; w_1) \in R, (v_2; w_2) \in R, w_1 \neq w_2\}$, given a Σ-protocol for relation R.

As a first step, note that it is easy to use AND-composition of the Σ-protocol for R to prove knowledge of both witnesses. The protocol then needs to be extended to show that the witnesses are indeed different.

We give an example based on Schnorr's protocol. Again, we work with two generators g_1, g_2, this time assuming that $\log_{g_1} g_2$ is unknown. Given two public keys g_1, h_1 and g_2, h_2, we know how to prove knowledge of $x_1 = \log_{g_1} h_1$ and $x_2 = \log_{g_2} h_2$. Moreover, since $x_1 \neq x_2$ we have that $x_1 - x_2 \neq 0$, hence we know that the multiplicative inverse of $x_1 - x_2$ modulo n is defined. Starting from

$$h_1 h_2 = g_1^{x_1} g_2^{x_2} = (g_1 g_2)^{x_1} g_2^{x_2 - x_1},$$

we therefore have

$$g_2 = (g_1 g_2)^{x_1/(x_1 - x_2)} (h_1 h_2)^{1/(x_2 - x_1)}. \tag{5.1}$$

Using Okamoto's protocol we may thus prove that we can write g_2 as a product of powers of $g_1 g_2$ and $h_1 h_2$. This will suffice to get the desired protocol as the AND-composition of two instances of Schnorr's protocol and one instance of Okamoto' protocol.

PROPOSITION 5.14 *The protocol in Figure 5.9 is a Σ-protocol for relation*

$$\{(g_1, h_1, g_2, h_2; x_1, x_2) : h_1 = g_1^{x_1}, h_2 = g_2^{x_2}, x_1 \neq x_2\},$$

assuming that $\log_{g_1} g_2$ is unknown.

PROOF As for completeness, clearly $g_i^{r_i} = a_i h_i^c$ holds for $i = 1, 2$. Furthermore, we have:

$$(g_1 g_2)^{r_3}(h_1 h_2)^{r_4} = (g_1 g_2)^{u_3}(h_1 h_2)^{u_4}((g_1 g_2)^{cx_1/(x_1-x_2)}(h_1 h_2)^{c/(x_2-x_1)}) = a_3 g_2^c,$$

using Eq. (5.1).

For special soundness, let $(a_1, a_2, a_3; c; r_1, r_2, r_3, r_4)$ and $(a_1, a_2, a_3; c'; r_1', r_2', r_3', r_4')$ be two accepting conversations with $c \neq c'$. This implies that we can extract witness (x_1, x_2), with $x_1 = (r_1' - r_1)/(c' - c)$ and $x_2 = (r_2' - r_2)/(c' - c)$ satisfying $h_1 = g_1^{x_1}$ and $h_2 = g_2^{x_2}$. Moreover, this implies that

$$g_2 = (g_1 g_2)^{(r_3 - r_3')/(c-c')}(h_1 h_2)^{(r_4 - r_4')/(c-c')}.$$

Now, suppose $x_1 = x_2$. Then, we can write $g_2 = (g_1 g_2)^\alpha$ for a known value of $\alpha \neq 0, 1$ (using that $g_1, g_2 \neq 1$), hence $g_2 = g_1^{\alpha/(1-\alpha)}$, contradicting that $\log_{g_1} g_2$ is unknown. Therefore, $x_1 \neq x_2$ follows, and we have shown that (x_1, x_2) is a valid witness.

Finally, for special honest-verifier zero-knowledgeness, let c be a given challenge. The distributions for the conversations with an honest verifier and for the simulated conversations are, respectively:

$$\begin{aligned}
&\{(a_1, a_2, a_3; c; r_1, r_2, r_3, r_4) : u_1, u_2, u_3, u_4 \in_R \mathbb{Z}_n; a_1 \leftarrow g_1^{u_1}; a_2 \leftarrow g_2^{u_2};\\
&\qquad a_3 \leftarrow (g_1 g_2)^{u_3}(h_1 h_2)^{u_4}; r_1 \leftarrow_n u_1 + cx_1; r_2 \leftarrow_n u_2 + cx_2;\\
&\qquad\qquad r_3 \leftarrow_n u_3 + cx_1/(x_1 - x_2); r_4 \leftarrow_n u_4 + c/(x_2 - x_1)\},\\
&\{(a_1, a_2, a_3; c; r_1, r_2, r_3, r_4) : r_1, r_2, r_3, r_4 \in_R \mathbb{Z}_n; a_1 \leftarrow g_1^{r_1} h_1^{-c}; a_2 \leftarrow g_2^{r_2} h_2^{-c};\\
&\qquad\qquad a_3 \leftarrow (g_1 g_2)^{r_3}(h_1 h_2)^{r_4} g_2^{-c}\}.
\end{aligned}$$

Using Eq. (5.1), these distributions can be seen to be identical provided $\log_{g_1} h_1 \neq \log_{g_2} h_2$, cf. Definition 5.1; furthermore, if $\log_{g_1} h_1 = \log_{g_2} h_2$, the simulated conversations are accepting, as required. □

5.3 MISCELLANEOUS CONSTRUCTIONS

In this section we explore some more examples of Σ-protocols, using various applications of AND-composition, EQ-composition, OR-composition, and NEQ-composition. We start out with an elaborate example, followed by several exercises.

EXAMPLE 5.15 *Let g, h denote generators of a group of large prime order n such that $\log_g h$ is unknown to anyone. Suppose we need to design a Σ-protocol for the following (arbitrary) relation:*

$$R = \{(A, B; x, y, z) : A = g^x h^y \ \wedge \ B = g^{xy} h^{(1-x)z} \ \wedge \ x \in \{0, 1\}\}.$$

To break down the problem, we distinguish the cases $x = 0$ and $x = 1$ for a given pair $(A, B; x, y, z) \in R$. If $x = 0$, then we have for a such a pair that $A = h^y \ \wedge \ B = h^z$ holds. Similarly, if $x = 1$, then we have that $A = gh^y \ \wedge \ B = g^y$ holds. Therefore, it suffices to design protocols for the following two relations and combine these using OR-composition:

$$\begin{aligned}
R_0 &= \{(A, B; y, z) : A = h^y \ \wedge \ B = h^z\},\\
R_1 &= \{(A, B; y) : A = gh^y \ \wedge \ B = g^y\}.
\end{aligned}$$

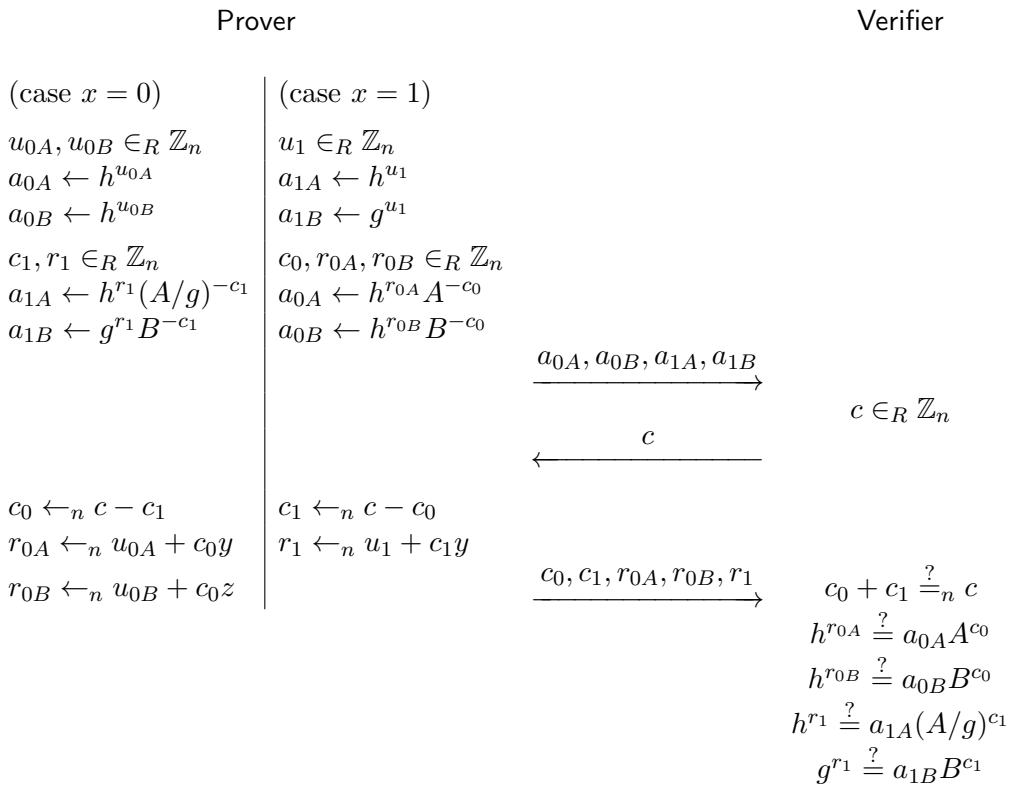


FIGURE 5.10: Σ-protocol for $\{(A, B; x, y, z) : A = g^x h^y \wedge B = g^{xy} h^{(1-x)z} \wedge x \in \{0, 1\}\}$

To obtain a Σ-protocol for relation R_0, we apply AND-composition to the following relations:

$$\begin{aligned} R_{0A} &= \{(A; y) : A = h^y\}, \\ R_{0B} &= \{(B; z) : B = h^z\}. \end{aligned}$$

To obtain a Σ-protocol for relation R_1, we apply EQ-composition to the following relations:

$$\begin{aligned} R_{1A} &= \{(A; y) : A/g = h^y\}, \\ R_{1B} &= \{(B; y) : B = g^y\}. \end{aligned}$$

Each of the relations R_{0A}, R_{0B}, R_{1A}, and R_{1B} can be handled by an instance of Schnorr's protocol, or a slight variation thereof. Hence, Σ-protocols for these relations are easily obtained. The complete protocol is given in Figure 5.10.

Protocols for relations such as in Example 5.15 should be correct by construction. Nevertheless, to exclude mistakes or errors in the construction, a direct proof showing that the Σ-protocol is correct may be prudent and instructive. In particular, to check that the soundness and zero-knowledge properties indeed hold.

EXERCISE 5.16 Prove that the protocol of Figure 5.10 is a Σ-protocol for relation R, as defined in Example 5.15.

EXERCISE 5.17 Let g, h denote generators of a group of large prime order n such that $\log_g h$ is unknown to anyone. Let $B = g^x h^y$ denote the common input to prover and verifier, where $x, y \in \mathbb{Z}_n$ is private input to the prover. In each of the following cases, design a Σ-protocol for proving knowledge of $x, y \in \mathbb{Z}_n$ such that $B = g^x h^y$ and $\psi(x, y)$ holds, for given predicate $\psi(x, y)$. In each case, prove that your protocol is indeed a Σ-protocol for the relation $\{(B; x, y) : B = g^x h^y, \psi(x, y)\}$; see hints below.

(a) $\psi(x, y) \equiv$ true;

(b) $\psi(x, y) \equiv x = y$;

(c) $\psi(x, y) \equiv \alpha x + \beta y = \gamma$ for given $\alpha, \beta \in \mathbb{Z}_n^*$, $\gamma \in \mathbb{Z}_n$;

(d) $\psi(x, y) \equiv x \in \{0, 1\}$;

(e) $\psi(x, y) \equiv x \in \{0, 1, \ldots, 2^\ell - 1\}$, where ℓ is a fixed integer, $1 \leq \ell \leq \lfloor \log_2 n \rfloor$;

(f) $\psi(x, y) \equiv x \neq 0$;

(g) $\psi(x, y) \equiv x \neq y$;

(h) $\psi(x, y) \equiv \alpha x + \beta y \neq \gamma$ for given $\alpha, \beta \in \mathbb{Z}_n^*$, $\gamma \in \mathbb{Z}_n$;

(i) $\psi(x, y) \equiv xy = 1$;

(j) $\psi(x, y) \equiv \exists_{\chi \in \mathbb{Z}_n} x = \chi^2$;

(k) $\psi(x, y) \equiv x^2 = y^2$.

Hints: for part (a) use Okamoto's protocol (see Figure 4.5); for part (b) consider modifications of EQ-composition of Schnorr's protocol (see Figure 5.7), or eliminate variable y directly using that $x = y$; for part (c) eliminate one of the variables x,y using the given equation for x and y; for part (d) use a protocol similar to OR-composition; for part (e) consider the binary representation of x and use k instances of the protocol of part (d); for part (f) use an instance of Okamoto's protocol by isolating g on one side of the equation $B = g^x h^y$, similar to Eq. (5.1) in NEQ-composition; parts (g) and (h) are generalizations of part (f); for part (i) isolate g on one side of the equation for B^y and use EQ-composition with equation for B; for part (j) use AND-composition and EQ-composition; for part (k) eliminate one of the variables and use OR-composition.

EXERCISE 5.18 See Figure 4.5. Is the protocol of Figure 5.11 a Σ-protocol for the relation $\{(h; x_1, x_2) : h = g_1^{x_1} g_2^{x_2}\}$?

EXERCISE 5.19 Let g, h denote generators of a group of large prime order n such that $\log_g h$ is unknown to anyone. Design Σ-protocols (and prove correctness) for the following relations:

(a) $\{(A, B; x, y, z) : A = g^x h^y, B = g^{1/x} h^z, x \neq 0\}$;

(b) $\{(A_1, A_2, B; x_1, x_2, y_1, y_2, z) : A_1 = g^{x_1} h^{y_1}, A_2 = g^{x_2} h^{y_2}, B = g^{x_1 x_2} h^z\}$.

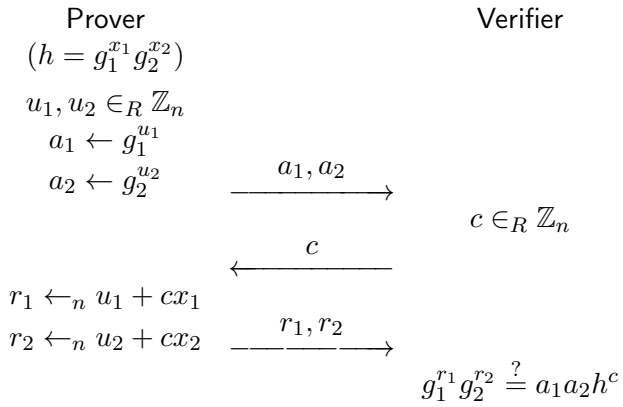


FIGURE 5.11: Alternative to Okamoto's protocol?

EXERCISE 5.20 Let g, h denote generators of a group of large prime order n such that $\log_g h$ is unknown to anyone. Consider an instance of the 3SAT problem for Boolean variables $v_1, \ldots, v_\ell$, given by a Boolean formula Φ consisting of m clauses, which each consist of *three* literals:

$$\Phi = (l_{1,1} \vee l_{1,2} \vee l_{1,3}) \wedge \cdots \wedge (l_{m,1} \vee l_{m,2} \vee l_{m,3}).$$

Each literal is of the form $l_{i,j} = v_k$ or $l_{i,j} = \overline{v}_k = 1 - v_k$ (negation of v_k), $1 \leq k \leq \ell$. Construct a Σ-protocol for the following relation:

$$R'_{3\text{SAT}} = \{(\Phi, B_1, \ldots, B_\ell; x_1, y_1, \ldots, x_\ell, y_\ell) : \Phi(x_1, \ldots, x_\ell), \forall_{k=1}^{\ell} B_k = g^{x_k} h^{y_k}, x_k \in \{0, 1\}\}.$$

EXERCISE 5.21 See the previous exercise. A Σ-protocol for $R'_{3\text{SAT}}$ actually proves knowledge of witnesses to open the given commitments $B_1, \ldots, B_\ell$. Construct a more flexible way for proving that Φ is satisfiable, by considering the following relation instead:

$$R_{3\text{SAT}} = \{(\Phi; x_1, \ldots, x_\ell) : \Phi(x_1, \ldots, x_\ell)\}.$$

5.4 NON-INTERACTIVE Σ-PROOFS

Recall that there are basically two forms of authentication schemes: interactive authentication schemes (e.g., identification schemes) and non-interactive authentication schemes (e.g., digital signature schemes). Similarly, one may distinguish two forms of zero-knowledge proof schemes: interactive proof schemes and non-interactive proof schemes. An interactive proof scheme comprises a protocol by which a prover convinces a verifier that a certain statement holds. A non-interactive proof scheme comprises an algorithm by which a prover generates a proof for a certain statement and another algorithm by which a verifier may verify a given proof.

It turns out that there is a simple but effective way to make any Σ-protocol non-interactive, known as the Fiat-Shamir heuristic. To emphasize the difference between an interactive Σ-protocol and its non-interactive counterpart, we will refer to the non-interactive version as a Σ-proof.

A distinctive feature of a non-interactive Σ-proof is that any entity may play the role of the verifier. As a consequence, a non-interactive Σ-proof can be verified independently by

many entities—just as a digital signature can be verified by anyone who is interested in its validity.

5.4.1 DIGITAL SIGNATURES FROM Σ-PROTOCOLS

A digital signature scheme consists of three algorithms: a key generation algorithm, a signature generation algorithm, and a signature verification algorithm. By means of an example we will show how to convert any Σ-protocol (used for identification) into a corresponding digital signature scheme, when given a cryptographic hash function $H : \{0,1\}^* \to \{0,1\}^k$, for a suitable value of k (see below).

Consider Schnorr's protocol (see Figure 4.3) for proving knowledge of $x = \log_g h$, for a given public key h. The Schnorr signature scheme is obtained by applying the Fiat-Shamir heuristic to Schnorr's protocol. This means that—rather than picking challenge c uniformly at random (and independent of announcement a)—challenge c is computed as a hash value of announcement a and message M, that is, $c \leftarrow H(a, M)$. In this way, no interaction with the verifier is required anymore, as the prover may compute challenge c on its own. The security of the resulting signature scheme follows if one views H as a random oracle. The intuition is that if H is a random function, then the value of challenge $c = H(a, M)$ will appear completely random to the prover for each new message M to be signed, hence c follows the same distribution as when the prover interacts with an honest verifier. Moreover, the prover must first choose announcement a before challenge c can be computed.

We will assume that $2^k \leq n$. As a consequence, bit strings in $\{0,1\}^k$ can be identified with integers in $\{0, 1, \ldots, 2^k - 1\} \subseteq \mathbb{Z}_n$. The Schnorr signature scheme is composed of the following three algorithms, where all users may share the group $\langle g \rangle$ of prime order n and hash function $H : \{0,1\}^* \to \{0,1\}^k$.

Key generation. A key pair $(h; x)$ is generated by choosing private key $x \in_R \mathbb{Z}_n$ and then setting public key h by $h \leftarrow g^x$.

Signature generation. On input of a message M and a private key x, choose $u \in_R \mathbb{Z}_n$, set $a \leftarrow g^u$, $c \leftarrow H(a, M)$, and $r \leftarrow_n u + cx$. The signature on M is the pair (c, r).

Signature verification. On input of a message M, a pair (c, r), and a public key h, accept (c, r) as a signature on M if and only if $c = H(g^r h^{-c}, M)$ holds.

Clearly, the Schnorr signature scheme is quite efficient. The computational cost of signature generation is dominated by the time to perform a single exponentiation of the form g^u; if desired, this exponentiation can be done beforehand, that is, before the message M is known. The computational cost of signature verification is dominated by the time to perform a "double" exponentiation of the form $g^r h^{-c}$; such an exponentiation can be computed considerably faster than just computing g^r and h^{-c} and multiplying the result. Since a Schnorr signature (c, r) consists of two numbers in $\mathbb{Z}_n$, the size of a signature is rather small (in practice, n may be a 256-bit number, hence a signature is just 512 bits.)

The Fiat-Shamir heuristic for converting Σ-protocols into signature schemes can be proved secure (against adaptive chosen-message attacks) in the random oracle model. The next exercise, however, shows that for *contrived* cases the resulting signature scheme may be insecure.

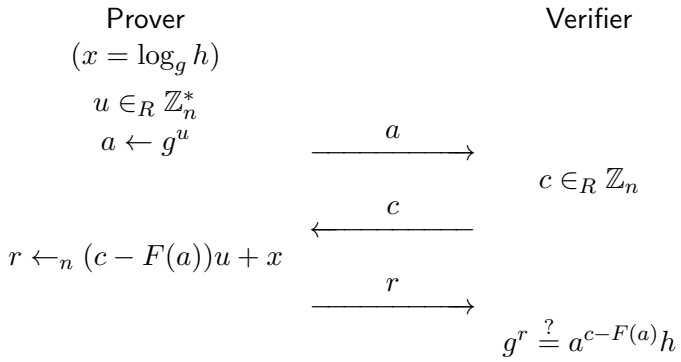


FIGURE 5.12: Parametrized insecure variant of Schnorr's protocol

EXERCISE 5.22 To see that the Fiat-Shamir heuristic does not necessarily lead to secure signature schemes, consider the following variant of Schnorr's protocol, see Figure 5.12. The function $F : \langle g \rangle \to \mathbb{Z}_n$ can be replaced by your favorite hash function; for simplicity it is assumed that the hash function maps into $\mathbb{Z}_n$. Note that the constant function $F(w) = 0$, for $w \in \langle g \rangle$, yields the protocol of Exercise 5.4.

(i) Show that the protocol is complete, special sound, and honest-verifier zero-knowledge (for any function $F : \langle g \rangle \to \mathbb{Z}_n$).

(ii) What happens if we generate the challenge as $c \leftarrow F(a)$ to obtain a non-interactive version of the protocol? That is, what happens if we instantiate the random oracle with $H = F$.

5.4.2 PROOFS OF VALIDITY

Consider a Pedersen commitment of the form $B = g^x h^y$, where $x \in \mathbb{Z}_n$ is the committed value and $y \in_R \mathbb{Z}_n$ (cf. Section 3.2.2). Now, suppose that it must be ensured that the committed value x is actually a bit, that is, $x \in \{0, 1\}$. We may do so by requiring the committer to execute a Σ-protocol proving that indeed $x \in \{0, 1\}$, for a given commitment B (see Exercise 5.17(d)).

In many applications, however, it is undesirable that the committer would need to engage in an interactive protocol with every potential verifier (of which there may be many). A better approach is therefore to apply the Fiat-Shamir heuristic to the Σ-protocol for proving the given statement. The resulting non-interactive Σ-proof may then be verified by anyone who is interested in its validity.

DEFINITION 5.23 *Let H be a cryptographic hash function. For any Σ-protocol as in Figure 5.1, a (non-interactive)* ***Σ-proof for relation R*** *is defined in terms of two algorithms.*

Proof generation. *Given $(v; w) \in R$, a Σ-proof is a pair $(\alpha(v; w; u_{\mathcal{P}}); \rho(v; w; H(a; v); u_{\mathcal{P}}))$.*

Proof verification. *For $v \in V$, $(a; r)$ is accepted as Σ-proof if and only if $\varphi(v; a; H(a; v); r)$.*

In general, a Σ-proof thus consists of an announcement a and a response r. Often, however, it is possible to reduce the size of a Σ-proof by replacing (parts of) a by the challenge c. A

well-known example is the above Schnorr signature scheme, in which $(c; r)$ is used as signature instead of $(a; r)$, exploiting the fact that a can be recovered from $(c; r)$ as $a = g^r h^{-c}$. Another typical example is the following Σ-proof for relation $\{(B; x, y) : B = g^x h^y, x \in \{0, 1\}\}$, where the reduction in size is quite substantial.

EXAMPLE 5.24 *The Σ-protocol of Exercise 5.17(d) is converted into a Σ-proof as follows. Let $B = g^x h^y$ be given and let $x \in \{0, 1\}, y \in \mathbb{Z}_n$ be the private input for the prover. Write $\overline{x} = 1 - x$ for $x \in \{0, 1\}$.*

Proof generation. *Choose $u_x, c_{\overline{x}}, r_{\overline{x}} \in_R \mathbb{Z}_n$, and set $a_x \leftarrow h^{u_x}$ and $a_{\overline{x}} \leftarrow h^{r_{\overline{x}}}(B/g^{\overline{x}})^{-c_{\overline{x}}}$. Set the challenge c as $c \leftarrow H(a_0, a_1, B)$, and compute the responses $c_x \leftarrow_n c - c_{\overline{x}}$ and $r_x \leftarrow_n u_x + c_x y$. The Σ-proof is the tuple (c_0, c_1, r_0, r_1).*

Proof verification. *Given commitment B, a tuple (c_0, c_1, r_0, r_1) is accepted as a Σ-proof if and only if $c_0 + c_1 =_n H(h^{r_0} B^{-c_0}, h^{r_1}(B/g)^{-c_1}, B)$.*

It is important to note that for a Σ-proof, value v is included in the input to the hash function, as well as announcement a. This way the "context" (statement to be proved) is fixed, and forgery of proofs is prevented. Hence, in Example 5.24, it is important that commitment B is included in the input to the hash function. Otherwise the Σ-proof can be forged, that is, valid Σ-proofs can be generated without knowing any $x \in \{0, 1\}, y \in \mathbb{Z}_n$ satisfying $B = g^x h^y$.

EXERCISE 5.25 Show how to forge a Σ-proof (c_0, c_1, r_0, r_1) for a commitment B if proof verification is changed into $c_0 + c_1 =_n H(h^{r_0} B^{-c_0}, h^{r_1}(B/g)^{-c_1})$ by making a suitable choice for B. Hint: B can be set *after* $c = H(a_0, a_1)$ is computed.

5.4.3 GROUP SIGNATURES

As a simple application of the techniques we have seen so far, we consider the problem of designing a group signature scheme. An (anonymous) group signature scheme allows users to sign on behalf of a group, in such a way that a signature does not disclose which user actually produced the signature. If desired, e.g., in case of disputes, a group manager is still able to prove which group member produced any given group signature.

DEFINITION 5.26 *A **group signature scheme** for a group formed by a group manager $\mathcal{P}_0$ and group members $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$, $\ell \geq 1$, consists of the following four components.*

Key generation. *A protocol between $\mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_\ell$ for generating a public key h for the group, a private key x_0 for the group manager $\mathcal{P}_0$ and a private key x_i for each group member $\mathcal{P}_i$, $1 \leq i \leq \ell$.*

Signature generation. *An algorithm that on input of a message M, the public key h of the group, and a private key x_i of a group member $\mathcal{P}_i$, outputs a group signature S.*

Signature verification. *An algorithm that on input of a message M, the public key h of the group, and a signature S, determines whether S is a valid group signature on M with respect to public key h.*

Signature opening. *An algorithm that on input of a message M, the public key h of the group, a valid group signature S, and the private key x_0 of the group manager, outputs the identity of the group member who generated S.*

A group signature scheme must meet similar security requirements as a basic digital signature scheme. In addition, there are requirements related to the anonymity of a group member and the role of the group manager. Given a signature, no-one except the group manager should be able to tell which group member produced the signature.[3] More generally, given two signatures, no-one except the group manager should be able to tell whether these signature were produced by the same group member or not (this property is called *unlinkability*). Of course, group members should not be able to produce signatures on behalf of other group members. Similarly, a group manager should not be able to *frame* a group member $\mathcal{P}_i$ by opening a signature produced by $\mathcal{P}_j$ as if it was produced by $\mathcal{P}_i$ $(i \neq j)$.

We next describe a simple group signature scheme, where we assume that all parties have access to a generator g of order n. As a warm-up exercise we first consider the problem of proving knowledge of 1-out-of-ℓ private keys. The base case $\ell = 1$ is just a Schnorr proof. The case $\ell = 2$ can be solved by an OR-composition of two Schnorr proofs. The general case may be solved by repeating OR-composition $\ell - 1$ times starting from ℓ Schnorr proofs, noting that OR-composition of a 1-out-of-ℓ_1 proof and a 1-out-of-ℓ_2 proof yields a 1-out-of-$(\ell_1 + \ell_2)$ proof.

EXERCISE 5.27 Construct a Σ-protocol for the following relation

$$R_{(1,\ell)} = \{(h_1, \ldots, h_\ell; x) : \exists_{i=1}^{\ell} h_i = g^x\}$$

by generalizing the technique of OR-composition, and show that it is indeed a Σ-protocol.

Ignoring the role of the group manager for a moment, we obtain a group signature by applying the Fiat-Shamir heuristic to the Σ-protocol of Exercise 5.27. To do so, we would set $c \leftarrow H(a_1, \ldots, a_\ell, M)$ for a given message M. The group signature for M is then defined as $S = (c_1, \ldots, c_\ell, r_1, \ldots, r_\ell)$.

The complete group signature scheme is now as follows, where group member $\mathcal{P}_i$ is required to include an ElGamal encryption of g^{x_i} under the group manager's public key in each group signature produced by $\mathcal{P}_i$.

Key generation. Each group member $\mathcal{P}_i$ picks its private key $x_i \in_R \mathbb{Z}_n$. Similarly, the group manager picks its private key $x_0 \in_R \mathbb{Z}_n$. The public key of the group is then set to $h = (h_0, h_1, \ldots, h_\ell)$, where $h_i = g^{x_i}$, $0 \leq i \leq \ell$.

Signature generation. Group member $\mathcal{P}_i$ computes an ElGamal encryption of $h_i = g^{x_i}$ under the group manager's public key h_0: $(A, B) = (g^u, h_0^u g^{x_i})$, where $u \in_R \mathbb{Z}_n$. Next, $\mathcal{P}_i$ produces a Σ-proof showing that (A, B) indeed encrypts one of the values $h_1, \ldots, h_\ell$, and that it knows the corresponding private key. More precisely, a Σ-proof for the following relation is given

$$R'_{(1,\ell)} = \{(A, B, h_0, h_1, \ldots, h_\ell; u, x) : A = g^u, B = h_0^u g^x, \exists_{i=1}^{\ell} h_i = g^x\}.$$

The given message M is also included as an additional input to the hash function in the Σ-proof (similar to the way a Schnorr signature is generated).

Signature verification. The Σ-proof contained in the group signature is verified (similar to the way a Schnorr signature is verified).

[3] For simplicity, we assume that group members do not keep track of which signatures they produced and so on.

Signature opening. The group manager decrypts the ElGamal encryption contained in the group signature, and proves that it performed decryption correctly. More precisely, given ElGamal ciphertext (A, B), the group manager outputs $d = A^{x_0}$ and a Σ-proof that $\log_g h_0$ is equal to $\log_A d$, using EQ-composition. From d anyone may compute B/d which is equal to the public key of the group member who produced the signature.

EXERCISE 5.28 Give a Σ-protocol for relation $R'_{(1,\ell)}$ and prove its correctness, in each of the following cases: (i) $\ell = 1$, (ii) $\ell = 2$, and (iii) arbitrary $\ell \geq 1$.

5.5 BIBLIOGRAPHIC NOTES

The notion of a Σ-protocol was identified and studied in [Cra97], as a further abstraction of the (special) honest-verifier zero-knowledge, special-sound protocols considered in [CDS94]. Nowadays many variants of Σ-protocols exist in the cryptographic literature.[4] The defining properties of completeness, soundness, and zero-knowledgeness in this text (cf. Definition 5.1) have been chosen as strong as reasonably possible. Special attention has been paid to ensure that the notion of a Σ-protocol is preserved under the compositions treated in Section 5.2.

The transformation from plain honest-verifier zero-knowledgeness to special honest-verifier zero-knowledgeness in Figure 5.2 is from [Dam10]. OR-composition is from [CDS94], whereas Exercise 5.17d is from [CFSY96]. EQ-composition is based on the protocol for proving equality of discrete logarithms from [CP93]. The Σ-protocol for linear relations (Exercise 5.17c) and the "not" proof (Exercise 5.17f–h) are from [Bra97]. NEQ-composition is related to (but different from) both the "not" proof and the disavowal protocol in undeniable signature schemes (see, e.g., [BCDP91]). The Σ-protocol of Exercise 5.19b is from [CD98].

The Fiat-Shamir heuristic is from [FS87]. Group signatures are due to [CH91], and the scheme considered in the text using ElGamal is from [Cam97]. However, the 1-out-of-ℓ signature scheme based on the Σ-protocol of Exercise 5.27 already dates back to [CDS94, Section 5].

[4] Since 2001, the name SIGMA ("SIGn-and-MAc") is also used by Hugo Krawczyk for a certain type of authenticated key-exchange protocol. Incidentally, there is also the thriller "The Sigma Protocol" by Robert Ludlum (October 2001).

CHAPTER 6

Threshold Cryptography

In many situations it is undesirable that access to valuable items is controlled by a single party only. For example, opening a personal safe at a bank requires the use of two keys, one kept by the owner of the safe and one kept by a bank employee. Similarly, in many cryptographic schemes it is undesirable that ownership of a secret key is limited to a single party only. Instead, the ownership (i.e., knowledge) of a secret key needs to be distributed between a number of parties.

Threshold cryptography, or more generally group-oriented cryptography, comprises techniques to distribute basic cryptographic schemes between a number of parties. For example, in a threshold version of a digital signature scheme the private key is shared between ten parties, such that each subset of seven parties (or more) is able to issue signatures, while subsets of six parties (or less) cannot produce valid signatures.

6.1 SECRET SHARING

Secret sharing schemes form the basis for threshold cryptography. The idea is to split a secret into several shares, such that the secret can be reconstructed whenever a sufficient number of shares is available; if an insufficient number of shares is available, it should not be possible to reconstruct the secret, nor any part of it.

In constructing secret sharing schemes one should be aware of certain pitfalls, as demonstrated in the following example.

EXAMPLE 6.1 *Consider the RSA cryptosystem with public exponent $e = 3$ and modulus m, $\gcd(e, \phi(m)) = 1$. Two persons like to split the private key $d = 1/e \bmod \phi(m)$ into two halves such that both halves are required to recover d. What about splitting d into its most-significant half and its least-significant half?*

Let $m = pq$, with distinct primes $p, q > 3$ of the same bit length. Then $3d = 1 + l\phi(m)$ for some integer l. Since $0 < d < \phi(m)$ it follows that $l = 1$ or $l = 2$. Since p and q are not divisible by 3, $\phi(m) \not\equiv 2 \pmod 3$, it follows that $l \equiv 2 \pmod 3$. Hence $l = 2$. Consequently, $d = (1 + 2\phi(m))/3$, which we may approximate by

$$\widehat{d} = \left\lfloor \frac{1 + 2(m - 2\sqrt{m} + 1)}{3} \right\rfloor .$$

The approximation error is equal to

$$\widehat{d} - d = \left\lfloor \frac{1 + 2(m - 2\sqrt{m} + 1)}{3} \right\rfloor - \frac{1 + 2\phi(m)}{3} = \left\lfloor \frac{2}{3}(p + q - 2\sqrt{m}) \right\rfloor .$$

Since $p+q > 2\sqrt{m}$ (which follows from $(\sqrt{p}-\sqrt{q})^2 > 0$), we have

$$0 \leq \widehat{d} - d < \sqrt{m},$$

using that w.l.o.g. $\sqrt{m/2} < p < \sqrt{m} < q < \sqrt{2m}$ (as p and q are of equal bit length). But this means that the most-significant half of d is equal to the most-significant half of $\widehat{d}$. (See also this Mathematica notebook.)

Thus, the person receiving the least-significant half of d is able to construct all of d's bits!

The example shows that breaking a bit string into two (or more) pieces is, in general, not a secure way to share a secret. Another problem with this approach is that it does not cover the case of a bit string of length one. How do we "split the bit"?

The critical step in "splitting the bit" is to use some additional randomness. To split a secret bit $s \in \{0,1\}$ into two shares, we choose an additional bit $u \in_R \{0,1\}$ and set as shares $s_1 = s \oplus u$ and $s_2 = u$. Then neither of the shares s_1, s_2 on its own reveals any information on s, but together s_1 and s_2 completely determine s, as $s_1 \oplus s_2 = s \oplus u \oplus u = s$.

EXERCISE 6.2 Suppose u is uniformly distributed. Show that s_1 and s_2 are also uniformly distributed, *irrespective* of the distribution of s.

As an immediate generalization, we may split a bit s into ℓ shares $s_1, \ldots, s_\ell$, by picking $\ell - 1$ random bits $u_2, \ldots, u_\ell \in_R \{0,1\}$ and setting as shares $s_1 = s \oplus u_2 \oplus \cdots \oplus u_\ell$, and $s_2 = u_2$, $\ldots$, $s_\ell = u_\ell$. The secret may then be reconstructed from all shares $s_1, \ldots, s_\ell$ by computing $s = s_1 \oplus \cdots \oplus s_\ell$. Less than ℓ shares do not yield any information on the secret s. In general, though, it is not required that the secret can be reconstructed only when *all* shares are available.

DEFINITION 6.3 *A **secret sharing scheme** for a **dealer** $\mathcal{D}$ and **participants** $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$ comprises the following two protocols.*

Distribution. *A protocol in which dealer $\mathcal{D}$ shares a secret s such that each participant $\mathcal{P}_i$ obtains a share s_i, $1 \leq i \leq \ell$.*

Reconstruction. *A protocol in which secret s is recovered by pooling shares s_i, $i \in Q$, of any* qualified *set of participants $Q \subseteq \{\mathcal{P}_1, \ldots, \mathcal{P}_\ell\}$.*

The set Γ of all qualified (or, authorized) subsets of $\{\mathcal{P}_1, \ldots, \mathcal{P}_\ell\}$ is called the **access structure**. More formally, access structure Γ is an element of the *powerset* of $\{\mathcal{P}_1, \ldots, \mathcal{P}_\ell\}$, or in symbols, $\Gamma \in 2^{\{\mathcal{P}_1, \ldots, \mathcal{P}_\ell\}}$, where $2^X = \{A : A \subseteq X\}$ for a set X. Usually, an access structure Γ is **monotone**, which means that Γ is closed under taking supersets: if $A \in \Gamma$ and $A \subseteq B$, then also $B \in \Gamma$, for all $A, B \subseteq \{\mathcal{P}_1, \ldots, \mathcal{P}_\ell\}$.

The following security requirements are imposed on a secret sharing scheme:

(i) any qualified set of participants is able to determine the value of s by pooling their shares, and

(ii) any non-qualified set of participants cannot determine *any* information on the value of s when pooling their shares.

Secret sharing schemes satisfying these requirements are called **perfect**.[1]

For the purpose of threshold cryptography, we will be concerned with (t, ℓ)-**threshold** secret sharing schemes only, where the access structure is defined as $\Gamma = \{Q \subseteq \{\mathcal{P}_1, \ldots, \mathcal{P}_\ell\} : |Q| \geq t\}$, $1 \leq t \leq \ell$. In a (t, ℓ)-threshold scheme, any group of t participants is able to recover the secret, but no group of $t-1$ or less participants can do so.

6.1.1 SHAMIR THRESHOLD SCHEME

Shamir proposed a simple and elegant (t, ℓ)-threshold secret sharing scheme, $1 \leq t \leq \ell$. The scheme can be applied whenever the secret belongs to a finite field $\mathbb{F}_q$ of order q, where $q > \ell$. For simplicity, however, we will describe Shamir's scheme for the case $q = p$ only, where p is prime, since this allows us to treat the elements of $\mathbb{F}_q = \mathbb{Z}_p$ as integers modulo p.

Shamir's (t, ℓ)-threshold scheme for sharing a secret $s \in \mathbb{Z}_p$ is defined as follows.

Distribution. The dealer picks a random polynomial $a(X) \in_R \mathbb{Z}_p[X]$ of degree less than t satisfying $a(0) = s$. It sends share $s_i = a(i)$ to participant $\mathcal{P}_i$, for $i = 1, \ldots, \ell$.

Reconstruction. Any set Q of t participants may recover secret s from their shares by Lagrange interpolation:[2]

$$s = \sum_{i \in Q} s_i \lambda_{Q,i}, \quad \text{with } \lambda_{Q,i} = \prod_{j \in Q \setminus \{i\}} \frac{j}{j-i}.$$

To see why reconstruction works, recall that the Lagrange interpolation formula for the *unique* polynomial $a(X)$ of degree less than t passing through the points (i, s_i), $i \in Q$, is given by

$$a(X) = \sum_{i \in Q} s_i \prod_{j \in Q \setminus \{i\}} \frac{X-j}{i-j}.$$

Since we are interested in the constant term $s = a(0)$ only, we may substitute 0 for X.

Next, we need to argue that non-qualified sets of participants cannot find the secret s. Suppose that participants $\mathcal{P}_i$ pool their shares s_i, $i \in A$, where $|A| = t-1$. Fix any (hypothetical) value $\tilde{s} \in \mathbb{Z}_p$ for the secret, as used by the dealer. Lagrange interpolation of the points $(0, \tilde{s})$ and (i, s_i), $i \in A$, implies that there exists a *unique* polynomial $\tilde{a}(X)$ of degree less than t passing through these points. In other words, given shares s_i, $i \in A$, any value $\tilde{s}$ for the secret is equally probable, which means that from these shares no information on the secret s can be gained. Therefore, the scheme is perfect.

Note that the security of Shamir's scheme does not depend on the size of p. The only condition on p is that $p > \ell$ holds.

6.2 VERIFIABLE SECRET SHARING

A basic secret sharing scheme is defined to resist passive attacks only, which means that its security depends on the assumption that all parties involved run the protocols as prescribed

[1] If non-qualified sets of participants are able to determine some (partial) information on secret s then the scheme is not perfect. Without going into detail, we mention that such schemes are also of use.

[2] Note that we frequently write $i \in Q$ as a shorthand for $\mathcal{P}_i \in Q$.

by the scheme. After (honestly) taking part in the distribution protocol, a non-qualified set of participants is not able to deduce any information on the secret.

In many applications, however, a secret sharing scheme is also required to withstand active attacks. This is accomplished by a **verifiable secret sharing** (VSS) scheme, which is designed to resist (combinations of) the following two types of active attacks:

- a dealer sending incorrect shares to some or all of the participants during the distribution protocol, and
- participants submitting incorrect shares during the reconstruction protocol.

Clearly, Shamir's scheme is not a VSS scheme, since it does not exclude either of these active attacks. During the distribution protocol, there is no guarantee that the shares s_i received actually correspond to a single polynomial $a(X)$ of degree less than t. Similarly, during the reconstruction protocol, there is no guarantee that a share s_i provided by participant $\mathcal{P}_i$ is actually equal to the share received by $\mathcal{P}_i$ during the distribution protocol: nothing prevents $\mathcal{P}_i$ from using a random value $\widetilde{s_i} \in_R \mathbb{Z}_p$ instead of the correct value s_i; the reconstructed value $\tilde{s}$ will be useless, but if $\mathcal{P}_i$ is the only cheating participant during reconstruction, $\mathcal{P}_i$ will still be able to find the value of s using the other $t-1$ correct shares.

We will consider two basic VSS schemes.

6.2.1 FELDMAN VSS

The idea behind Feldman's VSS scheme is to let the dealer send additional values to *all* participants based on which each share can be checked for validity. The scheme assumes a DL setting.

Let $\langle g \rangle$ be a group of order n, where n is a large prime. Feldman's (t, ℓ)-threshold VSS scheme for sharing a secret $s \in \mathbb{Z}_n$ is defined as an extension of Shamir's scheme.

Distribution. The dealer chooses a random polynomial of the form

$$a(X) = s + u_1 X + \cdots + u_{t-1} X^{t-1},$$

where $u_j \in_R \mathbb{Z}_n$, $1 \leq j < t$. The dealer sends shares $s_i = a(i)$ to participant $\mathcal{P}_i$ in private, for $i = 1, \ldots, \ell$. Set $u_0 = s$. In addition, the dealer broadcasts commitments $B_j = g^{u_j}$, $0 \leq j < t$. Upon receipt of share s_i, participant $\mathcal{P}_i$ verifies its validity by evaluating the following equation:

$$g^{s_i} = \prod_{j=0}^{t-1} B_j^{i^j}. \tag{6.1}$$

Reconstruction. Each share s_i contributed by participant $\mathcal{P}_i$ is verified using Eq. (6.1). The secret $s = a(0)$ is then recovered as in Shamir's scheme from t valid shares.

The commitments B_j broadcast by the dealer, commit the dealer to a single polynomial $a(X)$ of degree less than t over $\mathbb{Z}_n$. If $s_i = a(i)$, then Eq. (6.1) will indeed hold:

$$g^{s_i} = g^{a(i)} = g^{\sum_{j=0}^{t-1} u_j i^j} = \prod_{j=0}^{t-1} g^{u_j i^j} = \prod_{j=0}^{t-1} B_j^{i^j}.$$

Therefore, any attempts at cheating by the dealer or by one or more participants is detected. This security property does not depend on the DL assumption (or any other computational assumption).

To complete the security analysis of Feldman's scheme, however, we need to argue that any set of $t-1$ participants is not able to find the secret from their shares, given the fact that they also get to see the commitments B_j, $j = 0, \ldots, t-1$. In particular, note that $B_0 = g^{u_0} = g^s$ is available, hence it is possible to find s if one is able to compute discrete logs w.r.t. g. Thus, we will prove that if the DL assumption holds for $\langle g \rangle$, it follows that $t-1$ or less participants are not able to find the secret s.

Let $h \in \langle g \rangle$ be given, such that $\log_g h$ is unknown. Suppose w.l.o.g. that participants $\mathcal{P}_1, \ldots, \mathcal{P}_{t-1}$ (hence, collectively forming the adversary) are able to find the secret s. We show how to compute $\log_g h$, which contradicts the DL assumption.

Given h we construct a modified instance of Feldman's VSS as follows. The distribution protocol is modified by letting the dealer set $B_0 = h$ (which means that the secret is $s = \log_g h$ is not known to the dealer). The dealer also chooses $s_1, \ldots, s_{t-1} \in_R \mathbb{Z}_n$, and computes B_j for $j = 1, \ldots, t-1$ such that Eq. (6.1) holds for participants $\mathcal{P}_1, \ldots, \mathcal{P}_{t-1}$. (The shares for participants $\mathcal{P}_t, \ldots, \mathcal{P}_\ell$ are irrelevant.)

It is essential that the dealer is able to compute B_j for $j = 1, \ldots, t-1$ without knowing $s = \log_g h$ by setting:

$$B_j = \prod_{k=1}^{t-1} (g^{s_k}/h)^{\gamma_{j,k}}. \tag{6.2}$$

Here $(t-1) \times (t-1)$ matrix $(\gamma_{j,k})$ is the inverse of the following Vandermonde matrix:

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ 2 & 2^2 & \cdots & 2^{t-1} \\ \vdots & \vdots & & \vdots \\ t-1 & (t-1)^2 & \cdots & (t-1)^{t-1} \end{pmatrix}.$$

EXERCISE 6.4 Verify that Eq. (6.1) holds for $1 \leq i < t$ if $B_0 = h$ and B_j, $1 \leq j < t$, are defined by (6.2).

The view of participants $\mathcal{P}_1, \ldots, \mathcal{P}_{t-1}$ in the above modified instance of Feldman's scheme is identical to their view in a regular instance of the scheme. Therefore, any successful attack on the regular instances will be equally successful on the modified instances. However, in the modified instances the dealer itself does not even know the secret s, and breaking a modified instance actually means computing the "embedded" discrete logarithm $\log_g h$. Any successful attack on Feldman's scheme, recovering the secret from $t-1$ shares only, would thus contradict the DL assumption.

EXERCISE 6.5 The special case of an (ℓ, ℓ)-threshold scheme for secrets $s \in \mathbb{Z}_n$ can be solved simply by setting the shares as follows: choose $s_i \in_R \mathbb{Z}_n$ for $i = 2, \ldots, \ell$ and set $s_1 = (s - \sum_{i=2}^{\ell} s_i) \bmod n$. Extend this basic secret sharing scheme to a Feldman VSS scheme, and provide a security analysis of the resulting VSS scheme.

6.2.2 PEDERSEN VSS

There is one caveat for Feldman's scheme which we have ignored so far. We have tacitly assumed that the *a priori* distribution of the secret s used by the dealer is the uniform distribution on $\mathbb{Z}_n$. In some applications, however, the *a priori* distribution of s may be very skewed. For example, it may already be known that the secret is actually a bit, hence that $\Pr[s = v] = 0$ for all $v \in \mathbb{Z}_n \setminus \{0, 1\}$.

Pedersen's VSS scheme allows one to share a secret s in a verifiable way such that the security does not depend on the *a priori* distribution of s. In fact, the secret s will remain hidden without relying on a discrete log assumption. Hence, the secrecy of s is guaranteed in an information-theoretic way, just as for the basic Shamir scheme. The critical difference compared to Feldman's scheme is to use Pedersen commitments (see Section 3.2.2) for the commitments broadcast by the dealer.

The DL assumption is still relevant, however, since we need it to show that the dealer cannot cheat by sending inconsistent shares to the participants.

Let $\langle g \rangle$ be a group of order n, where n is a large prime. Let $h \in_R G \setminus \{1\}$ denote a random group element (such that $\log_g h$ is not known to any party). Pedersen's (t, ℓ)-threshold VSS scheme for sharing a secret $s \in \mathbb{Z}_n$ is defined as the following modification of Feldman's scheme.

Distribution. The dealer chooses random polynomials $a(X), b(X)$ of the form

$$a(X) = u_0 + u_1 X + \cdots + u_{t-1} X^{t-1}$$
$$b(X) = v_0 + v_1 X + \cdots + v_{t-1} X^{t-1},$$

where $u_j, v_j \in_R \mathbb{Z}_n$, $0 \le j < t$, subject to the condition $a(0) = s$. The dealer sends shares $s_i = (a(i), b(i))$ to participant $\mathcal{P}_i$ in private, for $i = 1, \ldots, \ell$. In addition, the dealer broadcasts commitments $C_j = g^{u_j} h^{v_j}$, $0 \le j < t$. Upon receipt of share $s_i = (s_{i1}, s_{i2})$, participant $\mathcal{P}_i$ verifies its by evaluating the following equation:

$$g^{s_{i1}} h^{s_{i2}} = \prod_{j=0}^{t-1} C_j^{i^j}. \tag{6.3}$$

Reconstruction. Each share s_i contributed by participant $\mathcal{P}_i$ is verified using Eq. (6.3). The secret $s = a(0)$ is then recovered as in Shamir's scheme from t valid shares.

Informally, the security of Pedersen's VSS scheme is analyzed as follows. Using that $u_0 = s$, note that the dealer broadcasts commitment $C_0 = g^s h^{v_0}$. Since this is a Pedersen commitment, it follows that the secret s is hidden in an information-theoretic way. This line of reasoning can be extended to show that even when $t - 1$ shares are known in addition to the commitments C_j, $0 \le j < t$, nothing can be deduced about the value of s (other than what already follows from the *a priori* distribution of s).

A technical detail is that the dealer would be able to cheat if it would know $\log_g h$. The fact that Pedersen's commitment scheme is computationally binding (under the DL assumption), however, ensures that the dealer is committed to the polynomials $a(X)$ and $b(X)$ once it broadcasts the C_j values.

EXERCISE 6.6 See Exercise 6.5. This time extend the basic (ℓ, ℓ)-threshold scheme with shares $s_i \in_R \mathbb{Z}_n$ for $i = 2, \ldots, \ell$ and $s_1 = (s - \sum_{i=2}^{\ell} s_i) \bmod n$ to a Pedersen VSS scheme, and provide a security analysis of the resulting VSS scheme.

6.3 THRESHOLD CRYPTOSYSTEMS

In a basic public-key cryptosystem the private key is held by a single party. The object of a (t, ℓ)-threshold cryptosystem is to distribute the knowledge of a private key between parties $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$ such that at least t of these parties are required for successful decryption, $1 \leq t \leq \ell$. As a concrete example we will consider a (t, ℓ)-threshold version of the ElGamal cryptosystem.

Recall from Section 2.1.4 that the basic ElGamal cryptosystem consists of a key generation algorithm, an encryption algorithm, and a decryption algorithm. To appreciate the definition of a threshold cryptosystem (Definition 6.7), we first consider a simple but flawed approach for obtaining a threshold version of the ElGamal cryptosystem.

The idea is to incorporate a (verifiable) secret sharing scheme as follows. A dealer first runs the key generation algorithm of the ElGamal cryptosystem, resulting in a private key x and a public key h, say. Subsequently, the dealer runs the distribution protocol of a (t, ℓ)-threshold secret sharing scheme, using x as the secret (e.g., using Feldman's VSS scheme). As a result, party $\mathcal{P}_i$ gets a share x_i of the private key. The encryption algorithm remains the same, using public key h. Finally, for decryption, the reconstruction protocol of the secret sharing scheme is run first to obtain the private key x, which is then used as input to the decryption algorithm.

However, this simple method suffers from two major drawbacks. Firstly, the dealer gets to know the private key x, hence the dealer must be trusted not to use x on its own. Secondly, during decryption the private key x is reconstructed, hence the parties involved must be trusted not to use x on their own as well.

To address these problems, a threshold cryptosystem is defined as follows.

DEFINITION 6.7 *A (t, ℓ)-**threshold cryptosystem**, $1 \leq t \leq \ell$, is a scheme for parties $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$ consisting of the following three components.*

Distributed key generation. *A protocol between $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$ for generating a public key h such that party $\mathcal{P}_i$ obtains a private share x_i (of the private key x corresponding to h) and a public **verification key** h_i, $1 \leq i \leq \ell$. The protocol depends on t.*

Encryption. *An algorithm that on input of a plaintext M, a public key h, outputs a ciphertext C of M under public key h.*

Threshold decryption. *A protocol between any set of t parties $\mathcal{P}_{i_1}, \ldots, \mathcal{P}_{i_t}$ that on input of a ciphertext C, private shares $x_{i_1}, \ldots, x_{i_t}$, and verification keys $h_{i_1}, \ldots, h_{i_t}$, outputs plaintext M.*

A basic security requirement for a threshold cryptosystem is that the private key x remains secret at all times, unless t or more parties decide to cheat by pooling their shares. Therefore, the distributed key generation (DKG) protocol is symmetric with respect to the roles of $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$, hence the DKG protocol does not rely on a special party acting as a (trusted) dealer in a secret sharing scheme. Similarly, the threshold decryption protocol should be such that it does not rely on reconstructing the private key x.

EXERCISE 6.8 See Section 2.1.4. Design an (ℓ, ℓ)-threshold ElGamal cryptosystem (for plaintexts in $\langle g \rangle$) with public key $h = \prod_{i=1}^{\ell} h_i$, where $x_i \in \mathbb{Z}_n$ is the private share of party $\mathcal{P}_i$ and $h_i = g^{x_i}$ is the corresponding verification key, $1 \leq i \leq \ell$, and discuss its security.

6.3.1 THRESHOLD ELGAMAL CRYPTOSYSTEM

As usual let $\langle g \rangle$ be a group of order n, where n is a large prime. In the basic ElGamal cryptosystem, a public key is of the form $h = g^x$, where $x \in_R \mathbb{Z}_n$ denotes the private key. For a (t, ℓ)-threshold ElGamal cryptosystem, a public key will be of the form $h = g^{a(0)}$, where $a(X) \in \mathbb{Z}_n[X]$ is a random polynomial of degree less than t. Each party $\mathcal{P}_i$ gets a private share $x_i = a(i)$, $1 \leq i \leq \ell$, hence $a(X)$ is the unique polynomial passing through the points $(1, x_1), \ldots, (\ell, x_\ell)$. Furthermore, the corresponding verification keys are defined as $h_i = g^{x_i}$.

Distributed Key Generation Protocol

The object of the DKG protocol is to let parties $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$ jointly generate the random polynomial $a(X)$. We will do so by having each party $\mathcal{P}_i$, $1 \leq i \leq \ell$, pick a random polynomial $a_i(X) \in \mathbb{Z}_n[X]$ of degree less than t and then defining $a(X) = \sum_{i=1}^{\ell} a_i(X)$. Feldman's VSS is used to share these polynomials between the parties. For simplicity we assume that each party behaves honestly (the protocol is able to identify cheating parties, but it is a bit cumbersome to describe how the cheating parties are to be eliminated).

The DKG protocol then consists of the following steps, using an auxiliary commitment scheme:

1. Each party $\mathcal{P}_i$ picks a random polynomial $a_i(X) \in \mathbb{Z}_n[X]$ of degree less than t, and broadcasts a commitment to the value of g^{s_i}, where $s_i = a_i(0)$.

2. Each party $\mathcal{P}_i$ opens its commitment to g^{s_i} and the public key h is set as $h = g^{\sum_{i=1}^{\ell} s_i}$.

3. Each party $\mathcal{P}_i$ runs an instance of Feldman's VSS scheme, using $s_i \in \mathbb{Z}_n$ as secret value. Party $\mathcal{P}_i$ plays the role of the dealer, and parties $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$ play the role of the participants. (Hence, $\mathcal{P}_i$ plays a double role, namely as the dealer and as a participant.)

4. Let s_{ij} denote the share of s_i as sent by party $\mathcal{P}_i$ to party $\mathcal{P}_j$, for $1 \leq i, j \leq \ell$. Each party $\mathcal{P}_i$ sums all its received shares s_{ji} to obtain its share $x_i = \sum_{j=1}^{\ell} s_{ji}$ of the private key x. The verification key of party $\mathcal{P}_i$ is defined as $h_i = g^{x_i}$.

Note that $s_{ji} = a_j(i)$. Since $a(X) = \sum_{i=1}^{\ell} a_i(X)$, it follows that $x_i = a(i)$.

The use of the auxiliary commitment scheme in the first two steps of the protocol ensures that no party is able to influence the value of the public key h, other than by contributing a random share s_i.

Threshold Decryption Protocol

Let $C = (A, B)$ be an ElGamal ciphertext for public key h. The threshold decryption protocol consists of the following two steps:

1. Each party $\mathcal{P}_i$ takes A as input and uses its share x_i to produce a value $d_i = A^{x_i}$ along with a Σ-proof showing that $\log_g h_i = \log_A d_i$.

2. Let Q be a set of t parties who produced correct d_i values. Then the plaintext M can be recovered by evaluating:

$$B / \prod_{i \in Q} d_i^{\lambda_{Q,i}} = B / A^x = M,$$

where $\lambda_{Q,i} = \prod_{j \in Q \setminus \{i\}} \frac{j}{j-i}$ denote Lagrange coefficients as in Shamir's scheme.

As long as t or more parties produce a correct decryption share d_i, decryption succeeds. Hence, the protocol tolerates at most $\ell - t$ faulty parties, who are unable or unwilling to participate in the decryption of a given ciphertext.

6.4 BIBLIOGRAPHIC NOTES

The observation that for small e the RSA cryptosystem leaks the most-significant half of the private key d can be traced back to [BDF98, p. 29]. Example 6.1 builds on this idea, using a slightly better approximation $\widehat{d}$ and optimizing for the case $e = 3$.

The Shamir threshold secret sharing scheme is from [Sha79]. Independently, Blakley discovered another way to do threshold secret sharing [Bla79]. Threshold cryptography was introduced as a new concept together with some first solutions in [Des88, DF90], see also [Des94]. The Feldman VSS and Pedersen VSS schemes are from [Fel87] and [Ped92], respectively. The threshold ElGamal cryptosystem is from [Ped91].

CHAPTER 7

Secure Multiparty Computation

Imagine a constellation of parties $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$ each holding a value $x_1, \ldots, x_\ell$, respectively, for which they like to evaluate the function value $f(x_1, \ldots, x_\ell)$ for some given function f. The problem of secure multiparty computation is to find a protocol for $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$ which enables them to jointly compute output value $f(x_1, \ldots, x_\ell)$, however in such a way that their respective input values $x_1, \ldots, x_\ell$ remain secret, except for the information that can be inferred logically from the output value.

In case $\ell = 2$, a classical example is Yao's millionaires problem. Parties $\mathcal{P}_1$ and $\mathcal{P}_2$ are two millionaires (or, billionaires by today's standards) who want to compare their wealth, hence to see who is the richer one. That is, writing x_1, x_2 for their respective wealths, they want to evaluate the function $f(x_1, x_2) = x_1 > x_2$ (we do not care about the improbable case that $x_1 = x_2$). They could simply do so by telling each other the values of x_1 and x_2 but obviously this way much more information than the value of $x_1 > x_2$ is revealed. What they need is a protocol for evaluating the value of $x_1 > x_2$ without leaking any further information on x_1 and x_2.

The beauty of the theory of secure multiparty computation is that a protocol for evaluating a given function f securely can be found, as long as f is an *efficiently computable* function. In this chapter we first cover electronic voting as a simple case of multiparty computation, corresponding to function $f(x_1, \ldots, x_\ell) = x_1 + \cdots + x_\ell$. Subsequently, we will consider the problem for general functions f, such as $f(x_1, \ldots, x_\ell) = x_1 \cdot \cdots \cdot x_\ell$ and $f(x_1, \ldots, x_\ell) = \max(x_1, \ldots, x_\ell)$, which are harder to handle.

7.1 ELECTRONIC VOTING

The problem of finding a secure and efficient electronic voting scheme has been a challenge since the beginning of the 1980s. Assuming that votes are binary (that is, either "yes" or "no"), electronic voting can be viewed as an instance of a secure multiparty computation, where the function to be evaluated is $f(x_1, \ldots, x_\ell) = x_1 + \cdots + x_\ell$ for votes $x_1, \ldots, x_\ell \in \{0, 1\}$. Given the techniques developed in the preceding chapters it is possible to arrive at a practical solution in just a few steps.

Let us briefly state the basic security requirements for an electronic voting scheme.

- **Eligibility** means that only eligible voters can cast a vote, and also that each eligible voter can cast at most one vote.

- **Privacy** of an individual vote is assured against any reasonably sized coalition of parties (not including the voter herself). Depending on the implementation some cryptographic assumptions need to be assumed as well.
- **Universal Verifiability** ensures that any party, including a passive observer, can check that the election is fair, i.e., that the published final tally is computed fairly from the ballots that were correctly cast.
- **Robustness** (or, fault-tolerance) means that the faulty behavior (either benign or malicious) of any reasonably sized coalition of participants can be tolerated. In large-scale elections this means that no coalition of voters of any size can disrupt the election; in other words, any cheating voter can be detected and discarded.

We will formulate a solution in terms of two types of roles: voters and talliers. Let $\mathcal{V}_1, \ldots, \mathcal{V}_{\ell'}$ denote the voters and $\mathcal{T}_1, \ldots, \mathcal{T}_\ell$ denote the talliers taking part in the election. The role of a voter is simply to cast a vote as an encrypted and authenticated message. The talliers will take care of computing the final tally (i.e., the sum of the votes), however, without compromising the privacy of the votes. The problem is to resolve the apparent contradiction that in order to compute the sum of the votes, the talliers need to decrypt the individual votes thereby compromising privacy.

A single party (person or entity) may take part as a voter, as a tallier, or as both a voter and a tallier. Two typical cases are large-scale elections with $\ell' \gg \ell$ and boardroom elections with $\ell' = \ell$. In large-scale elections the number of voters may range from 100 to 1,000,000 say, while the number of talliers is limited, e.g., between 5 and 50. In a boardroom election every person plays the role of both voter and tallier. The voting scheme below is suited for large-scale elections, limiting the work for voters to sending a single message.

For the solution presented below, the main tool is a **threshold homomorphic cryptosystem**. As a concrete example of such a cryptosystem, we will consider the threshold homomorphic ElGamal cryptosystem. This version of the ElGamal cryptosystem is the same as the threshold ElGamal cryptosystem of Section 6.3.1 with the modification that the plaintext space is $\mathbb{Z}_n$ instead of $\langle g \rangle$, where encryption of a plaintext $M \in \mathbb{Z}_n$ under public key h is defined as

$$(A, B) = (g^u, h^u g^M), \qquad u \in_R \mathbb{Z}_n.$$

In other words, a value $M \in \mathbb{Z}_n$ is *encoded* as a value $g^M \in \langle g \rangle$. Note that during decryption we need to apply the reverse transformation, that is given g^M for some $M \in \mathbb{Z}_n$, we need to compute M. In general, we would need to solve the discrete log problem to do so, which we assume to be infeasible. The way out is that we will see to it that M belongs to a small subset of $\mathbb{Z}_n$.

This modified ElGamal cryptosystem then enjoys the following (additive) **homomorphic property**. If we multiply an encryption (A, B) of M with an encryption (A', B') of M' we obtain an encryption of $M + M'$:

$$(A, B) * (A', B') = (AA', BB') = (g^{u+u'}, h^{u+u'} g^{M+M'}).$$

The electronic voting scheme is now as follows.

Key generation. The talliers $\mathcal{T}_1, \ldots, \mathcal{T}_\ell$ run the DKG protocol of the (t, ℓ)-threshold ElGamal cryptosystem, for an agreed upon value of t. Let h denote the resulting public key.

Voting. Each voter $\mathcal{V}_i$ casts a vote $v_i \in \{0,1\} \simeq \{\text{“no”}, \text{“yes”}\}$ by broadcasting a **ballot**[1] which is a message consisting of an ElGamal encryption $(A_i, B_i) = (g^{u_i}, h^{u_i} g^{v_i})$, where $u_i \in_R \mathbb{Z}_n$, and a Σ-proof that (A_i, B_i) is correctly formed.

Tallying. The talliers decrypt the product $(A, B) = \prod_{i=1}^{\ell'} (A_i, B_i)$ to obtain $g^{\sum_{i=1}^{\ell'} v_i}$ as intermediate result, from which $\sum_{i=1}^{\ell'} v_i$ is easily determined using that $0 \leq \sum_{i=1}^{\ell'} v_i \leq \ell'$.[2]

If all (A_i, B_i) are correctly formed, it follows from the homomorphic property that $(A, B) = (g^u, h^u g^{\sum_{i=1}^{\ell'} v_i})$ for some $u \in \mathbb{Z}_n$, which ensures the validity of the final tally.

EXERCISE 7.1 Provide the Σ-protocol for proving that (A_i, B_i) is correctly formed, and turn it into a Σ-proof (see Section 5.4.2 for a similar case).

In practice, voter $\mathcal{V}_i$ needs to authenticate its ballot, say by producing a digital signature on (A_i, B_i) and the accompanying proof. The officials running the voting scheme must know the public keys of the voters. During an election, the officials will check the signature on each submitted ballot. At most one ballot will be accepted and recorded for each voter.

The property of universal verifiability is that anyone is able to check that (i) all ballots (A_i, B_i) used to calculate the product (A, B) are correctly formed (by checking the accompanying proofs), (ii) each ballot (A_i, B_i) is correctly signed w.r.t. the public key of voter $\mathcal{V}_i$, (iii) product (A, B) is computed correctly, (iv) each tallier produced a correct share of the decryption of (A, B) (by checking the proofs, see Section 6.3.1), and finally (v) that the final tally corresponds to the decrypted value of (A, B).

EXERCISE 7.2 The topic of this exercise is a boardroom election scheme involving voters (doubling as talliers) $\mathcal{V}_1, \ldots, \mathcal{V}_\ell$, $\ell \geq 1$. Each voter $\mathcal{V}_i$ has a public key $h_i = g^{x_i}$, where $x_i \in_R \mathbb{Z}_n$ is $\mathcal{V}_i$'s private key. Let $H_i = h_\ell \prod_{j=1}^{i-1} h_j$, for $1 \leq i \leq \ell$.

First, voter $\mathcal{V}_\ell$ publishes the following encryption of its vote $v_\ell \in \{0,1\}$:

$$(A_\ell, B_\ell) = (g^{u_\ell}, H_\ell^{u_\ell} g^{v_\ell}),$$

where $u_\ell \in_R \mathbb{Z}_n$. Next, for $i = \ell - 1, \ldots, 1$ (in this order), voter $\mathcal{V}_i$ publishes the following encryption of its vote $v_i \in \{0,1\}$:

$$(A_i, B_i) = (A_{i+1} g^{u_i}, B_{i+1} A_{i+1}^{-x_i} H_i^{u_i} g^{v_i}),$$

where $u_i \in_R \mathbb{Z}_n$. Finally, voter $\mathcal{V}_\ell$ publishes $\log_g B_1 A_1^{-x_\ell}$.

Let $t_i = \sum_{j=i}^{\ell} v_j$, for $1 \leq i \leq \ell$. (i) Prove by induction on i that (A_i, B_i) is an ElGamal encryption of g^{t_i} under public key H_i. Hence, that $\mathcal{V}_\ell$ publishes $\sum_{j=1}^{\ell} v_j$ at the end of the protocol. (ii) Show how $\mathcal{V}_\ell, \mathcal{V}_1, \ldots, \mathcal{V}_{i-1}$ for any i, $2 \leq i \leq \ell$, are jointly able to decrypt (A_i, B_i), hence that they are able to determine the *intermediate* election result t_i. (iii) Describe the relations that need to be proved in each protocol step to show that the voter's output is formed correctly.

[1] The word “ballot” derives from the Italian “ballotta.” Small colored balls (like peas or beans) were first used for secret voting in Italy in mid 16th century. Here, we use “ballot” as a shorthand for “voted ballot,” which literally means “a sheet of paper filled out to cast a secret vote.”

[2] Given $h = g^x$, where $0 \leq x \leq \ell' < n$, the Pollard-λ (kangaroo) method finds x in $O(\sqrt{\ell'})$ time using $O(1)$ storage.

x	y	xy
0	0	0
1	0	0
0	1	0
1	1	1

$\cong$

Alice	Bob	match?
no	no	-
yes	no	-
no	yes	-
yes	yes	♡

FIGURE 7.1: Matching without embarrassments

7.2 BASED ON THRESHOLD HOMOMORPHIC CRYPTOSYSTEMS

The homomorphic ElGamal cryptosystem introduced in the previous section provides an easy way to compute an encryption of $x + y$ (modulo n), given encryptions of x and y. We will express this fact in a somewhat abstract way by saying that given homomorphic encryptions $E(x)$ and $E(y)$, we may compute the encryption $E(x + y) = E(x)E(y)$. Here, $E(x)$ stands for a homomorphic ElGamal encryption of $x \in \mathbb{Z}_n$ under some understood, fixed public key h; hence, $E(x) = (g^u, h^u g^x)$ for some $u \in_R \mathbb{Z}_n$.

So, addition is easily covered. What about multiplication, that is, computing $E(xy)$ from $E(x)$ and $E(y)$? This is indeed an important question as a solution to this problem basically implies that we would be able to compute any function $f : \mathbb{Z}_n \times \mathbb{Z}_n \to \mathbb{Z}_n$. To compute $E(f(x, y))$ from $E(x)$ and $E(y)$ we express $f(x, y)$ as a polynomial in $\mathbb{Z}_n[x, y]$, and repeatedly apply addition and multiplication to evaluate the polynomial. For example, to compute $E((x+y)^2)$ we first use addition to compute $E(x+y)$ and then use multiplication to compute $E((x + y)(x + y))$. Without going into details, we state that a solution for multiplication allows us, in principle, to handle any efficiently computable function.

We will now focus on the computation of $E(xy)$ given $E(x)$ and $E(y)$, considering the special case that $x, y \in \{0, 1\}$. Also, we restrict ourselves to the case of two-party computation ($\ell = 2$), writing $\mathcal{A}$ and $\mathcal{B}$ for parties $\mathcal{P}_1$ and $\mathcal{P}_2$. Despite these restrictions the problem of computing $E(xy)$ is still non-trivial.

We first consider the case where party $\mathcal{A}$ knows x and party $\mathcal{B}$ knows y. In other words, x is the private input of $\mathcal{A}$ and y is the private input of $\mathcal{B}$. Securely computing xy in this case may be interpreted as a way to implement the ultimate dating service, as may be concluded from Figure 7.1. The idea is as follows. Say, Alice and Bob like to find out if they want to go on a date (with each other). They might simply tell each other whether they are interested or not. In case they are both interested, this simple solution is satisfactory. However, if for example Alice is not interested in Bob but Bob is interested in Alice, Bob might feel embarrassed afterwards because he expressed interest in her; if he would have known beforehand that Alice was not interested anyway, he would have told Alice that he was not interested.

In terms of bits, we see that if $xy = 1$ it follows that $x = y = 1$ and there is nothing to hide. If $xy = 0$, then $x = 0$ and/or $y = 0$. If $x = 0$, then it follows that $xy = 0$ regardless of the value of y; in this case, a secure computation of xy is required to completely hide the value of y. Hence, if Alice is not interested in Bob, and she knows that there is not going to be a match anyway, she should not be able to find out whether Bob was interested or not. The same reasoning applies to the case $y = 0$.

Let $x \in \{0, 1\}$ be the private input of party $\mathcal{A}$, and let $y \in \{0, 1\}$ be the private input of party $\mathcal{B}$. To compute xy securely, the following protocol is executed, assuming that $\mathcal{A}$ and $\mathcal{B}$ have set up a $(2, 2)$-threshold ElGamal cryptosystem with public key h.

1. Party $\mathcal{A}$ sends encryption $(A, B) = (g^u, h^u g^x)$, where $u \in_R \mathbb{Z}_n$ to party $\mathcal{B}$.
2. Party $\mathcal{B}$ raises the encryption to the power y and sends the randomized result $(C, D) = (g^v A^y, h^v B^y)$, where $v \in_R \mathbb{Z}_n$, to party $\mathcal{A}$.
3. Parties $\mathcal{A}$ and $\mathcal{B}$ jointly decrypt (C, D) to obtain the value of xy.

The protocol is described for the passive case, that is, the case in which the parties follow the protocol exactly. If $\mathcal{A}$ and $\mathcal{B}$ follow the protocol, we see that $(C, D) = (g^{v+uy}, h^{v+uy} g^{xy})$, hence decryption of (C, D) indeed results in the value of xy.

To prove the security of the protocol in the passive case, we still need to do some work though. Namely, we must show that party $\mathcal{A}$ is not able to deduce any information on y other than implied by the values of its own input x and the common output xy. The only additional information $\mathcal{A}$ gets on y is the encryption (C, D), but note that $(C, D) = (g^v A^y, h^v B^y)$, where $v \in_R \mathbb{Z}_n$ is only known to party $\mathcal{B}$. Under the DDH assumption, it follows that (C, D) does not give any information on y. Similarly, the only additional information $\mathcal{B}$ gets on x is the encryption (A, B) which gives no information on x, again under the DDH assumption.

To cover the active case, zero-knowledge proofs should be added to enforce parties $\mathcal{A}$ and $\mathcal{B}$ to follow the protocol. In this case, $\mathcal{A}$ is required to prove that (A, B) is an encryption of a bit value, and $\mathcal{B}$ is required to prove that (C, D) is indeed of the form $(g^s A^y, h^s B^y)$ for some $s \in \mathbb{Z}_n$ and some $y \in \{0, 1\}$.

Next, we consider the more general case where values x and y are not known to parties $\mathcal{A}$ and $\mathcal{B}$, respectively, but x and y are only given by encryptions $E(x)$ and $E(y)$ say. (This case arises for example when x and y result from earlier intermediate computations.) In order to compute an encryption $E(xy)$ we will decrypt some information related to x but without revealing any information on x. For convenience, we will assume that the bits x, y are represented as $X = (-1)^x$ and $Y = (-1)^y$ respectively, mapping $\{0, 1\}$ to $\{1, -1\}$.[3] The protocol for the passive case is as follows, using $E(\cdot)$ notation to hide some of the details:

1. Party $\mathcal{A}$ chooses $u \in_R \{1, -1\}$ and sends encryptions $E(Xu) = E(X)^u$ and $E(Yu) = E(Y)^u$ to party $\mathcal{B}$.
2. Party $\mathcal{B}$ chooses $v \in_R \{1, -1\}$ and sends encryptions $E(Xuv) = E(Xu)^v$ and $E(Yuv) = E(Yu)^v$ to party $\mathcal{A}$.
3. Parties $\mathcal{A}$ and $\mathcal{B}$ jointly decrypt $E(Xuv)$ to obtain the value of $z = Xuv$. The output of the protocol is set to $E(Yuvz) = E(Yuv)^z$.

It follows that the output of the protocol is indeed $E(Yuvz) = E(YuvXuv) = E(XY)$. Moreover, the value $z = Xuv$ as decrypted during the protocol is statistically independent of the value of X, both from $\mathcal{A}$'s point of view and from $\mathcal{B}$'s point of view: $\mathcal{A}$ does not know v, which is chosen at random in $\{1, -1\}$ by $\mathcal{B}$, and similarly $\mathcal{B}$ does not know u.

7.3 BASED ON OBLIVIOUS TRANSFER

As an alternative approach to secure computation we will now consider a solution based on oblivious transfer (OT). The most basic form of oblivious transfer is a protocol between a

[3] Since $(-1)^x = 1 - 2x$ for $x \in \{0, 1\}$ we have that $E(X) = E(1)/E(x)^2$. The reverse transformation is given by $E(x) = (E(1)/E(X))^{1/2}$. These transformations can be computed using the homomorphic property of $E(\cdot)$.

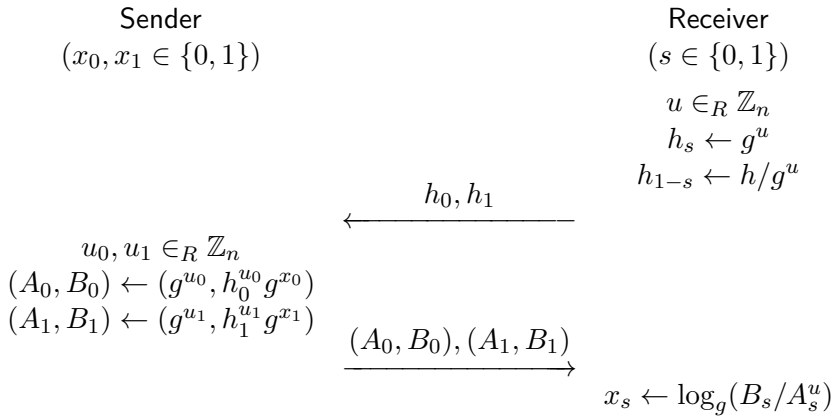


FIGURE 7.2: $\binom{2}{1}$-OT protocol

sender and receiver, achieving the following functionality. The sender uses one bit b as its private input to the protocol; the receiver does not provide any private input to the protocol. At the completion of the protocol, the receiver either gets the bit b or an undefined value $\perp$. Both cases occur with probability 50%, and the receiver knows whether it gets b or $\perp$. However, the sender does not know whether bit b was transferred successfully or not.

Despite the somewhat strange functionality of an oblivious transfer protocol, it turns out that OT is sufficiently powerful to construct a secure multiparty computation for *any* efficiently computable function. To argue why this is true, it is more convenient to use $\binom{2}{1}$-OT instead of the above "raw" version of OT. In a $\binom{2}{1}$-OT, the sender uses two private input bits x_0, x_1 and the receiver uses one private input bit s. At the completion of the protocol, the receiver now gets the bit x_s, whereas the sender does not get any information on the value of s, i.e., the sender does not know which bit was selected by the receiver. This functionality is denoted by $OT(x_0, x_1; s) = x_s$.

These two types of oblivious transfer are equivalent in the sense that either type can be constructed from the other one, using a polynomial time transformation. For the remainder of this section, we will only use $\binom{2}{1}$-OT.

In Figure 7.2 an example of a $\binom{2}{1}$-OT protocol for a discrete log setting is given. The common input to the protocol consists of a group $\langle g \rangle$ and a group element $h \in \langle g \rangle$, for which $\log_g h$ is not known to any party. The receiver is supposed to pick elements h_0, h_1 such that $h_0 h_1 = h$, which implies that the receiver cannot know both $\log_g h_0$ and $\log_g h_1$. Given h_0 and h_1, the sender returns homomorphic ElGamal encryptions of bits x_0 and x_1, respectively, using h_0 and h_1 as public keys. The receiver is then able to decrypt one of these encryptions, to recover either x_0 or x_1.

So, if both parties follow the protocol, the receiver learns exactly one of the bits x_0 and x_1 (and obtains no information about the other bit), and the sender does not have a clue which bit the receiver gets. Hence, the $\binom{2}{1}$-OT protocol of Figure 7.2 achieves its goal in the passive case.

Next, we show how parties $\mathcal{A}$ and $\mathcal{B}$, holding private input bits x and y respectively, may use such an OT protocol for computing the product xy. Hence, this is an alternative solution for matching without embarrassments. Focusing, again, on the passive case, we see that this problem can be handled using just a single run of a $\binom{2}{1}$-OT protocol, say in which $\mathcal{A}$ plays

the role of sender and $\mathcal{B}$ the role of receiver. The important observation is that

$$OT(x_0, x_1; s) = x_s = x_0(1-s) \oplus x_1 s,$$

which implies that if $\mathcal{A}$ uses 0 and x as values to send, and $\mathcal{B}$ uses y as selection bit, then we get $OT(0, x; y) = 0(1-y) \oplus xy = xy$.

More generally, it is possible to let two parties $\mathcal{A}$ and $\mathcal{B}$ multiply bits x and y securely, such that neither of them learns the values of x, y, and $z = xy$. This is done by letting $\mathcal{A}$ and $\mathcal{B}$ additively share these values, that is, at the start of the protocol $\mathcal{A}$ holds bits x_a, y_a and $\mathcal{B}$ holds bits x_b, y_b satisfying $x = x_a \oplus x_b$ and $y = y_a \oplus y_b$, and at the end $\mathcal{A}$ will hold a bit z_a and $\mathcal{B}$ a bit z_b satisfying $z = z_a \oplus z_b$.

Of course, the shares held by $\mathcal{A}$ should be independent of the values x, y, z and the same should be true for $\mathcal{B}$. Only if the shares are combined, the values x, y, z will result. A first attempt is to write $xy = (x_a \oplus x_b)(y_a \oplus y_b) = x_a y_a \oplus x_a y_b \oplus x_b y_a \oplus x_b y_b$, but it is unclear how to split this into shares z_a and z_b. Surely, the terms $x_a y_a$ and $x_b y_b$ can be computed by $\mathcal{A}$ and $\mathcal{B}$ on their own, whereas the terms $x_a y_b$ and $x_b y_a$ can be computed by two applications of the above protocol for the case of private inputs. However, by assembling z_a and z_b from these values, shares z_a and z_b will not be completely independent of shares x_b, y_b and x_a, y_a, respectively, which is also necessary for security.

The way out is to let both $\mathcal{A}$ and $\mathcal{B}$ contribute some randomness to the protocol. Concretely, each of them chooses a random bit $u_a, u_b \in_R \{0,1\}$, respectively, and uses it in a run of the OT protocol. The shares of $z = xy$ are computed symmetrically like this:

$$\begin{aligned} z_a &= x_a y_a \oplus u_a \oplus OT(u_b, x_b \oplus u_b; y_a) = x_a y_a \oplus u_a \oplus u_b \oplus x_b y_a \\ z_b &= x_b y_b \oplus u_b \oplus OT(u_a, x_a \oplus u_a; y_b) = x_b y_b \oplus u_b \oplus u_a \oplus x_a y_b. \end{aligned}$$

Clearly, the share z_a is now uniformly distributed and independent of all other values if u_b is selected uniformly at random (as an honest $\mathcal{B}$ will do), and similarly for z_b.

7.4 BIBLIOGRAPHIC NOTES

The basic results for secure multiparty computation date back to Yao's papers [Yao82a, Yao86], who focuses on the two-party case (and considers the millionaires problem as a concrete example), and to [GMW87, BGW88, CCD88] for the multiparty case. The electronic voting scheme is from [CGS97]; see [Sch10] for a survey of cryptographic voting schemes. The general approach for secure multiparty computation based on threshold homomorphic cryptosystems is due to [CDN01], whereas the adaptation to ElGamal is from [ST04]. Interestingly, the use of homomorphic cryptosystems for secure computation was already put forward in [RAD78], as a positive twist to the insecurity of *plain* versions of RSA, in which the RSA encryption and digital signature schemes are used without any message padding [RSA78].

The basic form of oblivious transfer in which the receiver gets the bit with 50% probability (and otherwise an undefined value) is due to Rabin [Rab81]. The $\binom{2}{1}$-OT protocol of Figure 7.2 is due to [BM89]. The (polynomial time) equivalence to $\binom{2}{1}$-OT was shown in [Cré87]. Kilian showed that OT is complete for secure multiparty computation, which basically means that protocols for securely evaluating any given efficiently computable function f can be built from OT only [Kil88]. His result covers the active case by showing how to implement the required commitments and zero-knowledge proofs from OT, whereas the case of passive adversaries had already been covered by [GMW87] (for which the sub-protocol described in Section 7.3 is an efficiency improvement due to [GV88]).

CHAPTER 8

Blind Signatures

A digital signature scheme provides the basic functionality of authentication of messages, allowing the holder of a private key to produce signatures on arbitrary messages. In many applications, however, there is a need for cryptographic schemes which are similar to digital signatures but not quite the same. The group signatures of Section 5.4.3 are an example. In this chapter, we will consider blind signature schemes as another variation.

Suppose we like to build an electronic payment scheme in which a bank issues electronic money to users who may then spend it at various shops. In such a scheme the bank may issue electronic money by sending the users digitally signed messages saying "This is a banknote worth $20. Serial number: P01144099136." To spend electronic money at a shop, the user will include such an electronic banknote in a payment to the shop. Since these banknotes are trivial to duplicate, however, the shop needs to deposit any received banknotes immediately at the bank to make sure that these banknotes have not been spent already at some other shop. The bank keeps a database containing all spent banknotes, and each banknote can be used for payment only once. If a payment to a shop contains a banknote that was already spent before, the payment is rejected.

Now, a basic property of any such payment scheme is that the bank is able to trace exactly at which places a user spends its money. This is due to the fact that in a digital signature scheme the signer gets to see all the messages it signs, and obviously, the signer knows all the signatures it produces for these messages. To achieve the level of anonymity as provided by cash payments using (metal) coins, where payment transactions need not leave any trace about the identity of the payers, one may use blind signatures instead of (ordinary) digital signatures.

8.1 DEFINITION

Like digital signatures, blind signatures are unforgeable and can be verified against a public key. The difference is that blind signatures are generated by means of a protocol between the signer and a receiver such that the signer does not see the message being signed. And, in addition, the signer does not learn any useful information on the signature being produced.

DEFINITION 8.1 *A **blind signature scheme** consists of the following three components.*

Key generation. *An algorithm that on input of a security parameter k, generates a key pair (sk, pk) consisting of a private key and a public key, respectively.*

Signature generation. *A two-party protocol between a signer $\mathcal{S}$ and a receiver $\mathcal{R}$ with a public key pk as common input. Private input of $\mathcal{S}$ is a private key sk, and private input of $\mathcal{R}$ is a message M. At the end of the protocol, $\mathcal{R}$ obtains a signature S on M as* private *output.*

Signature verification. *An algorithm that on input of a message M, a public key pk, and a signature S, determines whether S is a valid signature on M with respect to public key pk.*

The two basic security requirements for a blind signature scheme are unforgeability and unlinkability, which we state informally as follows. Let (sk, pk) be a key pair for a blind signature scheme. A pair (M, S) is valid if signature verification of M and S with respect to public key pk succeeds.

A blind signature scheme is **unforgeable** if for an adversary (not knowing sk) the only feasible way to obtain valid pairs (M, S) is to execute the signature generation protocol with a signer holding private key sk. More precisely, a blind signature scheme should withstand a **one-more forgery**: if an adversary is able to obtain ℓ valid pairs of messages and signatures, then the signer executed the signature generation protocol at least ℓ times. Preferably, we like this to hold for any positive ℓ bounded polynomially in the security parameter k.

A blind signature scheme is **unlinkable** if for an adversary (colluding with a signer) it is infeasible to link any valid pair (M, S) to the instance of the signature generation protocol in which it was created. More precisely, suppose a signer $\mathcal{S}$ and a receiver $\mathcal{R}$ play the following game. First they run the signature generation protocol resulting in a pair (M_0, S_0) and then they run it once more, resulting in (M_1, S_1). Then $\mathcal{R}$ flips a coin, that is, chooses $b \in_R \{0, 1\}$ and sends (M_b, S_b), (M_{1-b}, S_{1-b}) (in this order) to $\mathcal{S}$. Finally, $\mathcal{S}$ makes a guess for the value of b. Unlinkability means that the probability of $\mathcal{S}$ guessing b correctly is $\frac{1}{2}$, except for a difference negligible in the security parameter k.

8.2 CHAUM BLIND SIGNATURE SCHEME

Recall the RSA digital signature scheme, in which a public key consists of an RSA modulus $m = pq$ and a public exponent e, with $\gcd(e, \phi(m)) = 1$, and a corresponding private key consists of prime factors p, q and the private exponent $d = 1/e \bmod \phi(m)$. A signature S on a message $M \in \{0, 1\}^*$ is generated as $S = H(M)^d \bmod m$, where H is a suitable cryptographic hash function.[1] A signature is verified by checking that $S^e = H(M) \bmod m$.

Chaum's blind signature scheme is a modification of the RSA scheme, where signature generation is done using the protocol of Figure 8.1. The role of the signer is simply to extract an eth root for any value $y \in \mathbb{Z}_m^*$ it gets from a receiver. Thus a receiver is able to get signatures on arbitrary messages $M \in \{0, 1\}^*$. Instead of simply sending $H(M)$ to the signer, however, the receiver *blinds* it by sending $y = H(M)u^e$, where u is a random value, called a *blinding factor*. After receiving $x = y^{1/e}$, the receiver is able to *unblind* it such that a signature on M is obtained after all:

$$S^e = (x/u)^e = (y^{1/e}/u)^e = y/u^e = H(M) \bmod m.$$

[1] A so-called full-domain hash function $H : \{0, 1\}^* \to \mathbb{Z}_m^*$ should be used.

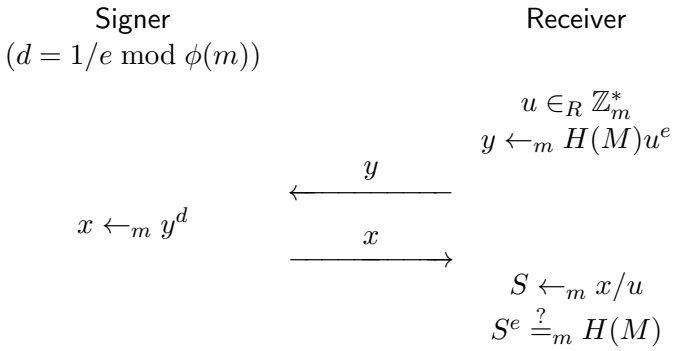


FIGURE 8.1: Chaum's blind signature protocol

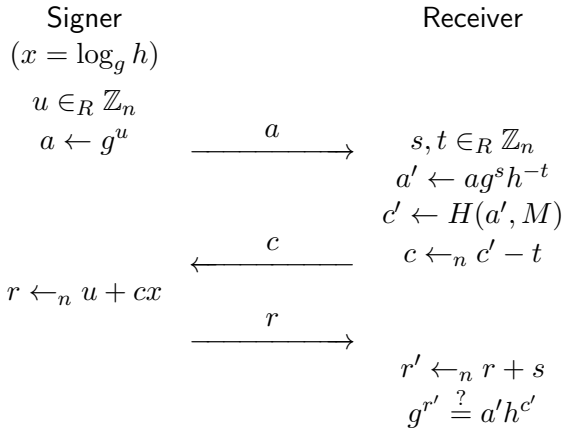


FIGURE 8.2: Schnorr-based blind signature protocol

The use of a blinding factor $u \in \mathbb{Z}_m^*$ ensures that the pair (M, S) is statistically independent of the pair (x, y), implying (perfect) unlinkability. On the other hand, it is not known whether unforgeability holds for Chaum's blind signature scheme under the standard RSA assumption (cf. Exercise 1.34). Under a special type of RSA assumption it is possible, however, to prove that the scheme is unforgeable.

EXERCISE 8.2 Consider Chaum's blind signature protocol (Figure 8.1) for a fixed message M. Show that the pair (x, y) is distributed uniformly random by proving that $\Pr[(x, y) = (x_0, y_0)] = 1/\phi(m)$ for any $x_0, y_0 \in \mathbb{Z}_m^*$ satisfying $y_0 = x_0^e \bmod m$.

8.3 BLIND SIGNATURES FROM Σ-PROTOCOLS

Almost all of the Σ-protocols we have seen so far enjoy the property that the verification relation is homomorphic. For example, referring to Schnorr's protocol of Figure 4.3, we have the following homomorphic property for two accepting conversations $(a; c; r)$ and $(a'; c'; r')$:

$$g^r = ah^c, \quad g^{r'} = a'h^{c'} \qquad \Rightarrow \qquad g^{r+r'} = aa'h^{c+c'}.$$

That is, $(aa'; c + c'; r + r')$ is an accepting conversation as well. For this reason it is easy to obtain blind signature schemes from Σ-protocols. Figure 8.2 shows a blind signature

protocol based on Schnorr's protocol. The output of the protocol is a signature $S = (c', r')$ on message M satisfying $c' = H(g^{r'}h^{-c'}, M)$, which is the same verification relation as for Schnorr signatures.

The completeness of the protocol is easily checked:

$$g^{r'} = g^{r+s} = ah^c g^s = a'g^{-s}h^t h^{c'-t}g^s = a'h^{c'}.$$

One may think of the triple $(g^s h^{-t}; t; s)$ as a simulated conversation, which is multiplied with conversation $(a; c; r)$ to obtain a blinded conversation $(a'; c'; r') = (ag^s h^{-t}; c + t; r + s)$, using the above-mentioned homomorphic property for accepting conversations.

It is also clear that the triples $(a; c; r)$ and $(a'; c'; r')$ are statistically independent, ensuring (perfect) unlinkability.

As for unforgeability, however, it turns out that one-more forgeries can only be provably excluded when the adversary is limited to a relatively small number of interactions with the signer.[2]

EXERCISE 8.3 Consider the Schnorr-based blind signature protocol (Figure 8.2) for a fixed message M. Show that the triples (a, c, r), (a', c', r') are distributed uniformly random and independent of each other by proving that $\Pr[(a; c; r) = (a_0; c_0; r_0) \wedge (a'; c'; r') = (a'_0; c'_0; r'_0)] = 1/n^3$ for any $(a_0; c_0; r_0)$, $(a'_0; c'_0; r'_0)$ satisfying $g^{r_0} = a_0 h^{c_0}$, $c'_0 = H(a'_0, M)$, and $g^{r'_0} = a'_0 h^{c'_0}$.

8.4 BIBLIOGRAPHIC NOTES

The notion of blind signatures was introduced by Chaum at CRYPTO'82 [Cha83], who also presented the first blind signature protocol at CRYPTO'83 [Cha84] (see also the corresponding patent [Cha88]). Originally, one of the main applications of blind signatures was anonymous electronic cash, which formed together with follow-up patents the core technology of the eCash system (developed by Chaum's company DigiCash when the world wide web was spun in the early 1990s, see also [Sch97]).

The idea that for many protocols, including Σ-protocols, one may render blind signature schemes in a systematic way is from Okamoto and Ohta [OO90]. The technique is called *divertibility*, and in the case of Σ-protocols, divertibility boils down to an intermediate party blinding and unblinding the messages exchanged between a prover and a verifier. This way unlinkability is ensured, but whether the resulting blind signature scheme is unforgeable as well depends on more specific properties of the underlying Σ-protocol.

For instance, Pointcheval and Stern [PS00] have shown that by diverting Okamoto's witness-indistinguishable Σ-protocol (see Figure 4.5), one obtains a blind signature scheme which is unforgeable as long as the number of (parallel) runs in which an attacker may engage is sufficiently small as a function of a security parameter; moreover, there actually exists a subexponential attack on this blind signature scheme if the attacker is allowed to engage in (too) many parallel runs, as shown by Wagner [Wag02]. A possible countermeasure for this type of attack is to limit the number of parallel runs that can be active at any given moment in time, but such a limitation may be undesirable in some situations and may be hard to enforce. Using more advanced—but also costlier—techniques, it is possible to get blind signature schemes without such limitations; see, e.g., [Fis06].

[2] In fact, when many interactions with the signer can be executed in parallel a one-more forgery becomes possible under certain circumstances.

Bibliography

[AL83] D. Angluin and D. Lichtenstein. Provable security of cryptosystems: A survey. Technical Report TR-288, Yale University, October 1983.

[BCDP91] J. Boyar, D. Chaum, I. Damgård, and T. Pedersen. Convertible undeniable signatures. In *Advances in Cryptology—CRYPTO '90*, volume 537 of *LNCS*, pages 189–205, Berlin, 1991. Springer.

[BDF98] D. Boneh, G. Durfee, and Y. Frankel. An attack on RSA given a small fraction of the private key bits. In *Advances in Cryptology—ASIACRYPT '98*, volume 1514 of *LNCS*, pages 25–34, Berlin, 1998. Springer.

[BGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. 20th Symposium on Theory of Computing (STOC '88)*, pages 1–10, New York, 1988. ACM.

[Bla79] G. R. Blakley. Safeguarding cryptographic keys. In *Proceedings of the National Computer Conference 1979*, volume 48 of *AFIPS Conference Proceedings*, pages 313–317, 1979.

[Blu82] M. Blum. Coin flipping by telephone. In A. Gersho, editor, *Advances in Cryptology—CRYPTO '81*, pages 133–137. IEEE, 1982.

[BM89] M. Bellare and S. Micali. Non-interactive oblivious transfer and applications. In *Advances in Cryptology—CRYPTO '89*, volume 435 of *LNCS*, pages 547–557, Berlin, 1989. Springer.

[BM92] S. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *IEEE Symposium on Research in Security and Privacy*, pages 72–84. IEEE Computer Society Press, 1992.

[BM03] C. Boyd and A. Mathuria. *Protocols for Authentication and Key Establishment.* Springer, Berlin, 2003.

[Bon98] D. Boneh. The decision Diffie-Hellman problem. In *Third Algorithmic Number Theory Symposium*, volume 1423 of *LNCS*, pages 48–63, Berlin, 1998. Springer.

[BR93] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *1st ACM Conference on Computer and Communications Security, Fairfax*, pages 62–73. ACM press, 1993.

[Bra93] S. Brands. An efficient off-line electronic cash system based on the representation problem. Report CS-R9323, Centrum voor Wiskunde en Informatica, Amsterdam, March 1993.

[Bra97] S. Brands. Rapid demonstration of linear relations connected by Boolean operators. In *Advances in Cryptology—EUROCRYPT '97*, volume 1233 of *LNCS*, pages 318–333, Berlin, 1997. Springer.

[Cam97] J. Camenisch. Efficient and generalized group signatures. In *Advances in Cryptology—EUROCRYPT '97*, volume 1233 of *LNCS*, pages 465–479, Berlin, 1997. Springer.

[CCD88] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *Proc. 20th Symposium on Theory of Computing (STOC '88)*, pages 11–19, New York, 1988. ACM.

[CD98] R. Cramer and I. Damgård. Zero-knowledge proofs for finite field arithmetic, or: Can zero-knowledge be for free? In *Advances in Cryptology—CRYPTO '98*, volume 1462 of *LNCS*, pages 424–441, Berlin, 1998. Springer.

[CDN01] R. Cramer, I. Damgård, and J.B. Nielsen. Multiparty computation from threshold homomorphic encryption. In *Advances in Cryptology—EUROCRYPT '01*, volume 2045 of *LNCS*, pages 280–300, Berlin, 2001. Springer. Full version eprint.iacr.org/2000/055, October 27, 2000.

[CDS94] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology—CRYPTO '94*, volume 839 of *LNCS*, pages 174–187, Berlin, 1994. Springer.

[CFSY96] R. Cramer, M. Franklin, B. Schoenmakers, and M. Yung. Multi-authority secret ballot elections with linear work. In *Advances in Cryptology—EUROCRYPT '96*, volume 1070 of *LNCS*, pages 72–83, Berlin, 1996. Springer.

[CGS97] R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *Advances in Cryptology—EUROCRYPT '97*, volume 1233 of *LNCS*, pages 103–118, Berlin, 1997. Springer.

[CH91] D. Chaum and E. van Heyst. Group signatures. In *Advances in Cryptology—EUROCRYPT '91*, volume 547 of *LNCS*, pages 257–265, Berlin, 1991. Springer.

[Cha83] D. Chaum. Blind signatures for untraceable payments. In D. Chaum, R.L. Rivest, and A.T. Sherman, editors, *Advances in Cryptology—CRYPTO '82*, pages 199–203, New York, 1983. Plenum Press.

[Cha84] D. Chaum. Blind signature system. In *Advances in Cryptology—CRYPTO '83*, page 153, New York, 1984. Plenum Press.

[Cha88] D. Chaum. Blind signature systems. U.S. Patent #4,759,063, 1988. Issued on July 19, 1988, filed on August 22, 1983.

[CJ02] D. Coppersmith and M. Jakobsson. Almost optimal hash sequence traversal. In *Financial Cryptography 2002*, volume 2357 of *LNCS*, pages 102–119, Berlin, 2002. Springer.

[CP93] D. Chaum and T. P. Pedersen. Wallet databases with observers. In *Advances in Cryptology—CRYPTO '92*, volume 740 of *LNCS*, pages 89–105, Berlin, 1993. Springer.

[Cra97] R. Cramer. *Modular Design of Secure yet Practical Cryptographic Protocols*. PhD thesis, Universiteit van Amsterdam, Netherlands, 1997.

[Cré87] C. Crépeau. Equivalence between two flavours of oblivious transfer. In *Advances in Cryptology—CRYPTO '87*, volume 293 of *LNCS*, pages 350–354, Berlin, 1987. Springer.

[Dam10] I. Damgård. On Σ-protocols, 2010. www.daimi.au.dk/~ivan/Sigma.pdf.

[Des88] Y. Desmedt. Society and group oriented cryptography: A new concept. In *Advances in Cryptology—CRYPTO '87*, volume 293 of *LNCS*, pages 120–127, Berlin, 1988. Springer.

[Des94] Y. Desmedt. Threshold cryptography. *European Transactions on Telecommunications*, 5(4):449–457, 1994.

[DF90] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *Advances in Cryptology—CRYPTO '89*, volume 435 of *LNCS*, pages 307–315, Berlin, 1990. Springer.

[DH76] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[ElG85] T. ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, IT-31(4):469–472, 1985.

[Fel87] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proc. 28th IEEE Symposium on Foundations of Computer Science (FOCS '87)*, pages 427–437. IEEE Computer Society, 1987.

[FF93] J. Feigenbaum and L. Fortnow. On the random-self-reducibility of complete sets. *SIAM Journal on Computing*, 22:994–1005, 1993.

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

[Fis06] M. Fischlin. Round-optimal composable blind signatures in the common reference string model. In *Advances in Cryptology—CRYPTO '06*, volume 4117 of *LNCS*, pages 60–77, Berlin, 2006. Springer.

[FMR96] M. J. Fischer, S. Micali, and C. Rackoff. A secure protocol for the oblivious transfer (extended abstract). *Journal of Cryptology*, 9(3):191–195, 1996.

[FS87] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology—CRYPTO '86*, volume 263 of *LNCS*, pages 186–194, New York, 1987. Springer.

[FS90] U. Feige and A. Shamir. Witness indistinguishable and witness hiding protocols. In *Proc. 22nd Symposium on Theory of Computing (STOC '90)*, pages 416–426, New York, 1990. ACM.

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18:186–208, 1989. Preliminary version in *17th ACM Symposium on the Theory of Computing (STOC '85)*.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game - or - a completeness theorem for protocols with honest majority. In *Proc. 19th Symposium on Theory of Computing (STOC '87)*, pages 218–229, New York, 1987. ACM.

[GQ88] L. C. Guillou and J.-J. Quisquater. A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory. In *Advances in Cryptology—EUROCRYPT '88*, volume 330 of *LNCS*, pages 123–128, Berlin, 1988. Springer.

[GV88] O. Goldreich and R. Vainish. How to solve any protocol problem–an efficiency improvement. In *Advances in Cryptology—CRYPTO '87*, volume 293 of *LNCS*, pages 73–86, Berlin, 1988. Springer.

[IR01] G. Itkis and L. Reyzin. Forward-secure signatures with optimal signing and verifying. In *Advances in Cryptology—CRYPTO '01*, volume 2139 of *LNCS*, pages 332–354, Berlin, 2001. Springer.

[Jak02] M. Jakobsson. Fractal hash sequence representation and traversal. In *Proc. IEEE International Symposium on Information Theory (ISIT '02)*, page 437. IEEE Press, 2002. Full version eprint.iacr.org/2002/001.

[Kil88] J. Kilian. Founding cryptography on oblivious transfer. In *Proc. 20th Symposium on Theory of Computing (STOC '88)*, pages 20–31, New York, 1988. ACM.

[Kob87] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.

[Lam81] L. Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, 1981.

[Mer87] R. Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology—CRYPTO '87*, volume 293 of *LNCS*, pages 369–378, Berlin, 1987. Springer.

[Mer89] R. Merkle. A certified digital signature. In *Advances in Cryptology—CRYPTO '89*, volume 435 of *LNCS*, pages 218–238, Berlin, 1989. Springer.

[Mil86] V. S. Miller. Use of elliptic curves in cryptography. In *Advances in Cryptology—CRYPTO '85*, volume 218 of *LNCS*, pages 417–426, Berlin, 1986. Springer.

[MOV97] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[NR97] M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *Proc. 38th IEEE Symposium on Foundations of Computer Science (FOCS '97)*, pages 458–467. IEEE Computer Society, 1997.

[Oka93] T. Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In *Advances in Cryptology—CRYPTO '92*, volume 740 of *LNCS*, pages 31–53, Berlin, 1993. Springer.

[OO90] T. Okamoto and K. Ohta. Divertible zero knowledge interactive proofs and commutative random self-reducibility. In *Advances in Cryptology—EUROCRYPT '89*, volume 434 of *LNCS*, pages 134–149, Berlin, 1990. Springer.

[Ped91] T. P. Pedersen. A threshold cryptosystem without a trusted party. In *Advances in Cryptology—EUROCRYPT '91*, volume 547 of *LNCS*, pages 522–526, Berlin, 1991. Springer.

[Ped92] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology—CRYPTO '91*, volume 576 of *LNCS*, pages 129–140, Berlin, 1992. Springer.

[PS00] D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, 2000.

[Rab81] M. Rabin. How to exchange secrets by oblivious transfer. Technical Memo TR-81, Aiken Computation Laboratory, Harvard University, 1981.

[RAD78] R. Rivest, L. Adleman, and M. Dertouzos. On data banks and privacy homomorphisms. In *Foundations of Secure Computation*, pages 169–177. Academic Press, 1978.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978. Reprinted in 1983 in 26(1):96–99.

[Sch91] C. P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[Sch97] B. Schoenmakers. Basic security of the eCash payment system. In *State of the Art in Applied Cryptography*, volume 1528 of *LNCS*, pages 338–352, Berlin, 1997. Springer. Correct version at `www.win.tue.nl/~berry/papers/cosic.pdf`.

[Sch10] B. Schoenmakers. Voting schemes. In M.J. Atallah and M. Blanton, editors, *Algorithms and Theory of Computation Handbook: Special Topics and Techniques*, chapter 15, pages **15**–1–21. Chapman & Hall/CRC, 2nd edition, 2010.

[Sch16] B. Schoenmakers. Explicit optimal binary pebbling for one-way hash chain reversal. In *Financial Cryptography 2016*, volume 9603 of *LNCS*, pages 299–320, Berlin, 2016. Springer. See also www.win.tue.nl/~berry/pebbling/.

[Sha79] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[Sho99] V. Shoup. On formal models for secure key exchange. Report RZ 3120, IBM Research, Zurich, April 1999. See also, updated paper (version 4, November 15, 1999), available at www.shoup.net/papers/skey.pdf.

[ST04] B. Schoenmakers and P. Tuyls. Practical two-party computation based on the conditional gate. In *Advances in Cryptology—ASIACRYPT '04*, volume 3329 of *LNCS*, pages 119–136, Berlin, 2004. Springer.

[Sta96] M. Stadler. Publicly verifiable secret sharing. In *Advances in Cryptology—EUROCRYPT '96*, volume 1070 of *LNCS*, pages 190–199, Berlin, 1996. Springer.

[Szy04] M. Szydlo. Merkle tree traversal in log space and time. In *Advances in Cryptology—EUROCRYPT '04*, volume 3027 of *LNCS*, pages 541–554, Berlin, 2004. Springer.

[TW87] M. Tompa and H. Woll. Random self-reducibility and zero knowledge interactive proofs of possession. In *Proc. 28th IEEE Symposium on Foundations of Computer Science (FOCS '87)*, pages 472–482. IEEE Computer Society, 1987.

[Wag02] D. Wagner. A generalized birthday problem. In *Advances in Cryptology—CRYPTO '02*, volume 2442 of *LNCS*, pages 288–304, Berlin, 2002. Springer.

[Yao82a] A. Yao. Protocols for secure computations. In *Proc. 23rd IEEE Symposium on Foundations of Computer Science (FOCS '82)*, pages 160–164. IEEE Computer Society, 1982.

[Yao82b] A. Yao. Theory and applications of trapdoor functions. In *Proc. 23rd IEEE Symposium on Foundations of Computer Science (FOCS '82)*, pages 80–91. IEEE Computer Society, 1982.

[Yao86] A. Yao. How to generate and exchange secrets. In *Proc. 27th IEEE Symposium on Foundations of Computer Science (FOCS '86)*, pages 162–167. IEEE Computer Society, 1986.