Foundations of Cryptography – A Primer

Oded Goldreich
Department of Computer Science
Weizmann Institute of Science
Rehovot, Israel.
oded.goldreich@weizmann.ac.il

February 13, 2005

Abstract

We survey the main paradigms, approaches and techniques used to conceptualize, define and provide solutions to natural cryptographic problems. We start by presenting some of the central tools used in cryptography; that is, computational difficulty (in the form of one-way functions), pseudorandomness, and zero-knowledge proofs. Based on these tools, we turn to the treatment of basic cryptographic applications such as encryption and signature schemes as well as the design of general secure cryptographic protocols.

Our presentation assumes basic knowledge of algorithms, probability theory and complexity theory, but nothing beyond this.

Keywords: Cryptography, Theory of Computation.

Contents

1 Introduction and Preliminaries

It is possible to build a cabin with no foundations,
but not a lasting building.

Eng. Isidor Goldreich (1906–1995)

1.1 Introduction

The vast expansion and rigorous treatment of cryptography is one of the major achievements of theoretical computer science. In particular, concepts such as computational indistinguishability, pseudorandomness and zero-knowledge interactive proofs were introduced, classical notions such as secure encryption and unforgeable signatures were placed on sound grounds, and new (unexpected) directions and connections were uncovered. Indeed, modern cryptography is strongly linked to complexity theory (in contrast to "classical" cryptography which is strongly related to information theory).

Modern cryptography is concerned with the construction of information systems that are robust against malicious attempts to make these systems deviate from their prescribed functionality. The prescribed functionality may be the private and authenticated communication of information through the Internet, the holding of incoercible and secret electronic voting, or conducting any "fault-resilient" multi-party computation. Indeed, the scope of modern cryptography is very broad, and it stands in contrast to "classical" cryptography (which has focused on the single problem of enabling secret communication over insecure communication media).

The design of cryptographic systems is a very difficult task. One cannot rely on intuitions regarding the "typical" state of the environment in which the system operates. For sure, the adversary attacking the system will try to manipulate the environment into "untypical" states. Nor can one be content with counter-measures designed to withstand specific attacks, since the adversary (which acts after the design of the system is completed) will try to attack the schemes in ways that are different from the ones the designer had envisioned. The validity of the above assertions seems self-evident, still some people hope that in practice ignoring these tautologies will not result in actual damage. Experience shows that these hopes rarely come true; cryptographic schemes based on make-believe are broken, typically sooner than later.

In view of the foregoing, we believe that it makes little sense to make assumptions regarding the specific *strategy* that the adversary may use. The only assumptions that can be justified refer to the computational *abilities* of the adversary. Furthermore, the design of cryptographic systems has to be based on *firm foundations*; whereas ad-hoc approaches and heuristics are a very dangerous way to go. A heuristic may make sense when the designer has a very good idea regarding the environment in which a scheme is to operate, yet a cryptographic scheme has to operate in a maliciously selected environment which typically transcends the designer's view.

This primer is aimed at presenting the foundations for cryptography. The foundations of cryptography are the paradigms, approaches and techniques used to conceptualize, define and provide solutions to natural "security concerns". We will present some of these paradigms, approaches and techniques as well as some of the fundamental results obtained using them. Our emphasis is on the clarification of fundamental concepts and on demonstrating the feasibility of solving several central cryptographic problems.

Solving a cryptographic problem (or addressing a security concern) is a two-stage process consisting of a *definitional stage* and a *constructive stage*. First, in the definitional stage, the functionality underlying the natural concern is to be identified, and an adequate cryptographic problem has to be defined. Trying to list all undesired situations is infeasible and prone to error. Instead, one should define the functionality in terms of operation in an imaginary ideal model, and require a candidate solution to emulate this operation in the real, clearly defined, model (which specifies the adversary's abilities). Once the definitional stage is completed, one proceeds to construct a system that satisfies the definition. Such a construction may use some simpler tools, and its security is proved relying on the features of these tools. In practice, of course, such a scheme may need to satisfy also some *specific* efficiency requirements.

This primer focuses on several archetypical cryptographic problems (e.g., encryption and signature schemes) and on several central tools (e.g., computational difficulty, pseudorandomness, and zero-knowledge proofs). For each of these problems (resp., tools), we start by presenting the natural concern underlying it (resp., its intuitive objective), then define the problem (resp., tool), and finally demonstrate that the problem may be solved (resp., the tool can be constructed). In the latter step, our focus is on demonstrating the feasibility of solving the problem, not on providing a practical solution. As a secondary concern, we typically discuss the level of practicality (or impracticality) of the given (or known) solution.

Computational Difficulty

The aforementioned tools and applications (e.g., secure encryption) exist only if some sort of computational hardness exists. Specifically, all these problems and tools require (either explicitly or implicitly) the ability to generate instances of hard problems. Such ability is captured in the definition of one-way functions. Thus, one-way functions are the very minimum needed for doing most natural tasks of cryptography. (It turns out, as we shall see, that this necessary condition is "essentially" sufficient; that is, the existence of one-way functions (or augmentations and extensions of this assumption) suffices for doing most of cryptography.)

Our current state of understanding of efficient computation does not allow us to prove that one-way functions exist. In particular, if $\mathcal{P} = \mathcal{NP}$ then no one-way functions exist. Furthermore, the existence of one-way functions implies that $\mathcal{NP}$ is not contained in $\mathcal{BPP} \supseteq \mathcal{P}$ (not even "on the average"). Thus, proving that one-way functions exist is not easier than proving that $\mathcal{P} \neq \mathcal{NP}$; in fact, the former task seems significantly harder than the latter. Hence, we have no choice (at this stage of history) but to assume that one-way functions exist. As justification to this assumption we may only offer the combined beliefs of hundreds (or thousands) of researchers. Furthermore, these beliefs concern a simply stated assumption, and their validity follows from several widely believed conjectures which are central to various fields (e.g., the conjectured intractability of integer factorization is central to computational number theory).

Since we need assumptions anyhow, why not just assume what we want (i.e., the existence of a solution to some natural cryptographic problem)? Well, first we need to know what we want: as stated above, we must first clarify what exactly we want; that is, go through the typically complex definitional stage. But once this stage is completed, can we just assume that the definition derived can be met? Not really: once a definition is derived, how can we know that it can at all be met? The way to demonstrate that a definition is viable (and that the corresponding intuitive security concern can be satisfied at all) is to construct a solution based on a *better understood* assumption (i.e., one that is more common and widely believed). For example, looking at the definition of zero-knowledge proofs, it is not a-priori clear that such proofs exist at all (in a non-trivial sense). The non-triviality of the notion was first demonstrated by presenting a zero-knowledge proof system for statements, regarding Quadratic Residuosity, which are believed to be hard to verify (without extra information). Furthermore, contrary to prior beliefs, it was later shown that the existence of one-way functions implies that any NP-statement can be proved in zero-knowledge. Thus, facts that were not known at all to hold (and even believed to be false), were shown to hold by reduction to widely believed assumptions (without which most of modern cryptography collapses anyhow). To summarize, not all assumptions are equal, and so reducing a complex, new and doubtful assumption to a widely-believed simple (or even merely simpler) assumption is of great value. Furthermore, reducing the solution of a new task to the assumed security of a well-known primitive typically means providing a construction that, using the known primitive, solves the new task. This means that we do not only know (or assume) that the new task is solvable but we also have a solution based on a primitive that, being well-known, typically has several candidate implementations.

Prerequisites and Structure

Our aim is to present the basic concepts, techniques and results in cryptography. As stated above, our emphasis is on the clarification of fundamental concepts and the relationship among them. This is done in a way independent of the particularities of some popular number theoretic examples. These particular examples played a central role in the development of the field and still offer the most practical implementations of all cryptographic primitives, but this does not mean that the presentation has to be linked to them. On the contrary, we believe that concepts are best clarified when presented at an abstract level, decoupled from specific implementations. Thus, the most relevant background for this primer is provided by basic knowledge of algorithms (including randomized ones), computability and elementary probability theory.

The primer is organized in two main parts, which are preceded by preliminaries (regarding efficient and feasible computations). The two parts are Part I – Basic Tools and Part II – Basic Applications. The basic tools consist of computational difficulty (one-way functions), pseudorandomness and zero-knowledge proofs. These basic tools are used for the basic applications, which in turn consist of Encryption Schemes, Signature Schemes, and General Cryptographic Protocols.

Introduction and Preliminaries
Part I: Basic Tools
- Chapter 2: Computational Difficulty (One-Way Functions)
- Chapter 3: Pseudorandomness
- Chapter 4: Zero-Knowledge

Part II: Basic Applications
- Chapter 5: Encryption Schemes
- Chapter 6: Signature and Message Authentication Schemes
- Chapter 7: General Cryptographic Protocols

Figure 1: Organization of this primer

In order to give some feeling of the flavor of the area, we have included in this primer a few proof sketches, which some readers may find too terse. We stress that following these proof sketches is *not* essential to understanding the rest of the material. In general, later sections may refer to definitions and results in prior sections, but not to the constructions and proofs that support these results. It may be even possible to understand later sections without reading any prior section, but we believe that the order we chose shoukd be preferred because it proceeds from the simplest notions to the most complex ones.

Suggestions for Further Reading

This primer is a brief summary of the author's two-volume work on the subject [67, 68]. Furthermore, Part I corresponds to [67], whereas Part II corresponds to [68]. Needless to say, the reader is referred to these textbooks for further detail.

Two of the topics reviewed by this primer are zero-knowledge proofs (which are probabilistic) and pseudorandom generators (and functions). A wider perspective on probabilistic proof systems and pseudorandomness is provided in [66, Chap. 2-3].

Current research on the foundations of cryptography appears in general computer science conferences (e.g., FOCS and STOC), in cryptography conferences (e.g., Crypto and EuroCrypt) as well as in the newly established *Theory of Cryptography Conference* (TCC).

Practice. The aim of this primer is to introduce the reader to the *theoretical foundations* of cryptography. As argued above, such foundations are necessary for *sound* practice of cryptography. Indeed, practice requires more than theoretical foundations, whereas the current primer makes no attempt to provide anything beyond the latter. However, given a sound foundation, one can learn and evaluate various practical suggestions that appear elsewhere (e.g., in [100]). On the other hand, lack of sound foundations results in inability to critically evaluate practical suggestions, which in turn leads to unsound decisions. Nothing could be more harmful to the design of schemes that need to withstand adversarial attacks than misconceptions about such attacks.

Non-cryptographic references: Some "non-cryptographic" works were referenced for sake of wider perspective. Examples include [3, 4, 5, 6, 57, 71, 79, 99, 123].

1.2 Preliminaries

Modern Cryptography, as surveyed here, is concerned with the construction of *efficient* schemes for which it is *infeasible* to violate the security feature. Thus, we need a notion of efficient computations as well as a notion of infeasible ones. The computations of the legitimate users of the scheme ought be efficient, whereas violating the security features (by an adversary) ought to be infeasible. We stress that we do not identify feasible computations with efficient ones, but rather view the former notion as potentially more liberal.

Efficient Computations and Infeasible ones

Efficient computations are commonly modeled by computations that are polynomial-time in the security parameter. The polynomial bounding the running-time of the legitimate user's strategy is *fixed and typically explicit* (and *small*). Indeed, our aim is to have a notion of efficiency that is as strict as possible (or, equivalently, develop strategies that are as efficient as possible). Here (i.e., when referring to the complexity of the legitimate users) we are in the same situation as in any algorithmic setting. Things are different when referring to our assumptions regarding the computational resources of the adversary, where we refer to the notion of feasible that we wish to be as wide as possible. A common approach is to postulate that feasible computations are polynomial-time too, but here the polynomial is *not a-priori specified* (and is to be thought of as arbitrarily large). In other words, the adversary is restricted to the class of polynomial-time computations and anything beyond this is considered to be infeasible.

Although many definitions explicitly refer to the convention of associating feasible computations with polynomial-time ones, this convention is *inessential* to any of the results known in the area.

In all cases, a more general statement can be made by referring to a general notion of feasibility, which should be preserved under standard algorithmic composition, yielding theories that refer to adversaries of running-time bounded by any specific super-polynomial function (or class of functions). Still, for sake of concreteness and clarity, we shall use the former convention in our formal definitions (but our motivational discussions will refer to an unspecified notion of feasibility that covers at least efficient computations).

Randomized (or probabilistic) Computations

Randomized computations play a central role in cryptography. One fundamental reason for this fact is that randomness is essential for the existence (or rather the generation) of secrets. Thus, we must allow the legitimate users to employ randomized computations, and certainly (since randomization is feasible) we must consider also adversaries that employ randomized computations. This brings up the issue of success probability: typically, we require that legitimate users succeed (in fulfilling their legitimate goals) with probability 1 (or negligibly close to this), whereas adversaries succeed (in violating the security features) with negligible probability. Thus, the notion of a negligible probability plays an important role in our exposition. One requirement of the definition of negligible probability is to provide a robust notion of rareness: A rare event should occur rarely even if we repeat the experiment for a feasible number of times. That is, in case we consider any polynomial-time computation to be feasible, a function $\mu:\mathbb{N}\to\mathbb{N}$ is called negligible if $1-(1-\mu(n))^{p(n)} < 0.01$ for every polynomial p and sufficiently big n (i.e., μ is negligible if for every positive polynomial p' the function $\mu(\cdot)$ is upper-bounded by $1/p'(\cdot)$). However, if we consider the function $T(n)$ to provide our notion of infeasible computation then functions bounded above by $1/T(n)$ are considered negligible (in n).

We will also refer to the notion of noticeable probability. Here the requirement is that events that occur with noticeable probability, will occur almost surely (i.e., except with negligible probability) if we repeat the experiment for a polynomial number of times. Thus, a function $\nu:\mathbb{N}\to\mathbb{N}$ is called noticeable if for some positive polynomial p' the function $\nu(\cdot)$ is lower-bounded by $1/p'(\cdot)$.

Part I
Basic Tools

In this part we survey three basic tools used in Modern Cryptography. The most basic tool is computational difficulty, which in turn is captured by the notion of one-way functions. Next, we survey the notion of computational indistinguishability, which underlies the theory of pseudorandomness as well as much of the rest of cryptography. In particular, pseudorandom generators and functions are important tools that will be used in later sections. Finally, we survey zero-knowledge proofs, and their use in the design of cryptographic protocols. For more details regarding the contents of the current part, see our textbook [67].

2 Computational Difficulty and One-Way Functions

Modern Cryptography is concerned with the construction of systems that are easy to operate (properly) but hard to foil. Thus, a complexity gap (between the ease of proper usage and the difficulty of deviating from the prescribed functionality) lies at the heart of Modern Cryptography. However, gaps as required for Modern Cryptography are not known to exist; they are only widely believed to exist. Indeed, almost all of Modern Cryptography rises or falls with the question of whether one-way functions exist. We mention that the existence of one-way functions implies that $\mathcal{NP}$ contains search problems that are hard to solve *on the average*, which in turn implies that $\mathcal{NP}$ is not contained in $\mathcal{BPP}$ (i.e., a worst-case complexity conjecture).

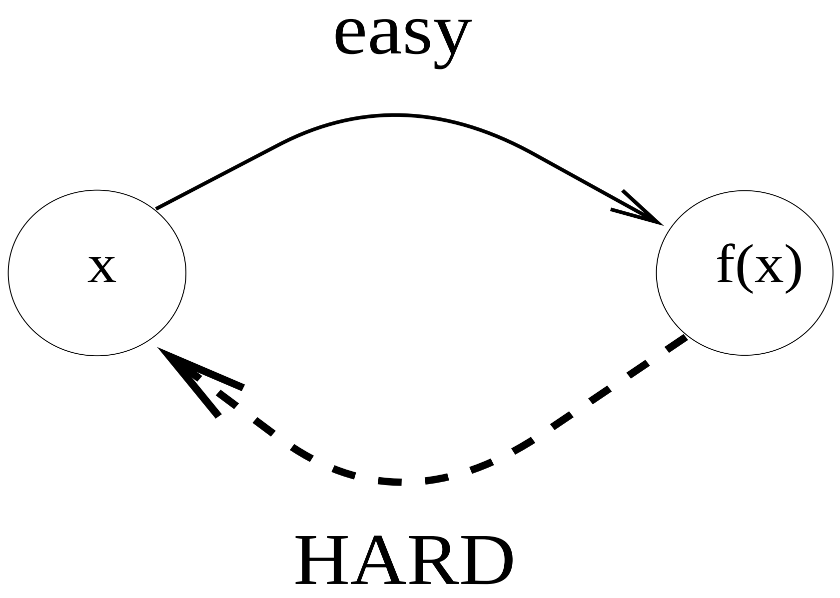


Figure 2: One way functions – an illustration.

Loosely speaking, one-way functions are functions that are easy to evaluate but hard (on the average) to invert. Such functions can be thought of as an efficient way of generating "puzzles" that are infeasible to solve (i.e., the puzzle is a random image of the function and a solution is a corresponding preimage). Furthermore, the person generating the puzzle knows a solution to it and can efficiently verify the validity of (possibly other) solutions to the puzzle. Thus, one-way functions have, by definition, a clear cryptographic flavor (i.e., they manifest a gap between the ease of one task and the difficulty of a related one).

2.1 One-Way Functions

One-way functions are functions that are efficiently computable but infeasible to invert (in an average-case sense). That is, a function $f:\{0,1\}^* \to \{0,1\}^*$ is called one-way if there is an efficient algorithm that on input x outputs $f(x)$, whereas any feasible algorithm that tries to find a preimage of $f(x)$ under f may succeed only with negligible probability (where the probability is taken uniformly over the choices of x and the algorithm's coin tosses). Associating feasible computations with probabilistic polynomial-time algorithms, we obtain the following definition.

Definition 2.1 (one-way functions): *A function $f:\{0,1\}^* \to \{0,1\}^*$ is called* one-way *if the following two conditions hold:*

1. easy to evaluate: *There exist a polynomial-time algorithm A such that $A(x) = f(x)$ for every $x \in \{0,1\}^*$.*
2. hard to invert: *For every probabilistic polynomial-time algorithm A', every polynomial p, and all sufficiently large n,*
$$\Pr[A'(f(x), 1^n) \in f^{-1}(f(x))] \; < \; \frac{1}{p(n)}$$
where the probability is taken uniformly over all the possible choices of $x \in \{0,1\}^n$ and all the possible outcomes of the internal coin tosses of algorithm A'.

Algorithm A' is given the auxiliary input 1^n so to allow it to run in time polynomial in the length of x, which is important in case f drastically shrinks its input (e.g., $|f(x)| = O(\log |x|)$). Typically, f is length preserving, in which case the auxiliary input 1^n is redundant. Note that A' is not required to output a specific preimage of $f(x)$; any preimage (i.e., element in the set $f^{-1}(f(x))$) will do. (Indeed, in case f is 1-1, the string x is the only preimage of $f(x)$ under f; but in general there may be other preimages.) It is required that algorithm A' fails (to find a preimage) with overwhelming probability, when the probability is also taken over the input distribution. That is, f is "typically" hard to invert, not merely hard to invert in some ("rare") cases.

Some of the most popular candidates for one-way functions are based on the conjectured intractability of computational problems in number theory. One such conjecture is that it is infeasible to factor large integers. Consequently, the function that takes as input two (equal length) primes and outputs their product is widely believed to be a one-way function. Furthermore, factoring such a composite is infeasible if and only if squaring modulo such a composite is a one-way function (see [113]). For certain composites (i.e., products of two primes that are both congruent to 3 mod 4), the latter function induces a permutation over the set of quadratic residues modulo this composite. A related permutation, which is widely believed to be one-way, is the RSA function [117]: $x \mapsto x^e \bmod N$, where $N = P \cdot Q$ is a composite as above, e is relatively prime to $(P-1) \cdot (Q-1)$, and $x \in \{0, ..., N-1\}$. The latter examples (as well as other popular suggestions) are better captured by the following formulation of a collection of one-way functions (which is indeed related to Definition 2.1):

Definition 2.2 (collections of one-way functions): *A collection of functions, $\{f_i : D_i \to \{0,1\}^*\}_{i \in \overline{I}}$, is called* one-way *if there exists three probabilistic polynomial-time algorithms, I, D and F, so that the following two conditions hold*

1. easy to sample and compute: *On input* 1^n*, the output of* (the index selection) *algorithm* I *is distributed over the set* $\overline{I} \cap \{0,1\}^n$ *(i.e., is an* n*-bit long index of some function). On input (an index of a function)* $i \in \overline{I}$*, the output of* (the domain sampling) *algorithm* D *is distributed over the set* D_i *(i.e., over the domain of the function). On input* $i \in \overline{I}$ *and* $x \in D_i$*,* (the evaluation) *algorithm* F always *outputs* $f_i(x)$*.*

2. hard to invert:[1] *For every probabilistic polynomial-time algorithm,* A'*, every positive polynomial* $p(\cdot)$*, and all sufficiently large* n*'s*

$$\Pr\left[A'(i, f_i(x)) \in f_i^{-1}(f_i(x))\right] < \frac{1}{p(n)}$$

   where $i \leftarrow I(1^n)$ *and* $x \leftarrow D(i)$*.*

The collection is said to be a collection of permutations *if each of the* f_i*'s is a permutation over the corresponding* D_i*, and* $D(i)$ *is almost uniformly distributed in* D_i*.*

For example, in case of the RSA, $f_{N,e} : D_{N,e} \rightarrow D_{N,e}$ satisfies $f_{N,e}(x) = x^e \bmod N$, where $D_{N,e} = \{0, ..., N-1\}$. Definition 2.2 is also a good starting point for the definition of a trapdoor permutation.[2] Loosely speaking, the latter is a collection of one-way permutations augmented with an efficient algorithm that allows for inverting the permutation when given adequate auxiliary information (called a trapdoor).

Definition 2.3 (trapdoor permutations): *A collection of permutations as in Definition 2.2 is called a* trapdoor permutation *if there are two auxiliary probabilistic polynomial-time algorithms* I' *and* F^{-1} *such that (1) the distribution* $I'(1^n)$ *ranges over pairs of strings so that the first string is distributed as in* $I(1^n)$*, and (2) for every* (i,t) *in the range of* $I'(1^n)$ *and every* $x \in D_i$ *it holds that* $F^{-1}(t, f_i(x)) = x$*.* (That is, t is a trapdoor that allows to invert f_i.)

For example, in case of the RSA, $f_{N,e}$ can be inverted by raising to the power d (modulo $N = P \cdot Q$), where d is the multiplicative inverse of e modulo $(P-1)\cdot(Q-1)$. Indeed, in this case, the trapdoor information is (N, d).

Strong versus Weak One-Way Functions

Recall that the above definitions require that any feasible algorithm *succeeds in inverting* the function *with negligible probability.* A weaker notion only requires that any feasible algorithm *fails to invert* the function *with noticeable probability.* It turns out that the existence of such weak one-way functions implies the existence of strong one-way functions (as defined above). The construction itself is straightforward: one just parses the argument to the new function into sufficiently many blocks, and applies the weak one-way function on the individual blocks. We warn that the hardness of inverting the resulting function is not established by mere "combinatorics" (i.e., considering the relative volume of S^t in U^t, for $S \subset U$, where S represents the set of "easy to invert" images). Specifically, one may *not* assume that the potential inverting algorithm works independently on each block. Indeed this assumption seems reasonable, but we should not assume that the adversary

[1] Note that this condition refers to the distributions $I(1^n)$ and $D(i)$, which are merely required to range over $\overline{I} \cap \{0,1\}^n$ and D_i, respectively. (Typically, the distributions $I(1^n)$ and $D(i)$ are (almost) uniform over $\overline{I} \cap \{0,1\}^n$ and D_i, respectively.)

[2] Indeed, a more adequate term would be a collection of trapdoor permutations, but the shorter (and less precise) term is the commonly used one.

behaves in a reasonable way (unless we can actually prove that it gains nothing by behaving in other ways, which seem unreasonable to us).

The hardness of inverting the resulting function is proved via a so called "reducibility argument" (which is used to prove all conditional results in the area). Specifically, we show that any algorithm that inverts the resulting function F with non-negligible success probability can be used to construct an algorithm that inverts the original function f with success probability that violates the hypothesis (regarding f). In other words, we reduce the task of "strongly inverting" f (i.e., violating its weak one-wayness) to the task of "weakly inverting" F (i.e., violating its strong one-wayness). We hint that, on input $y = f(x)$, the reduction invokes the F-inverter (polynomially) many times, each time feeding it with a sequence of random f-images that contains y at a random location. (Indeed such a sequence corresponds to a random image of F.) The analysis of this reduction, presented in [67, Sec. 2.3], demonstrates that dealing with computational difficulty is much more involved than the analogous combinatorial question. An alternative demonstration of the difficulty of reasoning about computational difficulty (in comparison to an analogous purely probabilistic situation) is provided in the proof of Theorem 2.5.

2.2 Hard-Core Predicates

Loosely speaking, saying that a function f is one-way implies that given y (in the range of f) it is infeasible to find a preimage of y under f. This does not mean that it is infeasible to find out partial information about the preimage(s) of y under f. Specifically it may be easy to retrieve half of the bits of the preimage (e.g., given a one-way function f consider the function g defined by $g(x,r) \stackrel{\rm def}{=} (f(x),r)$, for every $|x|=|r|$). As will become clear in subsequent sections, hiding partial information (about the function's preimage) plays an important role in more advanced constructs (e.g., secure encryption). Thus, we will first show how to transform any one-way function into a one-way function that hides specific partial information about its preimage, where this partial information is easy to compute from the preimage itself. This partial information can be considered as a "hard core" of the difficulty of inverting f. Loosely speaking, a *polynomial-time computable* (Boolean) predicate b, is called a hard-core of a function f if no feasible algorithm, given $f(x)$, can guess $b(x)$ with success probability that is non-negligibly better than one half.

Definition 2.4 (hard-core predicates [32]): *A polynomial-time computable predicate $b : \{0,1\}^* \to \{0,1\}$ is called a* hard-core *of a function f if for every probabilistic polynomial-time algorithm A', every positive polynomial $p(\cdot)$, and all sufficiently large n's*

$$\Pr\left[A'(f(x))=b(x)\right] < \frac{1}{2} + \frac{1}{p(n)}$$

where the probability is taken uniformly over all the possible choices of $x \in \{0,1\}^n$ and all the possible outcomes of the internal coin tosses of algorithm A'.

Note that for every $b : \{0,1\}^* \to \{0,1\}$ and $f : \{0,1\}^* \to \{0,1\}^*$, there exist obvious algorithms that guess $b(x)$ from $f(x)$ with success probability at least one half (e.g., the algorithm that, obliviously of its input, outputs a uniformly chosen bit). Also, if b is a hard-core predicate (for any function) then it follows that b is almost unbiased (i.e., for a uniformly chosen x, the difference $|\Pr[b(x)=0] - \Pr[b(x)=1]|$ must be a negligible function in n). Finally, if b is a hard-core of a 1-1 function f that is polynomial-time computable then f is a one-way function.

Theorem 2.5 ([74], see simpler proof in [67, Sec. 2.5.2]): *For any one-way function f, the inner-product mod 2 of x and r is a hard-core of $f'(x,r) = (f(x),r)$.*

The proof is by a so-called "reducibility argument" (which is used to prove all conditional results in the area). Specifically, we reduce the task of inverting f to the task of predicting the hard-core of f', while making sure that the reduction (when applied to input distributed as in the inverting task) generates a distribution as in the definition of the predicting task. Thus, a contradiction to the claim that b is a hard-core of f' yields a contradiction to the hypothesis that f is hard to invert. We stress that this argument is far more complex than analyzing the corresponding "probabilistic" situation (i.e., the distribution of the inner-product mod 2 of X and r, conditioned on a uniformly selected $r \in \{0,1\}^n$, where X is a random variable with super-logarithmic min-entropy, which represents the "effective" knowledge of x, when given $f(x)$).[3]

Proof sketch: The actual proof refers to an arbitrary algorithm B that, when given $(f(x), r)$, tries to guess $b(x,r)$. Suppose that this algorithm succeeds with probability $\frac{1}{2}+\epsilon$, where the probability is taken over the random choices of x and r (as well as the internal coin tosses of B). By an averaging argument, we first identify a $\epsilon/2$ fraction of the possible coin tosses of B such that using any of these coin sequences B succeeds with probability at least $\frac{1}{2}+\epsilon/2$. Similarly, we can identify a $\epsilon/4$ fraction of the x's such that B succeeds (in guessing $b(x,r)$) with probability at least $\frac{1}{2}+\epsilon/4$, where now the probability is taken only over the r's. We will show how to use B in order to invert f, on input $f(x)$, provided that x is in the good set (which has density $\epsilon/4$).

As a warm-up, suppose for a moment that, for the aforementioned x's, algorithm B succeeds with probability $p > \frac{3}{4}+1/\mathrm{poly}(|x|)$ (rather than at least $\frac{1}{2}+\epsilon/4$). In this case, retrieving x from $f(x)$ is quite easy: To retrieve the i^{th} bit of x, denoted x_i, we first randomly select $r \in \{0,1\}^{|x|}$, and obtain $B(f(x),r)$ and $B(f(x), r \oplus e^i)$, where $e^i = 0^{i-1}10^{|x|-i}$ and $v \oplus u$ denotes the addition mod 2 of the binary vectors v and u. Note that if both $B(f(x),r) = b(x,r)$ and $B(f(x), r\oplus e^i) = b(x, r\oplus e^i)$ indeed hold, then $B(f(x),r) \oplus B(f(x), r\oplus e^i)$ equals $b(x,r) \oplus b(x, r\oplus e^i) = b(x,e^i) = x_i$. The probability that both $B(f(x),r) = b(x,r)$ and $B(f(x), r\oplus e^i) = b(x, r\oplus e^i)$ hold, for a random r, is at least $1 - 2\cdot(1-p) > \frac{1}{2} + \frac{1}{\mathrm{poly}(|x|)}$. Hence, repeating the above procedure sufficiently many times (using independent random choices of such r's) and ruling by majority, we retrieve x_i with very high probability. Similarly, we can retrieve all the bits of x, and hence invert f on $f(x)$. However, the entire analysis was conducted under (the unjustifiable) assumption that $p > \frac{3}{4}+\frac{1}{\mathrm{poly}(|x|)}$, whereas we only know that $p > \frac{1}{2}+\frac{\epsilon}{4}$ (for $\epsilon > 1/\mathrm{poly}(|x|)$).

The problem with the above procedure is that it doubles the original error probability of algorithm B on inputs of the form $(f(x), \cdot)$. Under the unrealistic assumption (made above), that B's average error on such inputs is non-negligibly smaller than $\frac{1}{4}$, the "error-doubling" phenomenon raises no problems. However, in general (and even in the special case where B's error is exactly $\frac{1}{4}$) the above procedure is unlikely to invert f. Note that the *average* error probability of B (for a fixed $f(x)$, when the average is taken over a random r) can not be decreased by repeating B several times (e.g., for every x, it may be that B always answer correctly on three quarters of the pairs $(f(x), r)$, and always err on the remaining quarter). What is required is an *alternative way of using* the algorithm B, a way that does not double the original error probability of B.

The key idea is to generate the r's in a way that allows to apply algorithm B only once per each r (and i), instead of twice. Specifically, we will use algorithm B to obtain a "guess" for $b(x, r\oplus e^i)$, and obtain $b(x,r)$ in a different way (which does not use B). The good news is that the error probability is no longer doubled, since we only use B to get a "guess" of $b(x, r\oplus e^i)$. The bad news

[3] The min-entropy of X is defined as $\min_v\{\log_2(1/\Pr[X=v])\}$; that is, if X has min-entropy m then $\max_v\{\Pr[X=v]\} = 2^{-m}$. The Leftover Hashing Lemma [125, 23, 89] implies that, in this case, $\Pr[b(X,U_n)=1|U_n] = \frac{1}{2} \pm 2^{-\Omega(m)}$, where U_n denotes the uniform distribution over $\{0,1\}^n$, and $b(u,v)$ denotes the inner-product mod 2 of u and v.

is that we still need to know $b(x,r)$, and it is not clear how we can know $b(x,r)$ without applying B. The answer is that we can guess $b(x,r)$ by ourselves. This is fine if we only need to guess $b(x,r)$ for one r (or logarithmically in $|x|$ many r's), but the problem is that we need to know (and hence guess) the value of $b(x,r)$ for polynomially many r's. The obvious way of guessing these $b(x,r)$'s yields an exponentially small success probability. Instead, we generate these polynomially many r's such that, on one hand they are "sufficiently random" whereas, on the other hand, we can guess all the $b(x,r)$'s with noticeable success probability.[4] Specifically, generating the r's in a specific *pairwise independent* manner will satisfy both (seemingly contradictory) requirements. We stress that in case we are successful (in our guesses for all the $b(x,r)$'s), we can retrieve x with high probability. Hence, we retrieve x with noticeable probability.

A word about the way in which the pairwise independent r's are generated (and the corresponding $b(x,r)$'s are guessed) is indeed in place. To generate $m = \text{poly}(|x|)$ many r's, we uniformly (and independently) select $\ell \stackrel{\text{def}}{=} \log_2(m+1)$ strings in $\{0,1\}^{|x|}$. Let us denote these strings by $s^1, ..., s^\ell$. We then guess $b(x,s^1)$ through $b(x,s^\ell)$. Let us denote these guesses, which are uniformly (and independently) chosen in $\{0,1\}$, by σ^1 through σ^ℓ. Hence, the probability that all our guesses for the $b(x,s^i)$'s are correct is $2^{-\ell} = \frac{1}{\text{poly}(|x|)}$. The different r's correspond to the different *non-empty* subsets of $\{1,2,...,\ell\}$. Specifically, for every such subset J, we let $r^J \stackrel{\text{def}}{=} \oplus_{j\in J} s^j$. The reader can easily verify that the r^J's are pairwise independent and each is uniformly distributed in $\{0,1\}^{|x|}$. The key observation is that $b(x,r^J) = b(x, \oplus_{j\in J} s^j) = \oplus_{j\in J} b(x,s^j)$. Hence, our guess for $b(x,r^J)$ is $\oplus_{j\in J}\sigma^j$, and with noticeable probability all our guesses are correct. ■

3 Pseudorandomness

In practice "pseudorandom" sequences are often used instead of truly random sequences. The underlying belief is that if an (efficient) application performs well when using a truly random sequence then it will perform essentially as well when using a "pseudorandom" sequence. However, this belief is not supported by ad-hoc notions of "pseudorandomness" such as passing the statistical tests in [95] or having large linear-complexity (as in [85]). In contrast, the above belief is an easy corollary of defining pseudorandom distributions as ones that are computationally indistinguishable from uniform distributions.

Loosely speaking, pseudorandom generators are efficient procedures for creating long "random-looking" sequences based on few truly random bits (i.e., a short random seed). The relevance of such constructs to cryptography is in the ability of legitimate users that share short random seeds to create large objects that look random to any feasible adversary (which does not know the said seed).

3.1 Computational Indistinguishability

Indistinguishable things are identical
(or should be considered as identical).

The Principle of Identity of Indiscernibles
G.W. Leibniz (1646–1714)

(Leibniz admits that counterexamples to this principle are conceivable, but will not occur in real life because God is much too benevolent.)

[4]Alternatively, we can try all polynomially many possible guesses.

A central notion in Modern Cryptography is that of "effective similarity" (introduced by Goldwasser, Micali and Yao [82, 128]). The underlying thesis is that we do not care whether or not objects are equal, all we care about is whether or not a difference between the objects can be observed by a feasible computation. In case the answer is negative, the two objects are equivalent as far as any practical application is concerned. Indeed, in the sequel we will often interchange such (computationally indistinguishable) objects. Let $X = \{X_n\}_{n \in \mathbb{N}}$ and $Y = \{Y_n\}_{n \in \mathbb{N}}$ be probability ensembles such that each X_n and Y_n is a distribution that ranges over strings of length n (or polynomial in n). We say that X and Y are computationally indistinguishable if for every feasible algorithm A the difference $d_A(n) \stackrel{\text{def}}{=} |\Pr[A(X_n)=1] - \Pr[A(Y_n)=1]|$ is a negligible function in n. That is:

Definition 3.1 (computational indistinguishability [82, 128]): *We say that $X = \{X_n\}_{n \in \mathbb{N}}$ and $Y = \{Y_n\}_{n \in \mathbb{N}}$ are* computationally indistinguishable *if for every probabilistic polynomial-time algorithm D every polynomial p, and all sufficiently large n,*

$$|\Pr[D(X_n)=1] - \Pr[D(Y_n)=1]| \ < \ \frac{1}{p(n)}$$

where the probabilities are taken over the relevant distribution (i.e., either X_n or Y_n) *and over the internal coin tosses of algorithm D.*

That is, we can think of D as somebody who wishes to distinguish two distributions (based on a sample given to it), and think of 1 as D's verdict that the sample was drawn according to the first distribution. Saying that the two distributions are computationally indistinguishable means that if D is a feasible procedure then its verdict is not really meaningful (because the verdict is almost as often 1 when the input is drawn from the first distribution as when the input is drawn from the second distribution).

Indistinguishability by Multiple Samples

We comment that, for "efficiently constructible" distributions, indistinguishability by a single sample (as defined above) implies indistinguishability by multiple samples (see [67, Sec. 3.2.3]). The proof of this fact provides a simple demonstration of a central proof technique, known as a *hybrid argument*, which we briefly present next.

To prove that a sequence of independently drawn samples of one distribution is indistinguishable from a sequence of independently drawn samples from the other distribution, we consider *hybrid* sequences such that the i^{th} hybrid consists of i samples taken from the first distribution and the rest taken from the second distribution. The "homogeneous" sequences (which we wish to prove to be computational indistinguishable) are the extreme hybrids (i.e., the first and last hybrids considered above). The key observation is that distinguishing the extreme hybrids (towards the contradiction hypothesis) yields a procedure for distinguishing single samples of the two distributions (contradicting the hypothesis that the two distributions are indistinguishable by a single sample). Specifically, if D distinguishes the extreme hybrids, then it also distinguishes a random pair of neighboring hybrids (i.e., D distinguishes the i^{th} hybrid from the $i + 1^{\text{st}}$ hybrid, for a randomly selected i). Using D, we obtain a distinguisher D' of single samples: Given a single sample, D' selects i at random, generates i samples from the first distribution and the rest from the second, and invokes D with the corresponding sequence, while placing the input sample in location $i+1$ of the sequence. We stress that although the original distinguisher D (arising from the contradiction hypothesis) was only "supposed to work" for the extreme hybrids, we can consider D's performance on any distribution that we please, and draw adequate conclusions (as we have done).

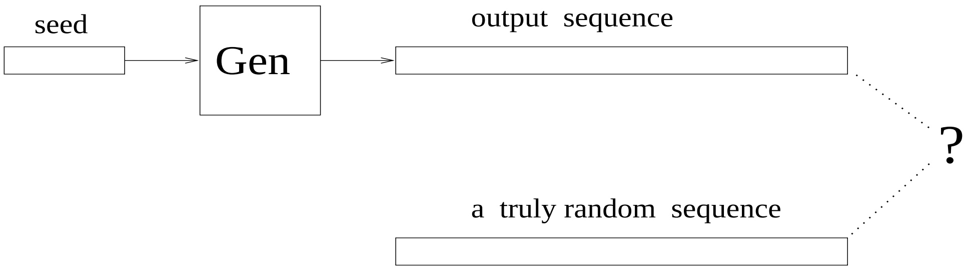


Figure 3: Pseudorandom generators – an illustration.

3.2 Pseudorandom Generators

Loosely speaking, a pseudorandom generator is an efficient (deterministic) algorithm that on input a short random *seed* outputs a (typically much) longer sequence that is computationally indistinguishable from a uniformly chosen sequence. Pseudorandom generators were introduced by Blum, Micali and Yao [32, 128], and are formally defined as follows.

Definition 3.2 (pseudorandom generator [32, 128]): *Let $\ell:\mathbb{N}\to\mathbb{N}$ satisfy $\ell(n) > n$, for all $n \in \mathbb{N}$. A* pseudorandom generator*, with* stretch function *ℓ, is a* (deterministic) *polynomial-time algorithm G satisfying the following:*

1. *For every $s \in \{0,1\}^*$, it holds that $|G(s)| = \ell(|s|)$.*
2. *$\{G(U_n)\}_{n\in\mathbb{N}}$ and $\{U_{\ell(n)}\}_{n\in\mathbb{N}}$ are computationally indistinguishable, where U_m denotes the uniform distribution over $\{0,1\}^m$.*

Indeed, the probability ensemble $\{G(U_n)\}_{n\in\mathbb{N}}$ is called pseudorandom.

Thus, pseudorandom sequences can replace truly random sequences not only in "standard" algorithmic applications but also in cryptographic ones. That is, *any* cryptographic application that is secure when the legitimate parties use truly random sequences, is also secure when the legitimate parties use pseudorandom sequences. The benefit in such a substitution (of random sequences by pseudorandom ones) is that the latter sequences can be efficiently generated using much less true randomness. Furthermore, *in an interactive setting*, it is possible to eliminate all random steps from the on-line execution of a program, by replacing them with the generation of pseudorandom bits based on a random seed selected and fixed off-line (or at set-up time).

Various cryptographic applications of pseudorandom generators will be presented in the sequel, but first let us show a construction of pseudorandom generators based on the simpler notion of a one-way function. Using Theorem 2.5, we may actually assume that such a function is accompanied by a hard-core predicate. We start with a simple construction that suffices for the case of 1-1 (and length-preserving) functions.

Theorem 3.3 ([32, 128], see [67, Sec. 3.4]): *Let f be a 1-1 function that is length-preserving and efficiently computable, and b be a hard-core predicate of f. Then $G(s) = b(s)\cdot b(f(s))\cdots b(f^{\ell(|s|)-1}(s))$ is a pseudorandom generator (with stretch function ℓ), where $f^{i+1}(x) \stackrel{\rm def}{=} f(f^i(x))$ and $f^0(x) \stackrel{\rm def}{=} x$.*

As a concrete example, consider the permutation[5] $x \mapsto x^2 \bmod N$, where N is the product of two primes each congruent to $3 \pmod 4$, and x is a quadratic residue modulo N. Then, we have

[5] It is a well-known fact (cf., [67, Apdx. A.2.4]) that, for such N's, the mapping $x \mapsto x^2 \bmod N$ is a permutation over the set of quadratic residues modulo N.

$G_N(s) = \text{lsb}(s) \cdot \text{lsb}(s^2 \bmod N) \cdots \text{lsb}(s^{2^{\ell(|s|)-1}} \bmod N)$, where $\text{lsb}(x)$ is the least significant bit of x (which is a hard-core of the modular squaring function [2]).

Proof sketch of Theorem 3.3: We use the fundamental fact that asserts that the following two conditions are equivalent:

1. The distribution X (in our case $\{G(U_n)\}_{n\in\mathbb{N}}$) is pseudorandom (i.e., is computationally indistinguishable from a uniform distribution (on $\{U_{\ell(n)}\}_{n\in\mathbb{N}}$)).
2. The distribution X is unpredictable in polynomial-time; that is, no feasible algorithm, given a prefix of the sequence, can guess its next bit with a non-negligible advantage over $\frac{1}{2}$.

Clearly, pseudorandomness implies polynomial-time unpredictability (i.e., polynomial-time predictability violates pseudorandomness). The converse is shown using a hybrid argument, which refers to hybrids consisting of a prefix of X followed by truly random bits (i.e., a suffix of the uniform distribution). Thus, we focus on proving that $G'(U_n)$ is polynomial-time unpredictable, where $G'(s) = b(f^{\ell(|s|)-1}(s)) \cdots b(f(s)) \cdot b(s)$ is the reverse of $G(s)$.

Suppose towards the contradiction that, for some $j < \ell \stackrel{\text{def}}{=} \ell(n)$, given the j-bit long prefix of $G'(U_n)$ an algorithm A' can predict the $j+1^{\text{st}}$ bit of $G'(U_n)$. That is, given $b(f^{\ell-1}(s)) \cdots b(f^{\ell-j}(s))$, algorithm A' predicts $b(f^{\ell-(j+1)}(s))$, where s is uniformly distributed in $\{0,1\}^n$. Then, for x uniformly distributed in $\{0,1\}^n$, given $y = f(x)$ one can predict $b(x)$ by invoking A' on input $b(f^{j-1}(y)) \cdots b(y) = b(f^j(x)) \cdots b(f(x))$, which in turn is polynomial-time computable from $y = f(x)$. In the analysis, we use the hypothesis that f induces a permutation over $\{0,1\}^n$, and associate x with $f^{\ell-(j+1)}(s)$. ■

We mention that the existence of a pseudorandom generator with any stretch function (including the very minimal stretch function $\ell(n) = n+1$) implies the existence of pseudorandom generators for any desired stretch function. The construction is similar to the one presented in Theorem 3.3. That is, for a pseudorandom generator G_1, let $F(x)$ (resp., $B(x)$) denote the first $|x|$ bits of $G_1(x)$ (resp., the $|x|+1^{\text{st}}$ bit of $G_1(x)$), and consider $G(s) = B(s) \cdot B(F(s)) \cdots B(F^{\ell(|s|)-1}(s))$, where ℓ is the desired stretch. Although F is not necessarily 1-1, it can be shown that G is a pseudorandom generator [67, Sec. 3.3.2].

We conclude this section by mentioning that pseudorandom generators can be constructed from *any* one-way function (rather than merely from one-way permutations, as above). On the other hand, the existence of one-way functions is a necessary condition to the existence of pseudorandom generators. That is:

Theorem 3.4 [87]: *Pseudorandom generators exist if and only if one-way functions exist.*

The necessary condition is easy to establish. Given a pseudorandom generator G that stretches by a factor of two, consider the function $f(x) = G(x)$ (or, to obtain a length preserving-function, let $f(x,y) = G(x)$, where $|x| = |y|$). An algorithm that inverts f with non-negligible success probability (on the distribution $f(U_n) = G(U_n)$) yields a distinguisher of $\{G(U_n)\}_{n\in\mathbb{N}}$ from $\{U_{2n}\}_{n\in\mathbb{N}}$, because the probability that U_{2n} is an image of f is negligible.

3.3 Pseudorandom Functions

Pseudorandom generators provide a way to efficiently generate long pseudorandom sequences from short random seeds. Pseudorandom functions, introduced and constructed by Goldreich, Goldwasser and Micali [70], are even more powerful: they provide efficient direct access to bits of a huge pseudorandom sequence (which is not feasible to scan bit-by-bit). More precisely, a pseudorandom function is an efficient (deterministic) algorithm that given an n-bit *seed*, s, and an n-bit *argument*, x, returns an n-bit string, denoted $f_s(x)$, so that it is infeasible to distinguish the values of f_s, for a uniformly chosen $s \in \{0,1\}^n$, from the values of a truly random function $F : \{0,1\}^n \to \{0,1\}^n$. That is, the (feasible) testing procedure is given oracle access to the function (but not its explicit description), and cannot distinguish the case it is given oracle access to a pseudorandom function from the case it is given oracle access to a truly random function.

One key feature of the above definition is that pseudorandom functions can be generated and shared by merely generating and sharing their seed; that is, a "random looking" function $f_s : \{0,1\}^n \to \{0,1\}^n$, is determined by its n-bit seed s. Parties wishing to share a "random looking" function f_s (determining 2^n-many values), merely need to generate and share among themselves the n-bit seed s. (For example, one party may randomly select the seed s, and communicate it, via a secure channel, to all other parties.) Sharing a pseudorandom function allows parties to determine (by themselves and without any further communication) random-looking values depending on their current views of the environment (which need not be known a priori). To appreciate the potential of this tool, one should realize that sharing a pseudorandom function is essentially as good as being able to agree, on the fly, on the association of random values to (on-line) given values, where the latter are taken from a huge set of possible values. We stress that this agreement is achieved without communication and synchronization: Whenever some party needs to associate a random value to a given value, $v \in \{0,1\}^n$, it will associate to v the (same) random value $r_v \in \{0,1\}^n$ (by setting $r_v = f_s(v)$, where f_s is a pseudorandom function agreed upon beforehand).

Theorem 3.5 ([70], see [67, Sec. 3.6.2]): *Pseudorandom functions can be constructed using any pseudorandom generator.*

Proof sketch: Let G be a pseudorandom generator that stretches its seed by a factor of two (i.e., $\ell(n) = 2n$), and let $G_0(s)$ (resp., $G_1(s)$) denote the first (resp., last) $|s|$ bits in $G(s)$. Define

$$G_{\sigma_{|s|}\cdots\sigma_2\sigma_1}(s) \stackrel{\text{def}}{=} G_{\sigma_{|s|}}(\cdots G_{\sigma_2}(G_{\sigma_1}(s))\cdots)$$

We consider the function ensemble $\{f_s : \{0,1\}^{|s|} \to \{0,1\}^{|s|}\}_{s\in\{0,1\}^*}$, where $f_s(x) \stackrel{\text{def}}{=} G_x(s)$. Pictorially, the function f_s is defined by n-step walks down a full binary tree of depth n having labels at the vertices. The root of the tree, hereafter referred to as the level 0 vertex of the tree, is labeled by the string s. If an internal vertex is labeled r then its left child is labeled $G_0(r)$ whereas its right child is labeled $G_1(r)$. The value of $f_s(x)$ is the string residing in the leaf reachable from the root by a path corresponding to the string x.

We claim that the function ensemble $\{f_s\}_{s\in\{0,1\}^*}$, defined above, is pseudorandom. The proof uses the hybrid technique: The i^{th} hybrid, H_n^i, is a function ensemble consisting of $2^{2^i \cdot n}$ functions $\{0,1\}^n \to \{0,1\}^n$, each defined by 2^i random n-bit strings, denoted $\overline{s} = \langle s_\beta \rangle_{\beta\in\{0,1\}^i}$. The value of such function $h_{\overline{s}}$ at $x = \alpha\beta$, where $|\beta| = i$, is $G_\alpha(s_\beta)$. (Pictorially, the function $h_{\overline{s}}$ is defined by placing the strings in $\overline{s}$ in the corresponding vertices of level i, and labeling vertices of lower levels using the very rule used in the definition of f_s.) The extreme hybrids correspond to our indistinguishability claim (i.e., $H_n^0 \equiv f_{U_n}$ and H_n^n is a truly random function), and neighboring hybrids can be related to our indistinguishability hypothesis (specifically, to the indistinguishability of $G(U_n)$ and U_{2n} under multiple samples). ■

Useful variants (and generalizations) of the notion of pseudorandom functions include Boolean pseudorandom functions that are defined for all strings (i.e., $f_s : \{0,1\}^* \to \{0,1\}$) and pseudorandom functions that are defined for other domains and ranges (i.e., $f_s : \{0,1\}^{d(|s|)} \to \{0,1\}^{r(|s|)}$, for arbitrary polynomially bounded functions $d, r : \mathbb{N} \to \mathbb{N}$). Various transformations between these variants are known (cf. [67, Sec. 3.6.4] and [68, Apdx. C.2]).

Applications and a generic methodology. Pseudorandom functions are a very useful cryptographic tool: One may first design a cryptographic scheme assuming that the legitimate users have black-box access to a random function, and next implement the random function using a pseudorandom function. The usefulness of this tool stems from the fact that having (black-box) access to a random function gives the legitimate parties a potential advantage over the adversary (which does not have free access to this function).[6] The security of the resulting implementation (which uses a pseudorandom function) is established in two steps: First one proves the security of an idealized scheme that uses a truly random function, and next one argues that the actual implementation (which uses a pseudorandom function) is secure (as otherwise one obtains an efficient oracle machine that distinguishes a pseudorandom function from a truly random one).

4 Zero-Knowledge

Zero-knowledge proofs, introduced by Goldwasser, Micali and Rackoff [83], provide a powerful tool for the design of cryptographic protocols. Loosely speaking, zero-knowledge proofs are proofs that yield nothing beyond the validity of the assertion. That is, a verifier obtaining such a proof only gains conviction in the validity of the assertion (as if it was told by a trusted party that the assertion holds). This is formulated by saying that anything that is feasibly computable from a zero-knowledge proof is also feasibly computable from the (valid) assertion itself. The latter formulation follows the simulation paradigm, which is discussed next.

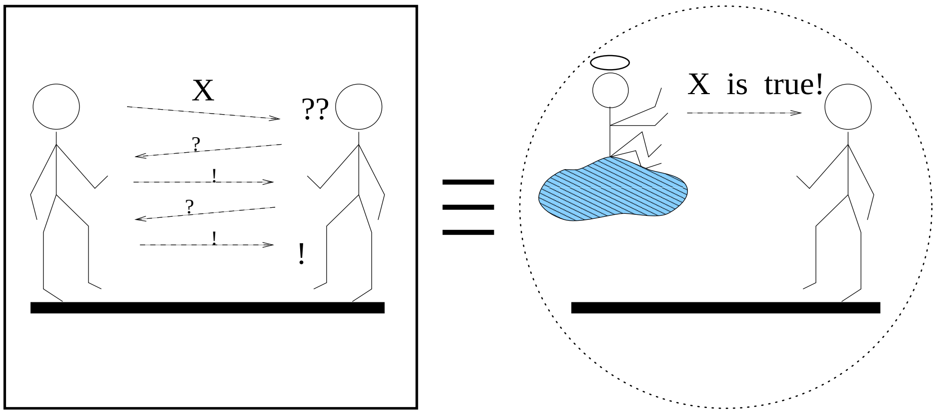


Figure 4: Zero-knowledge proofs – an illustration.

4.1 The Simulation Paradigm

A key question regarding the modeling of security concerns is how to express the intuitive requirement that an adversary "gains nothing substantial" by deviating from the prescribed behavior of

[6]The aforementioned methodology is sound provided that the adversary does not get the description of the pseudorandom function (i.e., the seed) in use, but has only (possibly limited) oracle access to it. This is different from the so-called Random Oracle Methodology formulated in [22] and criticized in [39].

an honest user. Our approach is that the adversary *gains nothing* if whatever it can obtain by unrestricted adversarial behavior can also be obtained within essentially the same computational effort by a benign behavior. The definition of the "benign behavior" captures what we want to achieve in terms of security, and is specific to the security concern to be addressed. For example, in the previous paragraph, we said that a proof is zero-knowledge if it yields nothing (to the adversarial verifier) beyond the validity of the assertion (i.e., the benign behavior is any computation that is based (only) on the assertion itself, while assuming that the latter is valid). Other examples are discussed in Sections 5.1 and 7.1.

A notable property of the aforementioned simulation paradigm, as well as of the entire approach surveyed here, is that this approach is overly liberal with respect to its view of the abilities of the adversary as well as to what might constitute a gain for the adversary. Thus, the approach may be considered overly cautious, because it prohibits also "non-harmful" gains of some "far fetched" adversaries. We warn against this impression. Firstly, there is nothing more dangerous in cryptography than to consider "reasonable" adversaries (a notion which is almost a contradiction in terms): typically, the adversaries will try exactly what the system designer has discarded as "far fetched". Secondly, it seems impossible to come up with definitions of security that distinguish "breaking the scheme in a harmful way" from "breaking it in a non-harmful way": what is harmful is application-dependent, whereas a good definition of security ought to be application-independent (as otherwise using the scheme in any new application will require a full re-evaluation of its security). Furthermore, even with respect to a specific application, it is typically very hard to classify the set of "harmful breakings".

4.2 The Actual Definition

A proof is whatever convinces me.

Shimon Even (1935–2004)

Before defining zero-knowledge proofs, we have to define proofs. The standard notion of a static (i.e., non-interactive) proof will not do, because static zero-knowledge proofs exist only for sets that are easy to decide (i.e, are in $\mathcal{BPP}$) [77], whereas we are interested in zero-knowledge proofs for arbitrary NP-sets. Instead, we use the notion of an interactive proof (introduced exactly for that reason in [83]). That is, here a proof is a (multi-round) randomized protocol for two parties, called verifier and prover, in which the prover wishes to convince the verifier of the validity of a given assertion. Such an interactive proof should allow the prover to convince the verifier of the validity of any true assertion (i.e., completeness), whereas NO prover strategy may fool the verifier to accept false assertions (i.e., soundness). Both the *completeness* and *soundness* conditions should hold with high probability (i.e., a negligible error probability is allowed). The prescribed verifier strategy is required to be efficient. No such requirement is made with respect to the prover strategy; yet we will be interested in "relatively efficient" prover strategies (see below).[7]

Zero-knowledge is a property of some prover strategies. More generally, we consider interactive machines that yield no knowledge while interacting with an arbitrary feasible adversary on a

[7] We stress that the relative efficiency of the prover strategy refers to the strategy employed in order to prove valid assertions; that is, relative efficiency of the prover strategy is a *strengthening* of the completeness condition (which is indeed *required* for cryptographic applications). This should not be confused with the relaxation (i.e., weakening) of the soundness condition that restricts its scope to feasible adversarial prover strategies (rather than to all possible prover strategies). The resulting notion of "computational soundness" is discussed in Section 4.4.1, and indeed *suffices* in most cryptographic applications. Still, we believe that it is simpler to present the material in terms of interactive proofs (rather than in terms of computationally sound proofs).

common input taken from a predetermined set (in our case the set of valid assertions). A strategy A is zero-knowledge on (inputs from) the set S if, for every feasible strategy B^*, there exists a feasible computation C^* such that the following two probability ensembles are computationally indistinguishable[8]:

1. $\{(A, B^*)(x)\}_{x \in S} \stackrel{\text{def}}{=}$ the output of B^* after interacting with A on common input $x \in S$; and

2. $\{C^*(x)\}_{x \in S} \stackrel{\text{def}}{=}$ the output of C^* on input $x \in S$.

We stress that the first ensemble represents an actual execution of an interactive protocol, whereas the second ensemble represents the computation of a stand-alone procedure (called the "simulator"), which does not interact with anybody.

The above definition does *not* account for auxiliary information that an adversary B^* may have prior to entering the interaction. Accounting for such auxiliary information is essential for using zero-knowledge proofs as subprotocols inside larger protocols (see [73, 77]). This is taken care of by a stricter notion called auxiliary-input zero-knowledge.

Definition 4.1 (zero-knowledge [83], revisited [77]): *A strategy A is* auxiliary-input zero-knowledge *on inputs from S if, for every probabilistic polynomial-time strategy B^* and every polynomial p, there exists a probabilistic polynomial-time algorithm C^* such that the following two probability ensembles are computationally indistinguishable:*

1. $\{(A, B^*(z))(x)\}_{x \in S\,,\, z \in \{0,1\}^{p(|x|)}} \stackrel{\text{def}}{=}$ *the output of B^* when having auxiliary-input z and interacting with A on common input $x \in S$; and*

2. $\{C^*(x, z)\}_{x \in S\,,\, z \in \{0,1\}^{p(|x|)}} \stackrel{\text{def}}{=}$ *the output of C^* on inputs $x \in S$ and $z \in \{0,1\}^{p(|x|)}$.*

Almost all known zero-knowledge proofs are in fact auxiliary-input zero-knowledge. As hinted above, *auxiliary-input zero-knowledge is preserved under sequential composition* [77]. A simulator for the multiple-session protocol can be constructed by iteratively invoking the single-session simulator that refers to the residual strategy of the adversarial verifier in the given session (while feeding this simulator with the transcript of previous sessions). Indeed, the residual single-session verifier gets the transcript of the previous sessions as part of its auxiliary input (i.e., z in Definition 4.1). (For details, see [67, Sec. 4.3.4].)

4.3 Zero-Knowledge Proofs for all NP-assertions and their applications

A question avoided so far is whether zero-knowledge proofs exist at all. Clearly, every set in $\mathcal{P}$ (or rather in $\mathcal{BPP}$) has a "trivial" zero-knowledge proof (in which the verifier determines membership by itself); however, what we seek is zero-knowledge proofs for statements that the verifier cannot decide by itself.

[8] Here we refer to a natural extension of Definition 3.1: Rather than referring to ensembles indexed by $\mathbb{N}$, we refer to ensembles indexed by a set $S \subseteq \{0,1\}^*$. Typically, for an ensemble $\{Z_\alpha\}_{\alpha \in S}$, it holds that Z_α ranges over strings of length that is polynomially-related to the length of α. We say that $\{X_\alpha\}_{\alpha \in S}$ and $\{Y_\alpha\}_{\alpha \in S}$ are computationally indistinguishable if for every probabilistic polynomial-time algorithm D every polynomial p, and all sufficiently long $\alpha \in S$,

$$|\Pr[D(\alpha, X_\alpha) = 1] - \Pr[D(\alpha, Y_\alpha) = 1]| < \frac{1}{p(|\alpha|)}$$

where the probabilities are taken over the relevant distribution (i.e., either X_α or Y_α) and over the internal coin tosses of algorithm D.

Assuming the existence of "commitment schemes" (see below), which in turn exist if one-way functions exist [104, 87], *there exist* (auxiliary-input) *zero-knowledge proofs of membership in any NP-set* (i.e., sets having efficiently verifiable static proofs of membership). These zero-knowledge proofs, first constructed by Goldreich, Micali and Wigderson [75] (and depicted in Figure 5), have the following important property: the prescribed prover strategy is efficient, provided it is given as auxiliary-input an NP-witness to the assertion (to be proved).[9] That is:

Theorem 4.2 ([75], using [87, 104]): *If one-way functions exist then every set $S \in \mathcal{NP}$ has a zero-knowledge interactive proof. Furthermore, the prescribed prover strategy can be implemented in probabilistic polynomial-time, provided it is given as auxiliary-input an NP-witness for membership of the common input in S.*

Theorem 4.2 makes zero-knowledge a very powerful tool in the design of cryptographic schemes and protocols (see below). We comment that the intractability assumption used in Theorem 4.2 seems essential; see [109, 127].

Commitment schemes are digital analogs of sealed envelopes (or, better, locked boxes). Sending a commitment means sending a string that binds the sender to a unique value without revealing this value to the receiver (as when getting a locked box). Decommitting to the value means sending some auxiliary information that allows the receiver to read the uniquely committed value (as when sending the key to the lock).

Common Input: A graph $G(V, E)$. Suppose that $V \equiv \{1, ..., n\}$ for $n \stackrel{\text{def}}{=} |V|$.

Auxiliary Input (to the prover): A 3-coloring $\phi : V \to \{1, 2, 3\}$.

The following 4 steps are repeated $t \cdot |E|$ many times so to obtain soundness error $\exp(-t)$.

Prover's first step (P1): Select uniformly a permutation π over $\{1, 2, 3\}$. For $i = 1$ to n, send the verifier a commitment to the value $\pi(\phi(i))$.

Verifier's first step (V1): Select uniformly an edge $e \in E$ and send it to the prover.

Prover's second step (P2): Upon receiving $e = (i, j) \in E$, decommit to the i-th and j-th values sent in Step (P1).

Verifier's second step (V2): Check whether or not the decommitted values are different elements of $\{1, 2, 3\}$ and whether or not they match the commitments received in Step (P1).

Figure 5: The zero-knowledge proof of Graph 3-Colorability (of [75]). Zero-knowledge proofs for other NP-sets can be obtained using the standard reductions.

Analyzing the protocol of Figure 5. Let us consider a single execution of the main loop (and rely on the preservation of zero-knowledge under sequential composition). Clearly, the prescribed prover is implemented in probabilistic polynomial-time, and always convinces the verifier (provided that it is given a valid 3-coloring of the common input graph). In case the graph is not 3-colorable then, no matter how the prover behaves, the verifier will reject with probability at least $1/|E|$

[9] The auxiliary-input given to the prescribed prover (in order to allow for an efficient implementation of its strategy) is not to be confused with the auxiliary-input that is given to malicious verifiers (in the definition of auxiliary-input zero-knowledge). The former is typically an NP-witness for the common input, which is available to the user that invokes the prover strategy (cf. the generic application discussed below). In contrast, the auxiliary-input that is given to malicious verifiers models arbitrary partial information that may be available to the adversary.

(because at least one of the edges must be improperly colored by the prover). We stress that the verifier selects uniformly which edge to inspect after the prover has committed to the colors of all vertices. Thus, Figure 5 depicts an interactive proof system for Graph 3-Colorability (with error probability $\exp(-t)$). As the reader might have guessed, the zero-knowledge property is the hardest to establish, and we will confine ourselves to presenting an adequate simulator. We start with three simplifying conventions (which are useful in general):

1. Without loss of generality, we may assume that the cheating verifier strategy is implemented by a *deterministic* polynomial-time algorithm with an auxiliary input. This is justified by fixing any outcome of the verifier's coins (as part of the auxiliary input), and observing that our (uniform) simulation of the various (residual) deterministic strategies yields a simulation of the original probabilistic strategy.

2. Without loss of generality, it suffices to consider cheating verifiers that (only) output their view of the interaction (i.e., their input, coin tosses, and the messages they received). In other words, it suffices to simulate the view of the cheating verifier rather than its output (which is the result of a polynomial-time post-processing of the view).

3. Without loss of generality, it suffices to construct a "weak simulator" that produces an output with some noticeable probability, provided that (conditioned on producing output) the output is computationally indistinguishable from the desired distribution (i.e., the view of the cheating verifier in a real interaction). This is the case because, by repeatedly invoking this weak simulator (polynomially) many times, we may obtain a simulator that fails to produce an output with negligible probability. Finally, letting the simulator produce an arbitrary output rather than failing, we obtain a simulator that never fails (as required by the definition), while skewing the output distribution by at most a negligible amount.

Our simulator starts by selecting uniformly and independently a random color (i.e., element of $\{1,2,3\}$) for each vertex, and feeding the verifier strategy with random commitments to these random colors. Indeed, the simulator feeds the verifier with a distribution that is very different from the distribution that the verifier sees in a real interaction with the prover. However, being computationally-restricted the verifier cannot tell these distributions apart (or else we obtain a contradiction to the security of the commitment scheme in use). Now, if the verifier asks to inspect an edge that is properly colored then the simulator performs the proper decommitment action and outputs the transcript of this interaction. Otherwise, the simulator halts proclaiming failure. We claim that failure occurs with probability approximately $1/3$ (or else we obtain a contradiction to the security of the commitment scheme in use). Furthermore, based on the same hypothesis (but via a more complex proof (cf. [67, Sec. 4.4.2.3])), conditioned on not failing, the output of the simulator is computationally indistinguishable from the verifier's view of the real interaction.

Commitment schemes. Loosely speaking, commitment schemes are two-stage (two-party) protocols allowing for one party to commit itself (at the first stage) to a value while keeping the value secret. In a (second) latter stage, the commitment is "opened" and it is guaranteed that the "opening" can yield only a single value determined in the committing phase. Thus, the (first stage of the) commitment scheme is both *binding* and *hiding*. A simple (uni-directional communication) commitment scheme can be constructed based on any one-way 1-1 function f (with a corresponding hard-core b). To commit to a bit σ, the sender uniformly selects $s \in \{0,1\}^n$, and sends the pair $(f(s), b(s) \oplus \sigma)$. Note that this is both binding and hiding. An alternative construction, which can be based on any one-way function, uses a pseudorandom generator G that stretches its seed by a factor of three (cf. Theorem 3.4). A commitment is established, via two-way communication, as follows (cf. [104]): The receiver selects uniformly $r \in \{0,1\}^{3n}$ and sends it to the sender, which selects uniformly $s \in \{0,1\}^n$ and sends $r \oplus G(s)$ if it wishes to commit to the value one and $G(s)$ if it wishes to commit to zero. To see that this is binding, observe that there are at most 2^{2n} "bad" values r that satisfy $G(s_0) = r \oplus G(s_1)$ for some pair (s_0, s_1), and with overwhelmingly high probability the receiver will not pick one of these bad values. The hiding property follows by the pseudorandomness of G.

Zero-knowledge proofs for other NP-sets. By using the standard Karp-reductions to 3-Colorability, the protocol of Figure 5 can be used for constructing zero-knowledge proofs for any set in $\mathcal{NP}$. We comment that this is probably the first time that an NP-completeness result was used in a "positive" way (i.e., in order to construct something rather than in order to derive a hardness result).[10]

Efficiency considerations. The protocol in Figure 5 calls for invoking some constant-round protocol for a non-constant number of times (and its analysis relies on the preservation of zero-knowledge under sequential composition). At first glance, it seems that one can derive a constant-round zero-knowledge proof system (of negligible soundness error) by performing these invocations in parallel (rather than sequentially). Unfortunately, as indicated in [73], it is not clear that the resulting interactive proof is zero-knowledge. Still, under standard intractability assumptions (e.g., the intractability of factoring), constant-round zero-knowledge proofs (of negligible soundness error) do exist for every set in $\mathcal{NP}$ (cf. [72]). We comment that the number of rounds in a protocol is commonly considered the most important efficiency criterion (or complexity measure), and typically one desires to have it be a constant.

A generic application. As mentioned above, Theorem 4.2 makes zero-knowledge a very powerful tool in the design of cryptographic schemes and protocols. This wide applicability is due to two important aspects regarding Theorem 4.2: Firstly, Theorem 4.2 provides a zero-knowledge proof for every NP-set, and secondly the prescribed prover can be implemented in probabilistic polynomial-time when given an adequate NP-witness. We now turn to a typical application of zero-knowledge proofs. In a typical cryptographic setting, a user U has a secret and is supposed to take some action depending on its secret. The question is how can other users verify that U indeed took the correct action (as determined by U's secret and publicly known information). Indeed, if U discloses its secret then anybody can verify that U took the correct action. However, U does not want to reveal its secret. Using zero-knowledge proofs we can satisfy both conflicting requirements (i.e., having other users verify that U took the correct action without violating U's interest in not revealing its secret). That is, U can prove in zero-knowledge that it took the correct action. Note that U's claim to having taken the correct action is an NP-assertion (since U's legal action is determined as a polynomial-time function of its secret and the public information), and that U has an NP-witness to its validity (i.e., the secret is an NP-witness to the claim that the action fits the public information). Thus, by Theorem 4.2, it is possible for U to efficiently prove the correctness of its action without yielding anything about its secret. Consequently, it is fair to ask U to prove (in zero-knowledge) that it behaves properly, and so to force U to behave properly. Indeed, "forcing proper behavior" is the canonical application of zero-knowledge proofs (see [76, 65]).

[10]Subsequent positive uses of completeness results have appeared in the context of interactive proofs [99, 123], probabilistically checkable proofs [5, 57, 4, 3], "hardness versus randomness trade-offs" [6], and statistical zero-knowledge [121].

This paradigm (i.e., "forcing proper behavior" via zero-knowledge proofs), which in turn is based on the fact that zero-knowledge proofs can be constructed for any NP-set, has been utilized in numerous different settings. Indeed, this paradigm is the basis for the wide applicability of zero-knowledge protocols in Cryptography.

Zero-knowledge proofs for all IP. For the sake of elegancy, we mention that under the same assumption used in the case of $\mathcal{NP}$, it holds that *any set that has an interactive proof also has a zero-knowledge interactive proof* (cf. [91, 25]).

4.4 Variants and Issues

In this section we consider numerous variants on the notion of zero-knowledge and the underlying model of interactive proofs. These include computational soundness (cf. Section 4.4.1), black-box simulation and other variants of zero-knowledge (cf. Section 4.4.2), as well as notions such as proofs of knowledge, non-interactive zero-knowledge, and witness indistinguishable proofs (cf. Section 4.4.3). We conclude this section by reviewing relatively recent results regarding the composition of zero-knowledge protocols and the power of non-black-box simulation (cf. Section 4.4.4).

4.4.1 Computational Soundness

A fundamental variant on the notion of interactive proofs was introduced by Brassard, Chaum and Crépeau [33], who relaxed the soundness condition so that it only refers to feasible ways of trying to fool the verifier (rather than to all possible ways). Specifically, the soundness condition was replaced by a computational soundness condition that asserts that it is infeasible to fool the verifier into accepting false statements. We warn that although the computational-soundness error can *always* be reduced by sequential repetitions, it is *not* true that this error can *always* be reduced by parallel repetitions (cf. [21]).

Protocols that satisfy the computational-soundness condition are called arguments.[11] We mention that argument systems may be more efficient than interactive proofs (see [93] vs. [71, 79]) as well as provide stronger zero-knowledge guarantees (see [33] vs. [61, 1]). Specifically, perfect zero-knowledge arguments for $\mathcal{NP}$ can be constructed based on some reasonable assumptions [33], where perfect zero-knowledge means that the simulator's output is distributed *identically* to the verifier's view in the real interaction (see discussion in Section 4.4.2). Note that stronger security for the prover (as provided by perfect zero-knowledge) comes at the cost of weaker security for the verifier (as provided by computational soundness). The answer to the question of whether or not this trade-off is worthwhile seems to be application dependent, and one should also take into account the availability and complexity of the corresponding protocols. (However, as stated in Footnote 7, we believe that a presentation in terms of proofs should be preferred for expositional purposes.)

4.4.2 Definitional variations

We consider several definitional issues regarding the notion of zero-knowledge (as defined in Definition 4.1).

[11] A related notion (not discussed here) is that of CS-proofs, introduced by Micali [102].

Universal and black-box simulation. Further strengthening of Definition 4.1 is obtained by requiring the existence of a universal simulator, denoted $\mathcal{C}$, that is given the program of the verifier (i.e., B^*) as an auxiliary-input; that is, in terms of Definition 4.1, one should replace $C^*(x,z)$ by $\mathcal{C}(x,z,\langle B^*\rangle)$, where $\langle B^*\rangle$ denotes the description of the program of B^* (which may depend on x and on z). That is, we effectively restrict the simulation by requiring that it be a uniform (feasible) function of the verifier's program (rather than arbitrarily depend on it). This restriction is very natural, because it seems hard to envision an alternative way of establishing the zero-knowledge property of a given protocol. Taking another step, one may argue that since it seems infeasible to reverse-engineer programs, the simulator may as well just use the verifier strategy as an oracle (or as a "black-box"). This reasoning gave rise to the notion of black-box simulation, which was introduced and advocated in [73] and further studied in numerous works (see, e.g., [41]). The belief was that inherent limitations regarding black-box simulation represent inherent limitations of zero-knowledge itself. For example, it was believed that the *fact* that the parallel version of the interactive proof of Figure 5 cannot be simulated in a black-box manner (unless $\mathcal{NP}$ is contained in $\mathcal{BPP}$ [73]) *implies* that this version is not zero-knowledge (as per Definition 4.1 itself). However, the (underlying) belief that any zero-knowledge protocol can be simulated in a black-box manner was refuted recently by Barak [7]. For further discussion, see Section 4.4.4.

Honest verifier versus general cheating verifier. Definition 4.1 refers to all feasible verifier strategies, which is most natural (in the cryptographic setting) because zero-knowledge is supposed to capture the robustness of the prover under *any feasible* (i.e., adversarial) attempt to gain something by interacting with it. A weaker and still interesting notion of zero-knowledge refers to what can be gained by an "honest verifier" (or rather a semi-honest verifier)[12] that interacts with the prover as directed, with the exception that it may maintain (and output) a record of the entire interaction (i.e., even if directed to erase all records of the interaction). Although such a weaker notion is not satisfactory for standard cryptographic applications, it yields a fascinating notion from a conceptual as well as a complexity-theoretic point of view. Furthermore, as shown in [78, 127], every proof system that is *zero-knowledge with respect to the honest-verifier* can be transformed into a *standard zero-knowledge* proof (without using intractability assumptions and in case of "public-coin" proofs this is done without significantly increasing the prover's computational effort).

Statistical versus Computational Zero-Knowledge. Recall that Definition 4.1 postulates that for every probability ensemble of one type (i.e., representing the verifier's output after interaction with the prover) there exists a "similar" ensemble of a second type (i.e., representing the simulator's output). One key parameter is the interpretation of "similarity". Three interpretations, yielding different notions of zero-knowledge, have been commonly considered in the literature (cf., [83, 61]):

1. Perfect Zero-Knowledge requires that the two probability ensembles be identical.[13]
2. Statistical Zero-Knowledge requires that these probability ensembles be statistically close (i.e., the variation distance between them is negligible).

[12] The term "honest verifier" is more appealing when considering an alternative (equivalent) formulation of Definition 4.1. In the alternative definition (see [67, Sec. 4.3.1.3]), the simulator is "only" required to generate the verifier's view of the real interaction, where the verifier's view includes its (common and auxiliary) inputs, the outcome of its coin tosses, and all messages it has received.

[13] The actual definition of Perfect Zero-Knowledge allows the simulator to fail (while outputting a special symbol) with negligible probability, and the output distribution of the simulator is conditioned on its not failing.

3. Computational (or rather general) Zero-Knowledge requires that these probability ensembles be computationally indistinguishable.

Indeed, Computational Zero-Knowledge is the most liberal notion, and is the notion considered in Definition 4.1. We note that the class of problems having statistical zero-knowledge proofs contains several problems that are considered intractable. The interested reader is referred to [126].

Strict versus expected probabilistic polynomial-time. So far, we did not specify what we exactly mean by the term probabilistic polynomial-time. Two common interpretations are:

1. Strict probabilistic polynomial-time. That is, there exist a (polynomial in the length of the input) bound on the *number of steps in each possible run* of the machine, regardless of the outcome of its coin tosses.

2. Expected probabilistic polynomial-time. The standard approach is to look at the running-time as a random variable and *bound its expectation* (by a polynomial in the length of the input). As observed by Levin (cf. [67, Sec. 4.3.1.6] and [12]), this definitional approach is quite problematic (e.g., it is not model-independent and is not closed under algorithmic composition), and an alternative treatment of this random variable is preferable.

Consequently, the notion of expected polynomial-time raises a variety of conceptual and technical problems. For that reason, whenever possible, one should prefer the more robust (and restricted) notion of strict (probabilistic) polynomial-time. Thus, with the *exception of constant-round* zero-knowledge protocols, whenever we talked of a probabilistic polynomial-time verifier (resp., simulator) we mean one in the strict sense. In contrast, with the exception of [7, 12], all results regarding *constant-round* zero-knowledge protocols refer to a strict polynomial-time verifier and an expected polynomial-time simulator, which is indeed a small cheat. For further discussion, the reader is referred to [12].

4.4.3 Related notions: POK, NIZK, and WI

We briefly discuss the notions of proofs of knowledge (POK), non-interactive zero-knowledge (NIZK), and witness indistinguishable proofs (WI).

Proofs of Knowledge. Loosely speaking, proofs of knowledge (cf. [83]) are interactive proofs in which the prover asserts "knowledge" of some object (e.g., a 3-coloring of a graph), and not merely its existence (e.g., the existence of a 3-coloring of the graph, which in turn is equivalent to the assertion that the graph is 3-colorable). Before clarifying what we mean by saying that a machine knows something, we point out that "proofs of knowledge", and in particular zero-knowledge "proofs of knowledge", have many applications to the design of cryptographic schemes and cryptographic protocols. One famous application of zero-knowledge proofs of knowledge is to the construction of identification schemes (e.g., the Fiat-Shamir scheme [60]).

What does it mean to say that a *machine* knows something? Any standard dictionary suggests several meanings for the verb `to know`, which are typically phrased with reference to *awareness*, a notion which is certainly inapplicable in the context of machines. We must look for a *behavioristic* interpretation of the verb `to know`. Indeed, it is reasonable to link knowledge with ability to do something (e.g., the ability to write down whatever one knows). Hence, we will say that a machine knows a string α if it *can* output the string α. But this seems as total non-sense too: a machine has a well defined output – either the output equals α or it does not. So what can be meant by saying that *a machine can do something?* Loosely speaking, it may mean that the machine can be *easily modified* so that it does whatever is claimed. More precisely, it may mean that there exists an *efficient* machine that, using the original machine as a black-box (or given its code as an input), outputs whatever is claimed.

So much for defining the "knowledge of machines". Yet, whatever a machine knows or does not know is "its own business". What can be of interest and reference *to the outside* is whatever can be deduced about the knowledge of a machine by interacting with it. Hence, we are interested in proofs of knowledge (rather than in mere knowledge). For sake of simplicity let us consider a concrete question: *how can a machine prove that it knows a 3-coloring of a graph?* An obvious way is just to send the 3-coloring to the verifier. Yet, we claim that applying the protocol in Figure 5 (i.e., the zero-knowledge proof system for 3-Colorability) is an alternative way of proving knowledge of a 3-coloring of the graph.

The definition of a *verifier of knowledge of 3-coloring* refers to any possible prover strategy. It requires the existence of an efficient universal way of "extracting" a 3-coloring of a given graph by using any prover strategy that convinces the verify to accept the graph (with noticeable probability). Surely, we should no expect much of prover strategies that convince the verifier to accept the graph with negligible probability. However, a robust definition should allow a smooth passage from noticeable to negligible, and should allow to establish the intuitive zero-knowledge property of a party that sends some information that the other party has proved it knows.

Loosely speaking, we may say that an interactive machine, V, constitutes a verifier for knowledge of 3-coloring if, for any prover strategy P, the complexity of extracting a 3-coloring of G when using machine P as a "black box"[14] is inversely proportional to the probability that the verifier is convinced by P (to accept the graph G). Namely, the extraction of the 3-coloring is done by an oracle machine, called an *extractor*, that is given access to a function specifying the behavior P (i.e., the messages it sends in response to particular messages it may receive). We require that the (expected) running time of the extractor, on input G and access to an oracle specifying P's messages, be inversely related (by a factor polynomial in $|G|$) to the probability that P convinces V to accept G. In case P always convinces V to accept G, the extractor runs in expected polynomial-time. The same holds in case P convinces V to accept with noticeable probability. On the other hand, in case P never convinces V to accept, essentially nothing is required of the extractor. (We stress that the latter special cases do not suffice for a satisfactory definition; see discussion in [67, Sec. 4.7.1].)

Non-Interactive Zero-Knowledge. The model of non-interactive zero-knowledge proof systems, introduced in [30], consists of three entities: a prover, a verifier and a uniformly selected reference string (which can be thought of as being selected by a trusted third party). Both the verifier and prover can read the reference string, and each can toss additional coins. The interaction consists of a single message sent from the prover to the verifier, who then is left with the final decision (whether to accept or not). The (basic) zero-knowledge requirement refers to a simulator that outputs pairs that should be computationally indistinguishable from the distribution (of pairs consisting of a uniformly selected reference string and a random prover message) seen in the real model.[15] Non-interactive zero-knowledge proof systems have numerous applications (e.g., to the construction of public-key encryption and signature schemes, where the reference string may

[14] Indeed, one may consider also non-black-box extractors as done in [12].

[15] Note that the verifier does not effect the distribution seen in the real model, and so the basic definition of zero-knowledge does not refer to it. The verifier (or rather a process of adaptively selecting assertions to be proved) will be referred to in the adaptive variants of the definition.

be incorporated in the public-key). Several different definitions of non-interactive zero-knowledge proofs were considered in the literature.

- In the *basic definition*, one considers proving a single assertion of a-priori bounded length, where this length may be smaller than the length of the reference string.
- A natural extension, required in many applications, is the ability to prove multiple assertions of varying length, where the total length of these assertions may exceed the length of the reference string (as long as the total length is polynomial in the length of the reference string). This definition is sometimes referred to as the *unbounded definition*, because the total length of the assertions to be proved is not a-priori bounded.
- Other natural extensions refer to the preservation of security (i.e., both soundness and zero-knowledge) when the assertions to be proved are selected *adaptively* (based on the reference string and possibly even based on previous proofs).
- Finally, we mention the notion of *simulation-soundness*, which is related to *non-malleability*. This extension, which mixes the zero-knowledge and soundness conditions, refers to the soundness of proofs presented by an adversary after it obtains proofs of assertions of its own choice (with respect to the same reference string). This notion is important in applications of non-interactive zero-knowledge proofs to the construction of public-key encryption schemes secure against chosen ciphertext attacks (see [68, Sec. 5.4.4.4]).

Constructing non-interactive zero-knowledge proofs seems more difficult than constructing interactive zero-knowledge proofs. Still, based on standard intractability assumptions (e.g., intractability of factoring), it is known how to construct a non-interactive zero-knowledge proof (even in the adaptive and non-malleable sense) for any NP-set (cf. [58, 50]).

Witness Indistinguishability and the FLS-Technique. The notion of witness indistinguishability was suggested in [59] as a meaningful relaxation of zero-knowledge. Loosely speaking, for any NP-relation R, a proof (or argument) system for the corresponding NP-set is called witness indistinguishable if no feasible verifier may distinguish the case in which the prover uses one NP-witness to x (i.e., w_1 such that $(x, w_1) \in R$) from the case in which the prover is using a different NP-witness to the same input x (i.e., w_2 such that $(x, w_2) \in R$). Clearly, any zero-knowledge protocol is witness indistinguishable, but the converse does not necessarily hold. Furthermore, it seems that witness indistinguishable protocols are easier to construct than zero-knowledge ones. Another advantage of witness indistinguishable protocols is that they are closed under arbitrary concurrent composition [59], whereas in general zero-knowledge protocols are not closed even under parallel composition [73]. Witness indistinguishable protocols turned out to be an *important tool in the construction of more complex protocols*, as is demonstrated next.

Feige, Lapidot and Shamir [58] introduced a technique for constructing zero-knowledge proofs (and arguments) based on witness indistinguishable proofs (resp., arguments). Following is a sketchy description of a special case of their technique, often referred to as the FLS-technique, which has been used in numerous works. On common input $x \in L$, where L is the NP-set defined by the witness relation R, the following two steps are performed:

1. The parties generate an instance x' for an auxiliary NP-set L', where L' is defined by a witness relation R'. The generation protocol in use must satisfy the following two conditions:

(a) If the verifier follows its prescribed strategy then no matter which strategy is used by the prover, with high probability, the protocol's outcome is a NO-instance of L'.

(b) Loosely speaking, there exists an efficient (non-interactive) procedure for producing a (random) transcript of the generation protocol such that the corresponding protocol's outcome is a YES-instance of L' and yet the produced transcript is computationally indistinguishable from the transcript of a real execution of the protocol. Furthermore, this procedure also outputs an NP-witness for the YES-instance that appears as the protocol's outcome.

For example, L' may consist of all possible outcomes of a pseudorandom generator that stretches its seed by a factor of two, and the generation protocol may consist of the two parties iteratively invoking a "coin tossing" protocol to obtain a random string. Note that the outcome of a real execution will be an almost uniformly distributed string, which is most likely a NO-instance of L', whereas it is easy to generate a (random) transcript corresponding to any desired outcome (provided that the parties use an adequate coin tossing protocol).

2. The parties execute a *witness indistinguishable* proof for the NP-set L'' defined by the witness relation $R'' = \{((\alpha,\alpha'),(\beta,\beta')) : (\alpha,\beta) \in R \vee (\alpha',\beta') \in R'\}$. The sub-protocol is such that the corresponding prover can be implemented in probabilistic polynomial-time given any NP-witness for $(\alpha,\alpha') \in L''$. The sub-protocol is invoked on common input (x,x'), where x' is the outcome of Step 1, and the sub-prover is invoked with the corresponding NP-witness as auxiliary input (i.e., with $(w,0)$, where w is the NP-witness for x (given to the main prover)).

The soundness of the above protocol follows by Property (a) of the generation protocol (i.e., with high probability $x' \notin L'$, and so $x \in L$ follows by the soundness of the protocol used in Step 2). To demonstrate the zero-knowledge property, we first generate a simulated transcript of Step 1 (with outcome $x' \in L'$) along with an adequate NP-witness (i.e., w' such that $(x',w') \in R'$), and then emulate Step 2 by feeding the sub-prover strategy with the NP-witness $(0,w')$. Combining Property (b) of the generation protocol and the witness indistinguishability property of the protocol used in Step 2, the simulation is indistinguishable from the real execution.

4.4.4 Two basic problems: composition and black-box simulation

We conclude this section by considering two basic problems regarding zero-knowledge, which actually arise also with respect to the security of other cryptographic primitives.

Composition of protocols. The first question refers to the *preservation of security* (i.e., zero-knowledge in our case) *under various types of composition operations.* These composition operations represent independent executions of a protocol that are attacked by an adversary (which coordinates its actions in the various executions). The preservation of security under such compositions (which involve only executions of the same protocol) is a first step towards the study of the security of the protocol when executed together with other protocols (see further discussion in Section 7.4). Turning back to zero-knowledge, we recall the main facts regarding sequential, parallel and concurrent execution of (arbitrary and/or specific) zero-knowledge protocols:

Sequential composition: As stated above, zero-knowledge (with respect to auxiliary inputs) is closed under sequential composition.

Parallel composition: In general, zero-knowledge is not closed under parallel composition [73]. Yet, some zero-knowledge proofs (for NP) preserve their security when many copies are executed in parallel. Furthermore, some of these protocol use a constant number of rounds (cf. [69]).

Concurrent composition: One may view parallel composition as concurrent composition in a model of strict synchronity. This leads us to consider more general models of concurrent composition. We distinguish between a model of full asynchronicity and a model naturally limited asynchronicity.

- In the full asynchronous model, some zero-knowledge proofs (for NP) preserve their security when many copies are executed concurrently (cf. [116, 94, 111]), but such a result is not known for constant-round protocols.
- In contrast, constant-round zero-knowledge proofs (for NP) are known (cf. [55, 69]) in a model of limited asynchronicity, where each party holds a local clock such that the relative clock rates are bounded by an a-priori known constant and the protocols may employ time-driven operations (i.e., `time-out` in-coming messages and `delay` out-going messages).

The study of zero-knowledge in the concurrent setting provides a good test case for the study of concurrent security of general protocols. In particular, the results in [73, 41] point out inherent limitations of the "standard proof methods" (used to establish zero-knowledge) when applied to the concurrent setting, where [73] treats the synchronous case and [41] uncovers much stronger limitations for the asynchronous case. By "standard proof methods" we refer to the establishment of zero-knowledge via a single simulator that obtains only oracle (or "black-box") access to the adversary procedure.

Black-box proofs of security. The second basic question regarding zero-knowledge refers to the usage of the adversary's program within the proof of security (i.e., demonstration of the zero-knowledge property). For 15 years, all known proofs of security used the adversary's program as a black-box (i.e., a universal simulator was presented using the adversary's program as an oracle). Furthermore, it was believed that there was no advantage in having access to the code of the adversary's program (cf. [73]). Consequently it was conjectured that negative results regarding black-box simulation represent an inherent limitation of zero-knowledge. This belief has been refuted recently by Barak [7] who constructed a zero-knowledge argument (for NP) that has important properties that are impossible to achieve by black-box simulation (unless $\mathcal{NP} \subseteq \mathcal{BPP}$). For example, this zero-knowledge argument uses a constant number of rounds and preserves its security when an a-priori fixed (polynomial) number of copies are executed concurrently.[16]

Barak's results (cf. [7] and also [8]) call for the re-evaluation of many common beliefs. Most concretely, they say that results regarding black-box simulators do not reflect inherent limitations of zero-knowledge (but rather an inherent limitation of a natural way of demonstrating the zero-knowledge property). Most abstractly, they say that there are meaningful ways of using a program other than merely invoking it as a black-box. Does this mean that a method was found to "reverse engineer" programs or to "understand" them? We believe that the answer is negative. Barak [7]

[16]This result falls short of achieving a fully concurrent zero-knowledge argument, because the number of concurrent copies must be fixed before the protocol is presented. Specifically, the protocol uses messages that are longer than the allowed number of concurrent copies. However, even preservation of security under an a priori bounded number of executions goes beyond the impossibility results of [73, 41] (which refers to black-box simulations).

is using the adversary's program in a significant way (i.e., more significant than just invoking it), without "understanding" it.

The key idea underlying Barak's protocol [7] is to have the prover prove that either the original NP-assertion is valid or that he (i.e., the prover) "knows the verifier's residual strategy" (in the sense that it can predict the next verifier message). Indeed, in a real interaction (with the honest verifier), it is infeasible for the prover to predict the next verifier message, and so computational-soundness of the protocol follows. However, a simulator that is given the code of the verifier's strategy (and not merely oracle access to that code), can produce a valid proof of the disjunction by properly executing the sub-protocol using its knowledge of an NP-witness for the second disjunctive. The simulation is computationally indistinguishable from the real execution, provided that one cannot distinguish an execution of the sub-protocol in which one NP-witness (i.e., an NP-witness for the original assertion) is used from an execution in which the second NP-witness (i.e., an NP-witness for the auxiliary assertion) is used. That is, the sub-protocol should be a *witness indistinguishable* argument system, and the entire construction uses the *FLS technique* (described in Section 4.4.3). We warn the reader that the actual implementation of the above idea requires overcoming several technical difficulties (cf. [7, 10]).

Part II

Basic Applications

Encryption and signature schemes are the most basic applications of Cryptography. Their main utility is in providing secret and reliable communication over insecure communication media. Loosely speaking, encryption schemes are used to ensure the secrecy (or privacy) of the actual information being communicated, whereas signature schemes are used to ensure its reliability (or authenticity). In this part we survey these basic applications as well as the construction of general secure cryptographic protocols. For more details regarding the contents of the current part, see our recent textbook [68].

5 Encryption Schemes

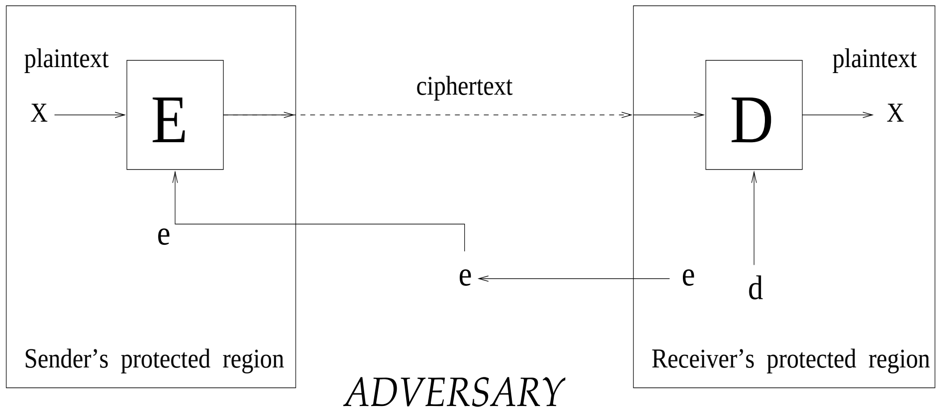
The problem of providing *secret communication over insecure media* is the traditional and most basic problem of cryptography. The setting of this problem consists of two parties communicating through a channel that is possibly tapped by an adversary. The parties wish to exchange information with each other, but keep the "wire-tapper" as ignorant as possible regarding the contents of this information. The canonical solution to the above problem is obtained by the use of encryption schemes. Loosely speaking, an encryption scheme is a protocol allowing these parties to communicate *secretly* with each other. Typically, the encryption scheme consists of a pair of algorithms. One algorithm, called encryption, is applied by the sender (i.e., the party sending a message), while the other algorithm, called decryption, is applied by the receiver. Hence, in order to send a message, the sender first applies the encryption algorithm to the message, and sends the result, called the ciphertext, over the channel. Upon receiving a ciphertext, the other party (i.e., the receiver) applies the decryption algorithm to it, and retrieves the original message (called the plaintext).

In order for the above scheme to provide secret communication, the communicating parties (at least the receiver) must know something that is not known to the wire-tapper. (Otherwise, the wire-tapper can decrypt the ciphertext exactly as done by the receiver.) This extra knowledge may take the form of the decryption algorithm itself, or some parameters and/or auxiliary inputs used by the decryption algorithm. We call this extra knowledge the *decryption-key*. Note that, without loss of generality, we may assume that the decryption algorithm is known to the wire-tapper, and that the decryption algorithm operates on two inputs: a ciphertext and a decryption-key. We stress that the existence of a decryption-key, not known to the wire-tapper, is merely a necessary condition for secret communication. The above description implicitly presupposes the existence of an efficient algorithm for generating (random) keys.

Evaluating the "security" of an encryption scheme is a very tricky business. A preliminary task is to understand what is "security" (i.e., to properly define what is meant by this intuitive term). Two approaches to defining security are known. The first ("classical") approach, introduced by Shannon [124], is *information theoretic*. It is concerned with the "information" about the plaintext that is "present" in the ciphertext. Loosely speaking, if the ciphertext contains information about the plaintext then the encryption scheme is considered insecure. It has been shown that such high (i.e., "perfect") level of security can be achieved only if the key in use is at least as long as the *total* amount of information sent via the encryption scheme [124]. This fact (i.e., that the key has to be longer than the information exchanged using it) is indeed a drastic limitation on the applicability of such (perfectly-secure) encryption schemes.

The second ("modern") approach, followed in the current text, is based on *computational com-*

plexity. This approach is based on the thesis that it *does not matter* whether the ciphertext contains information about the plaintext, but rather whether this information can be *efficiently extracted.* In other words, instead of asking whether it is *possible* for the wire-tapper to extract specific information, we ask whether it is *feasible* for the wire-tapper to extract this information. It turns out that the new (i.e., "computational complexity") approach can offer security even when the key is much shorter than the total length of the messages sent via the encryption scheme.

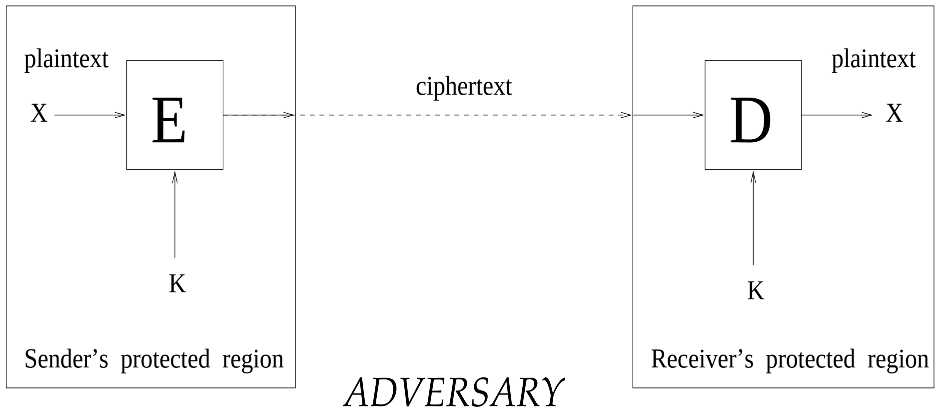


The key-pair (e, d) is generated by the receiver, who posts the encryption-key e on a public media, while keeping the decryption-key d secret.

Figure 6: Public-key encryption schemes – an illustration.

The computational complexity approach enables the introduction of concepts and primitives that cannot exist under the information theoretic approach. A typical example is the concept of *public-key encryption schemes*, introduced by Diffie and Hellman [51]. Recall that in the above discussion we concentrated on the decryption algorithm and its key. It can be shown that the encryption algorithm must get, in addition to the message, an auxiliary input that depends on the decryption-key. This auxiliary input is called the *encryption-key.* Traditional encryption schemes, and in particular all the encryption schemes used in the millennia until the 1980's, operate with an encryption-key that equals the decryption-key. Hence, the wire-tapper in these schemes must be ignorant of the encryption-key, and consequently the *key distribution* problem arises; that is, how can two parties wishing to communicate over an insecure channel agree on a secret encryption/decryption key. (The traditional solution is to exchange the key through an alternative channel that is secure though (much) more expensive to use.) The computational complexity approach allows the introduction of encryption schemes in which the encryption-key may be given to the wire-tapper without compromising the security of the scheme. Clearly, the decryption-key in such schemes is different from the encryption-key, and furthermore infeasible to compute from the encryption-key. Such encryption schemes, called public-key schemes, have the advantage of trivially resolving the key distribution problem (because the encryption-key can be publicized). That is, once some Party X generates a pair of keys and publicizes the encryption-key, any party can send encrypted messages to Party X so that Party X can retrieve the actual information (i.e., the plaintext), whereas nobody else can learn anything about the plaintext.

In contrast to public-key schemes, traditional encryption schemes in which the encryption-key equals the description-key are called private-key schemes, because in these schemes the encryption-



The key K is known to both receiver and sender, but is unknown to the adversary. For example, the receiver may generate K at random and pass it to the sender via a perfectly-private secondary channel (not shown here).

Figure 7: Private-key encryption schemes – an illustration.

key must be kept secret (rather than be public as in public-key encryption schemes). We note that a full specification of either schemes requires the specification of the way in which keys are generated; that is, a (randomized) key-generation algorithm that, given a security parameter, produces a (random) pair of corresponding encryption/decryption keys (which are identical in case of private-key schemes).

Thus, both private-key and public-key encryption schemes consist of three efficient algorithms: a key generation algorithm denoted G, an encryption algorithm denoted E, and a decryption algorithm denoted D. For every pair of encryption and decryption keys (e, d) generated by G, and for every plaintext x, it holds that $D_d(E_e(x)) = x$, where $E_e(x) \stackrel{\text{def}}{=} E(e, x)$ and $D_d(y) \stackrel{\text{def}}{=} D(d, y)$. The difference between the two types of encryption schemes is reflected in the definition of security: the security of a public-key encryption scheme should hold also when the adversary is given the encryption-key, whereas this is not required for a private-key encryption scheme. Below we focus on the public-key case (and the private-key case can be obtained by omitting the encryption-key from the sequence of inputs given to the adversary).

5.1 Definitions

A good disguise should not reveal the person's height.

Shafi Goldwasser and Silvio Micali, 1982

For simplicity, we first consider the encryption of a single message (which, for further simplicity, is assumed to be of length n).[17] As implied by the above discussion, a public-key encryption scheme is said to be secure if it is infeasible to gain any information about the plaintext by looking at the ciphertext (and the encryption-key). That is, whatever information about the plaintext one may compute from the ciphertext and some a-priori information, can be essentially computed

[17] In the case of public-key schemes no generality is lost by these simplifying assumptions, but in the case of private-key schemes one should consider the encryption of polynomially-many messages (as we do below).

as efficiently from the a-priori information alone. This fundamental definition of security (called semantic security) turns out to be equivalent to saying that, for any two messages, it is infeasible to distinguish the encryption of the first message from the encryption of the second message, even when given the encryption-key. Both definitions were introduced by Goldwasser and Micali [82].

Definition 5.1 (semantic security (following [82], revisited [64])): *A* public-key *encryption scheme* (G, E, D) *is* semantically secure *if for every probabilistic polynomial-time algorithm, A, there exists a probabilistic polynomial-time algorithm B so that for every two functions* $f, h : \{0,1\}^* \to \{0,1\}^*$ *such that* $|h(x)| = \mathrm{poly}(|x|)$*, and all probability ensembles* $\{X_n\}_{n \in \mathbb{N}}$*, where* X_n *is a random variable ranging over* $\{0,1\}^n$*, it holds that*

$$\Pr[A(e, E_e(x), h(x)) = f(x)] \;<\; \Pr[B(1^n, h(x)) = f(x)] + \mu(n)$$

where the plaintext x is distributed according to X_n*, the encryption-key e is distributed according to* $G(1^n)$*, and* μ *is a negligible function.*

That is, it is feasible to predict $f(x)$ from $h(x)$ as successfully as it is to predict $f(x)$ from $h(x)$ and $(e, E_e(x))$, which means that nothing is gained by obtaining $(e, E_e(x))$. Note that no computational restrictions are made regarding the functions h and f. We stress that the above definition (as well as the next one) refers to public-key encryption schemes, and in the case of private-key schemes algorithm A is not given the encryption-key e.

A good disguise should not allow a mother to distinguish her own children.

Shafi Goldwasser and Silvio Micali, 1982

The following technical interpretation of security states that it is infeasible to distinguish the encryptions of two plaintexts (of the same length).

Definition 5.2 (indistinguishability of encryptions (following [82])): *A* public-key *encryption scheme* (G, E, D) *has* indistinguishable encryptions *if for every probabilistic polynomial-time algorithm, A, and all sequences of triples,* $(x_n, y_n, z_n)_{n \in \mathbb{N}}$*, where* $|x_n| = |y_n| = n$ *and* $|z_n| = \mathrm{poly}(n)$,

$$|\Pr[A(e, E_e(x_n), z_n) = 1] - \Pr[A(e, E_e(y_n), z_n) = 1]| = \mu(n)$$

Again, e is distributed according to $G(1^n)$*, and* μ *is a negligible function.*

In particular, z_n may equal (x_n, y_n). Thus, it is infeasible to distinguish the encryptions of any two fixed messages (such as the all-zero message and the all-ones message).

Definition 5.1 is more appealing in most settings where encryption is considered the end goal. Definition 5.2 is used to establish the security of candidate encryption schemes as well as to analyze their application as modules inside larger cryptographic protocols. Thus, their equivalence is of major importance.

Equivalence of Definitions 5.1 and 5.2 – proof ideas. Intuitively, indistinguishability of encryptions (i.e., of the encryptions of x_n and y_n) is a special case of semantic security; specifically, it corresponds to the case that X_n is uniform over $\{x_n, y_n\}$, f indicates one of the plaintexts and h does not distinguish them (i.e., $f(w) = 1$ iff $w = x_n$ and $h(x_n) = h(y_n) = z_n$, where z_n is as in Definition 5.2). The other direction is proved by considering the algorithm B that, on input $(1^n, v)$ where $v = h(x)$, generates $(e, d) \leftarrow G(1^n)$ and outputs $A(e, E_e(1^n), v)$, where A is as in Definition 5.1. Indistinguishability of encryptions is used to prove that B performs as well as A (i.e., for every h, f and $\{X_n\}_{n \in \mathbb{N}}$, it holds that $\Pr[B(1^n, h(X_n)) = f(X_n)] = \Pr[A(e, E_e(1^n), h(X_n)) = f(X_n)]$ approximately equals $\Pr[A(e, E_e(X_n), h(X_n)) = f(X_n)]$).

Probabilistic Encryption: It is easy to see that a secure *public-key* encryption scheme must employ a probabilistic (i.e., randomized) encryption algorithm. Otherwise, given the encryption-key as (additional) input, it is easy to distinguish the encryption of the all-zero message from the encryption of the all-ones message.[18] This explains the association of the aforementioned robust security definitions and *probabilistic encryption*, an association that goes back to the title of the pioneering work of Goldwasser and Micali [82].

Further discussion: We stress that (the equivalent) Definitions 5.1 and 5.2 go way beyond saying that it is infeasible to recover the plaintext from the ciphertext. The latter statement is indeed a minimal requirement from a secure encryption scheme, but is far from being a sufficient requirement. Typically, encryption schemes are used in applications where even obtaining partial information on the plaintext may endanger the security of the application. When designing an application-independent encryption scheme, we do not know which partial information endangers the application and which does not. Furthermore, even if one wants to design an encryption scheme tailored to a specific application, it is rare (to say the least) that one has a precise characterization of all possible partial information that endanger this application. Thus, we need to require that it is infeasible to obtain any information about the plaintext from the ciphertext. Furthermore, in most applications the plaintext may not be uniformly distributed and some a-priori information regarding it is available to the adversary. We require that the secrecy of all partial information is preserved also in such a case. That is, even in presence of a-priori information on the plaintext, it is infeasible to obtain any (new) information about the plaintext from the ciphertext (beyond what is feasible to obtain from the a-priori information on the plaintext). The definition of semantic security postulates all of this. The equivalent definition of indistinguishability of encryptions is useful in demonstrating the security of candidate constructions as well as for arguing about their effect as part of larger protocols.

Security of multiple messages: Definitions 5.1 and 5.2 refer to the security of an encryption scheme that is used to encrypt a single plaintext (per generated key). Since the plaintext may be longer than the key[19], these definitions are already non-trivial, and an encryption scheme satisfying them (even in the private-key model) implies the existence of one-way functions. Still, in many cases, it is desirable to encrypt many plaintexts using the same encryption-key. Loosely speaking, an encryption scheme is secure in the multiple-messages setting if analogous definitions (to Definitions 5.1 and 5.2) hold when polynomially-many plaintexts are encrypted using the same encryption-key (cf. [68, Sec. 5.2.4]). It is easy to see that *in the public-key model*, security in the single-message setting implies security in the multiple-messages setting. We stress that this is not necessarily true *for the private-key model*.

5.2 Constructions

It is common practice to use "pseudorandom generators" as a basis for private-key encryption schemes. We stress that this is a very dangerous practice when the "pseudorandom generator" is

[18] The same holds for (stateless) *private-key* encryption schemes, when considering the security of encrypting several messages (rather than a single message as done above). For example, if one uses a deterministic encryption algorithm then the adversary can distinguish two encryptions of the same message from the encryptions of a pair of different messages.

[19] Recall that for sake of simplicity we have considered only messages of length n, but the general definitions refer to messages of arbitrary (polynomial in n) length. We comment that, in the general form of Definition 5.1, one should provide the length of the message as an auxiliary input to both algorithms (A and B).

easy to predict (such as the linear congruential generator or some modifications of it that output a constant fraction of the bits of each resulting number). However, this common practice becomes sound provided one uses pseudorandom generators (as defined in Section 3.2). An alternative and more flexible construction follows.

Private-Key Encryption Scheme based on Pseudorandom Functions: We present a simple construction that uses pseudorandom functions as defined in Section 3.3. The key generation algorithm consists of selecting a seed, denoted s, for a (pseudorandom) function, denoted f_s. To encrypt a message $x \in \{0,1\}^n$ (using key s), the encryption algorithm uniformly selects a string $r \in \{0,1\}^n$ and produces the ciphertext $(r, x \oplus f_s(r))$, where $\oplus$ denotes the exclusive-or of bit strings. To decrypt the ciphertext (r, y) (using key s), the decryption algorithm just computes $y \oplus f_s(r)$. The proof of security of this encryption scheme consists of two steps (suggested as a general methodology in Section 3.3):

1. Prove that an idealized version of the scheme, in which one uses a uniformly selected function $F:\{0,1\}^n \to \{0,1\}^n$, rather than the pseudorandom function f_s, is secure.

2. Conclude that the real scheme (as presented above) is secure (because, otherwise one could distinguish a pseudorandom function from a truly random one).

Note that we could have gotten rid of the randomization (in the encryption process) if we had allowed the encryption algorithm to be history dependent (e.g., use a counter in the role of r). This can be done provided that either only one party uses the key for encryption (and maintains a counter) or that all parties that encrypt, using the same key, coordinate their actions (i.e., maintain a joint state (e.g., counter)). Indeed, when using a private-key encryption scheme, a common situation is that the same key is only used for communication between two specific parties, which update a joint counter during their communication. Furthermore, if the encryption scheme is used for FIFO communication between the parties and both parties can reliably maintain the counter value, then there is no need (for the sender) to send the counter value. (The resulting scheme is related to "stream ciphers" that are commonly used in practice.)

We comment that the use of a counter (or any other state) in the encryption process is not reasonable in the case of public-key encryption schemes, because it is incompatible with the canonical usage of such schemes (i.e., allowing all parties to send encrypted messages to the "owner of the encryption-key" without engaging in any type of further coordination or communication). Furthermore, as discussed before, probabilistic encryption is essential for a secure public-key encryption scheme *even in the case of encrypting a single message* (unlike in the case of private-key schemes). Following Goldwasser and Micali [82], we now demonstrate the use of *probabilistic encryption* in the construction of a public-key encryption scheme.

Public-Key Encryption Scheme based on Trapdoor Permutations: We present two constructions that employ a collection of trapdoor permutations, as defined in Definition 2.3. Let $\{f_i : D_i \to D_i\}_i$ be such a collection, and let b be a corresponding hard-core predicate. The key generation algorithm consists of selecting a permutation f_i along with a corresponding trapdoor t, and outputting (i, t) as the key-pair. To encrypt a (*single*) bit σ (using the encryption-key i), the encryption algorithm uniformly selects $r \in D_i$, and produces the ciphertext $(f_i(r), \sigma \oplus b(r))$. To decrypt the ciphertext (y, τ) (using the decryption-key t), the decryption algorithm computes $\tau \oplus b(f_i^{-1}(y))$ (using the trapdoor t of f_i). Clearly, $(\sigma \oplus b(r)) \oplus b(f_i^{-1}(f_i(r))) = \sigma$. Indistinguishability of encryptions can be easily proved using the fact that b is a hard-core of f_i. We comment that the above scheme is quite wasteful in bandwidth; however, the paradigm underlying its construction (i.e., applying the trapdoor permutation to a randomized version of the plaintext rather than to the actual plaintext) is valuable in practice. A more efficient construction of a public-key encryption scheme, which uses the same key-generation algorithm, follows. To encrypt an ℓ-bit long string x (using the encryption-key i), the encryption algorithm uniformly selects $r \in D_i$, computes $y \leftarrow b(r) \cdot b(f_i(r)) \cdots b(f_i^{\ell-1}(r))$ and produces the ciphertext $(f_i^{\ell}(r), x \oplus y)$. To decrypt the ciphertext (u, v) (using the decryption-key t), the decryption algorithm first recovers $r = f_i^{-\ell}(u)$ (using the trapdoor t of f_i), and then obtains $v \oplus b(r) \cdot b(f_i(r)) \cdots b(f_i^{\ell-1}(r))$. Note the similarity to the construction in Theorem 3.3, and the fact that the proof can be extended to establish the computational indistinguishability of $(b(r) \cdots b(f_i^{\ell-1}(r)), f_i^{\ell}(r))$ and $(r', f_i^{\ell}(r))$, for random and independent $r \in D_i$ and $r' \in \{0,1\}^{\ell}$. Indistinguishability of encryptions follows, and thus the aforementioned scheme is secure.

Key-generation on security parameter n:

1. Select at random two n-bit primes, P and Q, each congruent to 3 mod 4.
2. Compute $d_P = ((P+1)/4)^{\ell(n)} \bmod P-1$, $d_Q = ((Q+1)/4)^{\ell(n)} \bmod Q-1$, $c_P = Q \cdot (Q^{-1} \bmod P)$, and $c_Q = P \cdot (P^{-1} \bmod Q)$.

The output key-pair is (N, T), where $N = PQ$ and $T = (P, Q, N, c_P, d_P, c_Q, d_Q)$.

(Note: for every s, it holds that $(s^2)^{(P+1)/4} \equiv s \pmod P$, and so $(s^{2^{\ell(n)}})^{d_P} \equiv s \pmod P$. Thus, raising to the d_P-th power modulo P is equivalent to taking the 2^{ℓ}-th root modulo P. To recover roots modulo N, we use the Chinese Remainder Theorem with the corresponding coefficients c_P and c_Q.)

Encryption of message $x \in \{0,1\}^{\ell(n)}$ using the encryption-key N:

1. Uniformly select $s_0 \in \{1, ..., N\}$.
2. For $i = 1, .., \ell(n)+1$, compute $s_i \leftarrow s_{i-1}^2 \bmod N$ and $\sigma_i = \mathrm{lsb}(s_i)$.

The ciphertext is $(s_{\ell(n)+1}, y)$, where $y = x \oplus \sigma_1 \sigma_2 \cdots \sigma_{\ell(n)}$.

(Note: s_1 plays the role played by r in the general scheme.)

Decryption of the ciphertext (r, y) using the encryption-key $T = (P, Q, N, c_P, d_P, c_Q, d_Q)$:

1. Let $s' \leftarrow r^{d_P} \bmod P$, and $s'' \leftarrow r^{d_Q} \bmod Q$.
2. Let $s_1 \leftarrow c_P \cdot s' + c_Q \cdot s'' \bmod N$.
3. For $i = 1, .., \ell(n)$, compute $\sigma_i = \mathrm{lsb}(s_i)$ and $s_{i+1} \leftarrow s_i^2 \bmod N$.

The plaintext is $y \oplus \sigma_1 \sigma_2 \cdots \sigma_{\ell(n)}$.

Note: lsb is a hard-core of the modular squaring function [2].

Figure 8: The Blum–Goldwasser Public-Key Encryption Scheme [31]. For simplicity we assume that ℓ, which is polynomially bounded (e.g., $\ell(n) = n$), is known at key-generation time.

Concrete implementations of the aforementioned public-key encryption schemes: For the first scheme, we are going to use the RSA scheme [117] as a trapdoor permutation (rather than using it directly as an encryption scheme).[20] The RSA scheme has an instance-generating

[20] Recall that RSA itself is not semantically secure, because it employs a deterministic encryption algorithm. The scheme presented here can be viewed as a "randomized version" of RSA.

algorithm that randomly selects two primes, p and q, computes their product $N = p \cdot q$, and selects at random a pair of integers (e, d) such that $e \cdot d \equiv 1 \pmod{\phi(N)}$, where $\phi(N) \stackrel{\text{def}}{=} (p-1) \cdot (q-1)$. (The "plain RSA" operations are raising to power e or d modulo N.) We construct a public-key encryption scheme as follows: The key-generation algorithm is identical to the instance-generator algorithm of RSA, and the encryption-key is set to (N, e) (resp., the decryption-key is set to (N, d)), just as in "plain RSA". To encrypt *a single bit* σ (using the encryption-key (N, e)), the encryption algorithm uniformly selects an element, r, in the set of residues mod N, and produces the ciphertext $(r^e \bmod N, \sigma \oplus \mathrm{lsb}(r))$, where $\mathrm{lsb}(r)$ denotes the least significant bit of r (which is a hard-core of the RSA function [2]). To decrypt the ciphertext (y, τ) (using the decryption-key (N, d)), the decryption algorithm just computes $\tau \oplus \mathrm{lsb}(y^d \bmod N)$. Turning to the second scheme, we assume the intractability of factoring large integers, and use squaring modulo a composite as a trapdoor permutation over the corresponding quadratic residues (while using composites that are the product of two primes, each congruent to 3 modulo 4). The resulting secure public-key encryption scheme, depicted in Figure 8, has efficiency comparable to that of (plain) RSA. We comment that special properties of modular squaring were only used (in Figure 8) to speed-up the computation of $f_i^{-\ell}$ (i.e., rather than iteratively extracting modular square roots ℓ times, we extracted the modular 2^ℓ-th root).

5.3 Beyond Eavesdropping Security

Our treatment so far has referred only to a "passive" attack in which the adversary merely eavesdrops on the line over which ciphertexts are being sent. Stronger types of attacks, culminating in the so-called Chosen Ciphertext Attack, may be possible in various applications. Specifically, in some settings it is feasible for the adversary to make the sender encrypt a message of the adversary's choice, and in some settings the adversary may even make the receiver decrypt a ciphertext of the adversary's choice. This gives rise to *chosen plaintext attacks* and to *chosen ciphertext attacks*, respectively, which are not covered by the security definitions considered in previous sections. In this section we briefly discuss such "active" attacks, focusing on chosen ciphertext attacks (of the stronger type known as "a posteriori" or "CCA2").

Loosely speaking, in a chosen ciphertext attack, the adversary may obtain the decryptions of ciphertexts of its choice, and is deemed successful if it learns something regarding the plaintext that corresponds to some different ciphertext (see [92, 19] and [68, Sec. 5.4.4]). That is, the adversary is given oracle access to the decryption function corresponding to the decryption-key in use (and, in the case of private-key schemes, it is also given oracle access to the corresponding encryption function). The adversary is allowed to query the decryption oracle on any ciphertext except for the "test ciphertext" (i.e., the very ciphertext for which it tries to learn something about the corresponding plaintext). It may also make queries that do not correspond to legitimate ciphertexts, and the answer will be accordingly (i.e., a special 'failure' symbol). Furthermore, the adversary may effect the selection of the test ciphertext (by specifying a distribution from which the corresponding plaintext is to be drawn).

Private-key and public-key encryption schemes secure against chosen ciphertext attacks can be constructed under (almost) the same assumptions that suffice for the construction of the corresponding passive schemes. Specifically:

Theorem 5.3 (folklore, see [68, Sec. 5.4.4.3]): *Assuming the existence of* one-way functions*, there exist* private-key *encryption schemes that are secure against chosen ciphertext attack.*

Theorem 5.4 ([107, 52], using [30, 58], see [68, Sec. 5.4.4.4]): *Assuming the existence of* enhanced[21] trapdoor permutations*, there exist* public-key *encryption schemes that are secure against chosen ciphertext attack.*

Both theorems are proved by constructing encryption schemes in which the adversary's gain from a chosen ciphertext attack is eliminated by making it infeasible (for the adversary) to obtain any useful knowledge via such an attack. In the case of private-key schemes (i.e., Theorem 5.3), this is achieved by making it infeasible (for the adversary) to produce legitimate ciphertexts (other than those explicitly given to it, in response to its request to encrypt plaintexts of its choice). This, in turn, is achieved by augmenting the ciphertext with an "authentication tag" that is hard to generate without knowledge of the encryption-key; that is, we use a message-authentication scheme (as defined in Section 6). In the case of public-key schemes (i.e., Theorem 5.4), the adversary can certainly generate ciphertexts by itself, and the aim is to to make it infeasible (for the adversary) to produce legitimate ciphertexts without "knowing" the corresponding plaintext. This, in turn, will be achieved by augmenting the plaintext with a non-interactive zero-knowledge "proof of knowledge" of the corresponding plaintext.

Security against chosen ciphertext attack is related to the notion of *non-malleability* of the encryption scheme (cf. [52]). Loosely speaking, in a non-malleable encryption scheme it is infeasible for an adversary, given a ciphertext, to produce a valid ciphertext for a related plaintext (e.g., given a ciphertext of a plaintext $1x$, for an unknown x, it is infeasible to produce a ciphertext to the plaintext $0x$). For further discussion see [52, 19, 92].

6 Signature and Message Authentication Schemes

Both signature schemes and message authentication schemes are methods for "validating" data; that is, verifying that the data was approved by a certain party (or set of parties). The difference between signature schemes and message authentication schemes is that signatures should be "universally verifiable", whereas authentication tags are only required to be verifiable by parties that are also able to generate them.

Signature Schemes: The need to discuss "digital signatures" [51, 112] has arisen with the introduction of computer communication to the business environment (in which parties need to commit themselves to proposals and/or declarations that they make). Discussions of "unforgeable signatures" did take place also in previous centuries, but the objects of discussion were handwritten signatures (and not digital ones), and the discussion was not perceived as related to "cryptography". Loosely speaking, a scheme for unforgeable signatures should satisfy the following:

- each user can *efficiently produce its own signature* on documents of its choice;
- every user can *efficiently verify* whether a given string is a signature of another (specific) user on a specific document; but
- *it is infeasible to produce signatures of other users* to documents they did not sign.

We note that the formulation of unforgeable digital signatures provides also a clear statement of the essential ingredients of handwritten signatures. The ingredients are each person's ability to sign for

[21] Loosely speaking, the enhancement refers to the hardness condition of Definition 2.2, and requires that it be hard to recover $f_i^{-1}(y)$ also when given the coins used to sample y (rather than merely y itself). See [68, Apdx. C.1].

itself, a universally agreed verification procedure, and the belief (or assertion) that it is infeasible (or at least hard) to forge signatures (i.e., produce some other person's signatures to documents that were not signed by it such that these "unauthentic" signatures are accepted by the verification procedure). It is not clear to what extent handwritten signatures meet these requirements. In contrast, our discussion of digital signatures provides precise statements concerning the extent to which digital signatures meet the above requirements. Furthermore, unforgeable digital signature schemes can be constructed based on some reasonable computational assumptions (i.e., the existence of one-way functions).

Message authentication schemes: Message authentication is a task related to the setting considered for encryption schemes; that is, communication over an insecure channel. This time, we consider an active adversary that is monitoring the channel and may alter the messages sent on it. The parties communicating through this insecure channel wish to authenticate the messages they send so that their counterpart can tell an original message (sent by the sender) from a modified one (i.e., modified by the adversary). Loosely speaking, a scheme for message authentication should satisfy the following:

- each of the communicating parties can *efficiently produce an authentication tag* to any message of its choice;
- each of the communicating parties can *efficiently verify* whether a given string is an authentication tag of a given message; but
- *it is infeasible for an external adversary* (i.e., a party other than the communicating parties) *to produce authentication tags* to messages not sent by the communicating parties.

Note that in contrast to the specification of signature schemes we do not require universal verification: only the designated receiver is required to be able to verify the authentication tags. Furthermore, we do not require that the receiver can not produce authentication tags by itself (i.e., we only require that *external parties* can not do so). Thus, message authentication schemes cannot convince *a third party* that the sender has indeed sent the information (rather than the receiver having generated it by itself). In contrast, signatures can be used to convince third parties: in fact, a signature to a document is typically sent to a second party so that in the future this party may (by merely presenting the signed document) convince third parties that the document was indeed generated (or sent or approved) by the signer.

6.1 Definitions

Formally speaking, both signature schemes and message authentication schemes consist of three efficient algorithms: key generation, signing and verification. As in the case of encryption schemes, the key-generation algorithm is used to generate a pair of corresponding keys, one is used for signing and the other is used for verification. The difference between the two types of schemes is reflected in the definition of security. In the case of signature schemes, the adversary is given the verification-key, whereas in the case of message authentication schemes the verification-key (which may equal the signing-key) is not given to the adversary. Thus, schemes for message authentication can be viewed as a private-key version of signature schemes. This difference yields different functionalities (even more than in the case of encryption): In typical use of a signature scheme, each user generates a pair of signing and verification keys, publicizes the verification-key and keeps the signing-key secret. Subsequently, each user may sign documents using its own signing-key, and these signatures are *universally verifiable* with respect to its public verification-key. In contrast, message authentication schemes are typically used to authenticate information sent among a set of *mutually trusting* parties that agree on a secret key, which is being used both to produce and verify authentication-tags. (Indeed, it is assumed that the mutually trusting parties have generated the key together or have exchanged the key in a secure way, prior to the communication of information that needs to be authenticated.)

We focus on the definition of secure signature schemes. Following Goldwasser, Micali and Rivest [84], we consider very powerful attacks on the signature scheme as well as a very liberal notion of breaking it. Specifically, the attacker is allowed to obtain signatures to any message of its choice. One may argue that in many applications such a general attack is not possible (because messages to be signed must have a specific format). Yet, our view is that it is impossible to define a general (i.e., application-independent) notion of admissible messages, and thus a general/robust definition of an attack seems to have to be formulated as suggested here. (Note that at worst, our approach is overly cautious.) Likewise, the adversary is said to be successful if it can produce a valid signature to *any* message for which it has not asked for a signature during its attack. Again, this refers to the ability to form signatures to possibly "nonsensical" messages as a breaking of the scheme. Yet, again, we see no way to have a general (i.e., application-independent) notion of "meaningful" messages (so that only forging signatures to them will be considered a breaking of the scheme).

Definition 6.1 (secure signature schemes – a sketch): *A* chosen message attack *is a process that, on input a verification-key, can obtain signatures* (relative to the corresponding signing-key) *to messages of its choice. Such an attack is said to* succeed (in existential forgery) *if it outputs a valid signature to a message for which it has* NOT *requested a signature during the attack. A signature scheme is* secure (or unforgeable) *if every* feasible *chosen message attack succeeds with at most negligible probability, where the probability is taken over the initial choice of the key-pair as well as over the adversary's actions.*

We stress that *plain* RSA (alike plain versions of Rabin's scheme [113] and the DSS [108]) is not secure under the above definition. However, it may be secure if the message is "randomized" before RSA (or the other schemes) is applied.

6.2 Constructions

Secure *message authentication schemes* can be constructed using pseudorandom functions [70]. Specifically, the key-generation algorithm consists of selecting a seed $s \in \{0,1\}^n$ for such a function, denoted $f_s : \{0,1\}^* \to \{0,1\}^n$, and the (only valid) tag of message x with respect to the key s is $f_s(x)$. As in the case of our private-key encryption scheme, the proof of security of the current message authentication scheme consists of two steps:

1. Prove that an idealized version of the scheme, in which one uses a uniformly selected function $F:\{0,1\}^* \to \{0,1\}^n$, rather than the pseudorandom function f_s, is secure (i.e., unforgeable).

2. Conclude that the real scheme (as presented above) is secure (because, otherwise one could distinguish a pseudorandom function from a truly random one).

Note that the aforementioned message authentication scheme makes an "extensive use of pseudorandom functions" (i.e., the pseudorandom function is applied directly to the message, which requires a generalized notion of pseudorandom functions (cf. Section 3.3)). More efficient schemes may be obtained either based on a more restricted use of a pseudorandom function (cf., e.g., [17]) or based on other cryptographic primitives (cf., e.g., [96]).

Constructing secure *signature schemes* seems more difficult than constructing message authentication schemes. Nevertheless, secure signature schemes can be constructed based on any one-way function. Furthermore:

Theorem 6.2 ([106, 119], see [68, Sec. 6.4]): *The following three conditions are equivalent.*

1. *One-way functions exist.*
2. *Secure signature schemes exist.*
3. *Secure message authentication schemes exist.*

We stress that, unlike in the case of public-key encryption schemes, the construction of signature schemes (which may be viewed as a public-key analogue of message authentication) does not use a trapdoor property.

How to construct secure signature schemes

Three central paradigms used in the construction of secure *signature schemes* are the "refreshing" of the "effective" signing-key, the usage of an "authentication tree", and the "hashing paradigm" (all to be discussed in the sequel). In addition to being used in the proof of Theorem 6.2, all three paradigms are also of independent interest.

The refreshing paradigm. Introduced in [84], the *refreshing paradigm* is aimed at limiting the potential dangers of chosen message attacks. This is achieved by signing the actual document using a newly (randomly) generated instance of the signature scheme, and authenticating (the verification-key of) this random instance with respect to the fixed public-key. That is, consider a basic signature scheme (G, S, V) used as follows. Suppose that the user U has generated a key-pair, $(s, v) \leftarrow G(1^n)$, and has placed the verification-key v on a public-file. When a party asks U to sign some document α, the user U generates a new ("fresh") key-pair, $(s', v') \leftarrow G(1^n)$, signs v' using the original signing-key s, signs α using the new signing-key s', and presents $(S_s(v'), v', S_{s'}(\alpha))$ as a signature to α. An alleged signature, (β_1, v', β_2), is verified by checking whether both $V_v(v', \beta_1) = 1$ and $V_{v'}(\alpha, \beta_2) = 1$ hold. Intuitively, the gain in terms of security is that a full-fledged chosen message attack cannot be launched on a fixed instance of (G, S, V) (i.e., on the fixed verification-key that resides in the public-file and is known to the attacker). All that an attacker may obtain (via a chosen message attack on the new scheme) is signatures, relative to the original signing-key s of (G, S, V), to random strings (distributed according to $G(1^n)$) as well as additional signatures that are each relative to a random and independently distributed signing-key.

Authentication trees. The security benefits of the refreshing paradigm are increased when combining it with the use of *authentication trees*, as introduced in [101]. The idea is to use the public verification-key in order to authenticate several (e.g., two) fresh instances of the signature scheme, use each of these instances to authenticate several additional fresh instances, and so on. We obtain a tree of fresh instances of the basic signature scheme, where each internal node authenticates its children. We can now use the leaves of this tree in order to sign actual documents, where each leaf is used at most once. Thus, a signature to an actual document consists of (1) a signature to this document authenticated with respect to the verification-key associated with some leaf, and (2) a sequence of verification-keys associated with the nodes along the path from the root to this leaf, where each such verification-key is authenticated with respect to the verification-key of its parent. We stress that (by suitable implementation to be discussed below) each instance of the signature scheme is used to sign at most one string (i.e., a single sequence of verification-keys if the instance resides in an internal node, and an actual document if the instance resides in a leaf). Thus, it suffices to use a signature scheme that is secure as long as it is used to legitimately sign a single string. Such signature schemes, called one-time signature schemes and introduced in [112], are easier to construct than standard signature schemes, especially if one only wishes to sign strings that are significantly shorter than the signing-key (resp., than the verification-key). For example, using a one-way function f, we may let the signing-key consist of a sequence of n pairs of strings, let the corresponding verification-key consist of the corresponding sequence of images of f, and sign an n-bit long message by revealing the adequate pre-images.[22]

The hashing paradigm. Note, however, that in the aforementioned authentication-tree, the instances of the signature scheme (associated with internal nodes) are used to sign a pair of verification-keys. Thus, we need a one-time signature scheme that can be used for signing messages that are longer than the verification-key. Here is where the *hashing paradigm* comes into play. This paradigm refers to the common practice of signing documents via a two stage process: First the actual document is hashed to a (relatively) short bit string, and next the basic signature scheme is applied to the resulting string. This practice (as well as other usages of the hashing paradigm) is sound provided that the hashing function belongs to a family of *collision-free hashing* functions (i.e., loosely speaking, given a random hash function in the family, it is infeasible to find two different strings that are hashed by this function to the same value; cf. [48]). (A variant of the *hashing paradigm* uses the weaker notion of a family of *Universal One-Way Hash Functions* (cf. [106]), which in turn can be constructed using any one-way function [106, 119].)

Implementation details. In order to implement the aforementioned (full-fledged) signature scheme one needs to store in (secure) memory all the instances of the basic (one-time) signature scheme that are generated throughout the entire signing process (which refers to numerous documents). This can be done by extending the model so to allow for memory-dependent signature schemes. Alternatively, we note that all that we need to store are the random-coins used for generating each of these instances, and the former can be determined by a pseudorandom function (applied to the name of the corresponding vertex in the tree). Indeed, the seed of this pseudorandom function will be part of the signing-key of the resulting (full-fledged) signature scheme.

6.3 Public-Key Infrastructure

The standard use of public-key encryption schemes (resp., signature schemes) in real-life communication requires a mechanism for providing the sender (resp., signature verifier) with the receiver's authentic encryption-key (resp., signer's authentic verification-key). Specifically, this problem arises in large-scale systems, where typically the sender (resp., verifier) does not have a local record of the receiver's encryption-key (resp., signer's verification-key), and so must obtain this key in a "reliable" way (i.e., typically, certified by some trusted authority). In most theoretical works, one assumes that the keys are posted on and can be retrieved from a public-file that is maintained by a trusted party (which makes sure that each user can post only keys bearing its own identity). In practice, maintaining such a public-file is a major problem, and mechanisms that implement this

[22] That is, the signing-key consist of a sequence $((s_1^0, s_1^1), ..., (s_n^0, s_n^1)) \in \{0,1\}^{2n^2}$, the corresponding verification-key is $(f(s_1^0), f(s_1^1)), ..., (f(s_n^0), f(s_n^1)))$, and the signature of the message $\sigma_1 \cdots \sigma_n$ is $(s_1^{\sigma_1}, ..., s_n^{\sigma_n})$.

abstraction are typically referred to by the generic term "public-key infrastructure (PKI)". For a discussion of the practical problems regarding PKI deployment see, e.g., [100, Chap. 13].

7 General Cryptographic Protocols

The design of secure protocols that implement arbitrary desired functionalities is a major part of modern cryptography. Taking the opposite perspective, the design of any cryptographic scheme may be viewed as the design of a secure protocol for implementing a suitable functionality. Still, we believe that it makes sense to differentiate between basic cryptographic primitives (which involve little interaction) like encryption and signature schemes on one hand, and general cryptographic protocols on the other hand.

We survey *general* results concerning secure *multi*-party computations, where the *two*-party case is an important special case. In a nutshell, these results assert that one can construct protocols for securely computing *any* desirable multi-party functionality. Indeed, what is striking about these results is their generality, and we believe that the wonder is not diminished by the (various alternative) conditions under which these results hold.

Our focus on the *general* study of secure multi-party computation (rather than on protocols for solving specific problems) is natural in the context of the theoretical treatment of the subject matter. We wish to highlight the importance of this *general* study to practice. Firstly, this study clarifies fundamental issues regarding security in a multi-party environment. Secondly, it draws the lines between what is possible in principle and what is not. Thirdly, it develops general techniques for designing secure protocols. And last, sometimes, it may even yield schemes (or modules) that may be incorporated in practical systems.

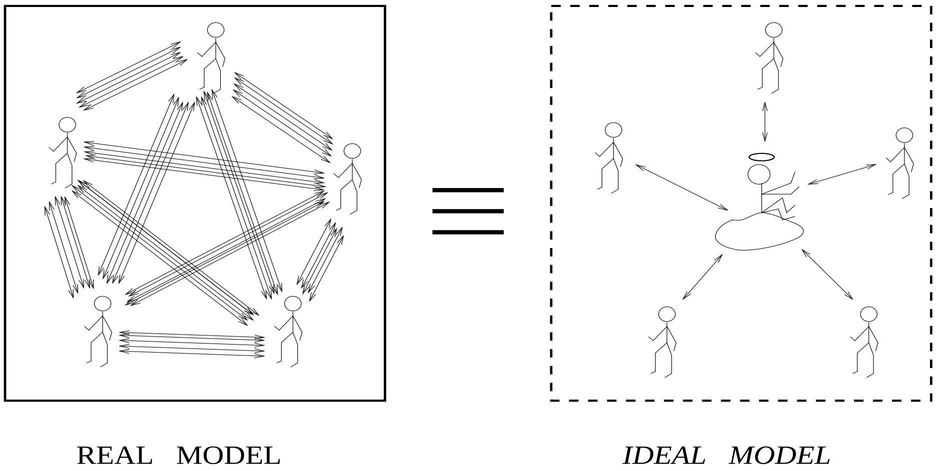


Figure 9: Secure protocols emulate a trusted party – an illustration.

A general framework for casting (m-party) cryptographic (protocol) problems consists of specifying a random process[23] that maps m inputs to m outputs. The inputs to the process are to be

[23] That is, we consider the secure evaluation of randomized functionalities, rather than "only" the secure evaluation of functions. Specifically, we consider an arbitrary (randomized) process F that on input $(x_1, ..., x_m)$, first selects at random (depending only on $\ell \stackrel{\text{def}}{=} \sum_{i=1}^{m} |x_i|$) an m-ary function f, and then outputs the m-tuple $f(x_1, ..., x_m) = (f_1(x_1, ..., x_m), ..., f_m(x_1, ..., x_m))$. In other words, $F(x_1, ..., x_m) = F'(r, x_1, ..., x_m)$, where r is uniformly selected in $\{0,1\}^{\ell'}$ (with $\ell' = \text{poly}(\ell)$), and F' is a function mapping $(m+1)$-long sequences to m-long sequences.

thought of as the local inputs of m parties, and the m outputs are their corresponding (desired) local outputs. The random process describes the desired functionality. That is, if the m parties were to trust each other (or trust some external party), then they could each send their local input to the trusted party, who would compute the outcome of the process and send to each party the corresponding output. A pivotal question in the area of cryptographic protocols is to what extent can this (imaginary) trusted party be "emulated" by the mutually distrustful parties themselves.

The results surveyed below describe a variety of models in which such an "emulation" is possible. The models vary by the underlying assumptions regarding the communication channels, numerous parameters relating to the extent of adversarial behavior, and the desired level of emulation of the trusted party (i.e., level of "security").

Organization: Section 7.1 provides a rather comprehensive survey of the various definitions used in the area of secure multi-party computation, whereas Section 7.2 similarly surveys the known results. However, some readers may prefer to first consider one concrete case of the definitional approach, as provided in Section 7.1.2, and proceed directly to see some constructions. Indeed, a few constructions are sketched in Section 7.3. All the above refers to the security of stand-alone executions, and the preservation of security in an environment in which many executions of many protocols are being attacked is considered in Section 7.4.

7.1 The Definitional Approach and Some Models

Before describing the aforementioned results, we further discuss the notion of "emulating a trusted party", which underlies the definitional approach to secure multi-party computation (as initiated and developed in [81, 103, 13, 14, 35, 36]) The approach can be traced back to the definition of zero-knowledge (cf. [83]), and even to the definition of secure encryption (cf. [64], rephrasing [82]). The underlying paradigm (called the simulation paradigm (cf. Section 4.1)) is that a scheme is secure if whatever a feasible adversary can obtain after attacking it, is also feasibly attainable "from scratch". In the case of zero-knowledge this amounts to saying that whatever a (feasible) verifier can obtain after interacting with the prover on a prescribed valid assertion, can be (feasibly) computed from the assertion itself. In the case of multi-party computation we compare the effect of adversaries that participate in the execution of the actual protocol to the effect of adversaries that participate in an imaginary execution of a trivial (ideal) protocol for computing the desired functionality with the help of a trusted party. If whatever the adversaries can feasibly obtain in the former real setting can also be feasibly obtained in the latter ideal setting then the protocol "emulates the ideal setting" (i.e., "emulates a trusted party"), and so is deemed secure. This basic approach can be applied in a variety of models, and is used to define the goals of security in these models.[24] We first discuss some of the parameters used in defining various models, and next demonstrate the application of this approach in two important models. For further details, see [36] or [68, Sec. 7.2 and 7.5.1].

[24] A few technical comments are in place. Firstly, we assume that the inputs of all parties are of the same length. We comment that as long as the lengths of the inputs are polynomially related, the above convention can be enforced by padding. On the other hand, some length restriction is essential for the security results, because in general it is impossible to hide all information regarding the length of the inputs to a protocol. Secondly, we assume that the desired functionality is computable in probabilistic polynomial-time, because we wish the secure protocol to run in probabilistic polynomial-time (and a protocol cannot be more efficient than the corresponding centralized algorithm). Clearly, the results can be extended to functionalities that are computable within any given (time-constructible) time bound, using adequate padding.

7.1.1 Some parameters used in defining security models

The following parameters are described in terms of the actual (or real) computation. In *some cases*, the corresponding definition of security is obtained by imposing some restrictions or provisions on the ideal model. For example, in the case of two-party computation (see below), secure computation is possible only if premature termination is *not* considered a breach of security. In that case, the suitable security definition is obtained (via the simulation paradigm) by allowing (an analogue of) premature termination in the ideal model. In *all cases*, the desired notion of security is defined by requiring that for any adequate adversary in the real model, there exist a corresponding adversary in the corresponding ideal model that obtains essentially the same impact (as the real-model adversary).

The communication channels: The parameters of the model include questions like whether or not the channels may be tapped by an adversary, whether or not they are tamper-free, and questions referring to the network behavior (in the case of multi-party protocols).

Wire-tapping versus the private-channel model: The standard assumption in cryptography is that the adversary may tap all communication channels (between honest parties). In contrast, one may *postulate* that the adversary cannot obtain messages sent between a pair of honest parties, yielding the so-called *private-channel model* (cf. [26, 43]). The latter postulate may be justified in some settings. Furthermore, it may be viewed as a useful abstraction that provides a clean model for the study and development of secure protocols. In this respect, it is important to mention that, in a variety of settings of the other parameters, private channels can be easily emulated by ordinary "tapped channels".

Broadcast channel: In the multi-party context, one may postulate the existence of a *broadcast channel* (cf. [115]), and the motivation and justifications are as in the case of the private-channel model.

The *tamper-free assumption*: The standard assumption in the area is that the adversary cannot modify, duplicate, or generate messages sent over the communication channels (between honest parties). Again, this assumption can be justified in some settings and can be emulated in others (cf., [18, 37]).

Network behavior: Most works in the area assume that communication is *synchronous* and that point-to-point channels exist between every pair of processors (i.e., a *complete network*). However, one may also consider *asynchronous communication* (cf. [24]) and *arbitrary networks* of point-to-point channels (cf. [53]).

Set-up assumptions: Unless stated differently, we make no set-up assumptions (except for the obvious assumption that all parties have identical copies of the protocol's program). However, in some cases it is assumed that each party knows a verification-key corresponding to each of the other parties (or that a public-key infrastructure is available). Another assumption, made more rarely, is that all parties have access to some common (trusted) random string.

Computational limitations: Typically, we consider computationally-bounded adversaries (e.g., probabilistic polynomial-time adversaries). However, the private-channel model allows for the (meaningful) consideration of computationally-unbounded adversaries.

We stress that, also in the case of computationally-unbounded adversaries, security should be defined by requiring that for every real adversary, whatever the adversary can compute after participating in the execution of the actual protocol is computable *within comparable* *time* by an imaginary adversary participating in an imaginary execution of the trivial ideal protocol (for computing the desired functionality with the help of a trusted party). That is, although no computational restrictions are made on the real-model adversary, it is required that the ideal-model adversary that obtains the same impact does so within comparable time (i.e., within time that is polynomially related to the running time of the real-model adversary being simulated). Thus, any construction proven secure in the computationally-unbounded adversary model is (trivially) secure with respect to computationally-bounded adversaries.

Restricted adversarial behavior: The parameters of the model include questions like whether or not the adversary is "adaptive" and "active" (where these terms are discussed next).

Adaptive versus non-adaptive: The most general type of an adversary considered in the literature is one that may corrupt parties to the protocol while the execution goes on, and does so based on partial information it has gathered so far (cf., [38]). A somewhat more restricted model, which seems adequate in many settings, postulates that the set of dishonest parties is fixed (arbitrarily) before the execution starts (but this set is, of course, not known to the honest parties). The latter model is called *non-adaptive* as opposed to the *adaptive* adversary discussed first. Although the adaptive model is stronger, the author believes that the non-adaptive model provides a reasonable level of security in many applications.

Active versus passive: An orthogonal parameter of restriction refers to whether a dishonest party takes active steps to disrupt the execution of the protocol (i.e., sends messages that differ from those specified by the protocol), or merely gathers information (which it may latter share with the other dishonest parties). The latter adversary has been given a variety of names such as *semi-honest*, *passive*, and *honest-but-curious*. This restricted model may be justified in certain settings, and certainly provides a useful methodological locus (cf., [75, 76, 65] and Section 7.3). Below we refer to the adversary of the unrestricted model as to *active*; another commonly used name is *malicious*.

Restricted notions of security: One important example is the willingness to tolerate "unfair" protocols in which the execution can be suspended (at any time) by a dishonest party, provided that it is detected doing so. We stress that in case the execution is suspended, the dishonest party does not obtain more information than it could have obtained when not suspending the execution. (What may happen is that the honest parties will not obtain their desired outputs, but rather will detect that the execution was suspended.) We stress that the motivation to this restricted model is the impossibility of obtaining general secure two-party computation in the unrestricted model.

Upper bounds on the number of dishonest parties: In some models, secure multi-party computation is possible only if a majority of the parties is honest (cf., [26, 45]). Sometimes even a special majority (e.g., 2/3) is required. General "(resilient) adversarial-structures" have been considered too (cf. [88]).

Mobile adversary: In most works, once a party is said to be dishonest it remains so throughout the execution. More generally, one may consider transient adversarial behavior (e.g., an adversary seizes control of some site and later withdraws from it). This model, introduced in [110], allows to construct protocols that remain secure even in case the adversary may seize control of all sites during the execution (but never control concurrently, say, more than 10% of the sites). We comment that schemes secure in this model were later termed "proactive" (cf., [40]).

7.1.2 Example: Multi-party protocols with honest majority

Here we consider an active, non-adaptive, computationally-bounded adversary, and do not assume the existence of private channels. Our aim is to define multi-party protocols that remain secure provided that the honest parties are in majority. (The reason for requiring a honest majority will be discussed at the end of this subsection.)

Consider any multi-party protocol. We first observe that each party may change its local input before even entering the execution of the protocol. However, this is unavoidable also when the parties utilize a trusted party. Consequently, such an effect of the adversary on the real execution (i.e., modification of its own input prior to entering the actual execution) is not considered a breach of security. In general, whatever cannot be avoided when the parties utilize a trusted party, is not considered a breach of security. We wish secure protocols (in the real model) to suffer only from whatever is unavoidable also when the parties utilize a trusted party. Thus, the basic paradigm underlying the definitions of *secure multi-party computations* amounts to requiring that the only situations that may occur in the real execution of a secure protocol are those that can also occur in a corresponding ideal model (where the parties may employ a trusted party). In other words, the "effective malfunctioning" of parties in secure protocols is restricted to what is postulated in the corresponding ideal model.

When defining secure multi-party protocols with honest majority, we need to pin-point what cannot be avoided in the ideal model (i.e., when the parties utilize a trusted party). This is easy, because the ideal model is very simple. Since we are interested in executions in which the majority of parties are honest, we consider an ideal model in which any minority group (of the parties) may collude as follows:

1. Firstly this dishonest minority shares its original inputs and decides together on replaced inputs to be sent to the trusted party. (The other parties send their respective original inputs to the trusted party.)

2. Upon receiving inputs from all parties, the trusted party determines the corresponding outputs and sends them to the corresponding parties. (We stress that the information sent between the honest parties and the trusted party is not seen by the dishonest colluding minority.)

3. Upon receiving the output-message from the trusted party, each honest party outputs it locally, whereas the dishonest colluding minority may determine their outputs based on all they know (i.e., their initial inputs and their received outputs).

Note that the above behavior of the minority group is unavoidable in any execution of any protocol (even in presence of trusted parties). This is the reason that the ideal model was defined as above. Now, a *secure multi-party computation with honest majority* is required to emulate this ideal model. That is, the effect of any feasible adversary that controls a minority of the parties in a real execution of the actual protocol, can be essentially simulated by a (different) feasible adversary that controls the corresponding parties in the ideal model. That is:

Definition 7.1 (secure protocols – a sketch): *Let f be an m-ary functionality and Π be an m-party protocol operating in the real model.*

- *For a real-model adversary A, controlling some minority of the parties* (and tapping all communication channels)*, and an m-sequence $\overline{x}$, we denote by $\text{REAL}_{\Pi,A}(\overline{x})$ the sequence of m outputs resulting from the execution of Π on input $\overline{x}$ under attack of the adversary A.*

- *For an ideal-model adversary A', controlling some minority of the parties, and an m-sequence $\overline{x}$, we denote by* $\text{IDEAL}_{f,A'}(\overline{x})$ *the sequence of m outputs resulting from the ideal process described above, on input $\overline{x}$ under attack of the adversary A'.*

We say that Π securely implements f with honest majority *if for every feasible real-model adversary A, controlling some minority of the parties, there exists a feasible ideal-model adversary A', controlling the same parties, so that the probability ensembles* $\{\text{REAL}_{\Pi,A}(\overline{x})\}_{\overline{x}}$ *and* $\{\text{IDEAL}_{f,A'}(\overline{x})\}_{\overline{x}}$ *are computationally indistinguishable (as in Footnote 8).*

Thus, security means that the effect of each minority group in a real execution of a secure protocol is "essentially restricted" to replacing its own local inputs (independently of the local inputs of the majority parties) before the protocol starts, and replacing its own local outputs (depending only on its local inputs and outputs) after the protocol terminates. (We stress that in the real execution the minority parties do obtain additional pieces of information; yet in a secure protocol they gain nothing from these additional pieces of information, because they can actually reproduce those by themselves.)

The fact that Definition 7.1 refers to a model without private channels is due to the fact that our (sketchy) definition of the real-model adversary allowed it to tap the channels, which in turn effects the set of possible ensembles $\{\text{REAL}_{\Pi,A}(\overline{x})\}_{\overline{x}}$. When defining security in the private-channel model, the real-model adversary is not allowed to tap channels between honest parties, and this again effects the possible ensembles $\{\text{REAL}_{\Pi,A}(\overline{x})\}_{\overline{x}}$. On the other hand, when we wish to define security with respect to passive adversaries, both the scope of the real-model adversaries and the scope of the ideal-model adversaries changes. In the real-model execution, all parties follow the protocol but the adversary may alter the output of the dishonest parties arbitrarily depending on all their intermediate internal states (during the execution). In the corresponding ideal-model, the adversary is not allowed to modify the *inputs* of dishonest parties (in Step 1), but is allowed to modify their outputs (in Step 3).

We comment that a definition analogous to Definition 7.1 can be presented also in case the dishonest parties are not in minority. In fact, such a definition seems more natural, but the problem is that such a definition cannot be satisfied. That is, most natural functionalities do not have a protocol for computing them securely in case at least half of the parties are dishonest and employ an adequate adversarial strategy. This follows from an impossibility result regarding two-party computation, which essentially asserts that there is no way to prevent a party from prematurely suspending the execution [47]. On the other hand, secure multi-party computation with dishonest majority is possible if premature suspension of the execution is not considered a breach of security (cf. Section 7.1.3).

7.1.3 Another example: Two-party protocols allowing abort

In light of the last paragraph, we now consider multi-party computations in which premature suspension of the execution is not considered a breach of security. For concreteness, we focus here on the special case of two-party computations.[25]

Intuitively, in any two-party protocol, each party may suspend the execution at any point in time, and furthermore it may do so as soon as it learns the desired output. Thus, in case the output of each parties depends on both inputs, it is always possible for one of the parties to obtain the desired output while preventing the other party from fully determining its own output. The same phenomenon occurs even in case the two parties just wish to generate a common random

[25] As in Section 7.1.2, we consider a non-adaptive, active, computationally-bounded adversary.

value. Thus, when considering active adversaries in the two-party setting, we do not consider such premature suspension of the execution a breach of security. Consequently, we consider an ideal model where each of the two parties may "shut-down" the trusted (third) party at any point in time. In particular, this may happen after the trusted party has supplied the outcome of the computation to one party but before it has supplied it to the other. That is, an execution in the ideal model proceeds as follows:

1. Each party sends its input to the trusted party, where the dishonest party may replace its input or send no input at all (which can be treated as sending a default value).

2. Upon receiving inputs from both parties, the trusted party determines the corresponding outputs, and sends the first output to the first party.

3. In case the first party is dishonest, it may instruct the trusted party to halt, otherwise it always instructs the trusted party to proceed. If instructed to proceed, the trusted party sends the second output to the second party.

4. Upon receiving the output-message from the trusted party, the honest party outputs it locally, whereas the dishonest party may determine its output based on all it knows (i.e., its initial input and its received output).

A secure two-party computation allowing abort is required to emulate this ideal model. That is, as in Definition 7.1, security is defined by requiring that for every feasible real-model adversary A, there exists a feasible ideal-model adversary A', controlling the same party, so that the probability ensembles representing the corresponding (real and ideal) executions are computationally indistinguishable. This means that each party's "effective malfunctioning" in a secure protocol is restricted to supplying an initial input of its choice and aborting the computation at any point in time. (Needless to say, the choice of the initial input of each party may *not* depend on the input of the other party.)

We mention that an alternative way of dealing with the problem of premature suspension of execution (i.e., abort) is to restrict our attention to single-output functionalities; that is, functionalities in which only one party is supposed to obtain an output. The definition of secure computation of such functionalities can be made identical to Definition 7.1, with the exception that no restriction is made on the set of dishonest parties (and in particular one may consider a single dishonest party in the case of two-party protocols). For further details, see [68, Sec. 7.2.3].

7.2 Some Known Results

We next list some of the models for which general secure multi-party computation is known to be attainable (i.e., models in which one can construct secure multi-party protocols for computing any desired functionality). We mention that the first results of this type were obtained by Goldreich, Micali, Wigderson and Yao [75, 129, 76].

- *Assuming the existence of enhanced*[26] *trapdoor permutations*, secure multi-party computation is possible in the following models (cf. [75, 129, 76] and details in [65, 68]):

  1. Passive adversary, for any number of dishonest parties (cf. [68, Sec. 7.3]).
  2. Active adversary that may control only a minority of the parties (cf. [68, Sec. 7.5.4]).

[26] See Footnote 21.

3. Active adversary, for any number of bad parties, provided that suspension of execution is not considered a violation of security (i.e., as discussed in Section 7.1.3). (See [68, Sec. 7.4 and 7.5.5].)

In all these cases, the adversary is computationally-bounded and non-adaptive. On the other hand, the adversary may tap the communication lines between honest parties (i.e., we do not assume "private channels" here). The results for active adversaries assume a broadcast channel. Indeed, the latter can be implemented (while tolerating any number of bad parties) using a signature scheme and assuming a public-key infrastructure (or that each party knows the verification-key corresponding to each of the other parties).

- Making no computational assumptions and allowing computationally-unbounded adversaries, but *assuming private channels*, secure multi-party computation is possible in the following models (cf. [26, 43]):

  1. Passive adversary that may control only a minority of the parties.
  2. Active adversary that may control only less than one third of the parties.[27]

  In both cases the adversary may be adaptive (cf. [26, 38]).

- Secure multi-party computation is possible against an active, adaptive and *mobile* adversary that may control a small constant fraction of the parties at any point in time [110]. This result makes no computational assumptions, allows computationally-unbounded adversaries, but *assumes private channels*.

- *Assuming the existence of trapdoor permutations*, secure multi-party computation is possible in a model allowing an active and *adaptive* computationally-bounded adversary that may control only less than one third of the parties [38, 49]. We stress that this result does not assume "private channels".

Results for asynchronous communication and arbitrary networks of point-to-point channels were presented in [24, 27] and [53], respectively.

Note that the implementation of a broadcast channel can be cast as a cryptographic protocol problem (i.e., for the functionality $(v, \lambda, ..., \lambda) \mapsto (v, v, ..., v)$, where λ denotes the empty string). Thus, it is not surprising that the results regarding active adversaries either assume the existence of such a channel or require a setting in which the latter can be implemented.

Secure reactive computation: All the above results (easily) extend to a reactive model of computation in which each party interacts with a high-level process (or application). The high-level process supplies each party with a sequence of inputs, one at a time, and expect to receive corresponding outputs from the parties. That is, a reactive system goes through (a possibly unbounded number of) iterations of the following type:

- Parties are given inputs for the current iteration.

- Depending on the current inputs, the parties are supposed to compute outputs for the current iteration. That is, the outputs in iteration j are determined by the inputs of the jth iteration.

[27] Fault-tolerance can be increased to a regular minority if a broadcast channel exists [115].

A more general formulation allows the outputs of each iteration to depend also on a global state, which is possibly updated in each iteration. The global state may include all inputs and outputs of previous iterations, and may only be partially known to individual parties. (In a secure reactive computation such a global state may be maintained by all parties in a "secret sharing" manner.) For further discussion, see [68, Sec. 7.7.1].

Efficiency considerations: One important efficiency measure regarding protocols is the number of communication rounds in their execution. The results mentioned above were originally obtained using protocols that use an unbounded number of rounds. In some cases, subsequent works obtained secure constant-round protocols: for example, in the case of multi-party computations with honest majority (cf. [15]) and in the case of two-party computations allowing abort (cf. [97]). Other important efficiency considerations include the total number of bits sent in the execution of a protocol, and the local computation time. The (communication and computation) complexities of the protocols establishing the above results are related to the *computational* complexity of the computation, but alternative relations (e.g., where the complexities of the secure protocols are related to the (insecure) *communication* complexity of the computation) may be possible (cf. [105]).

Theory versus practice (or general versus specific): This primer is focused on presenting general notions and general feasibility results. Needless to say, practical solutions to specific problems (e.g., voting [86], secure payment systems [16], and threshold cryptosystems [62]) are typically derived by specific constructions (and not by applying general results of the abovementioned type). Still, the (abovementioned) general results are of great importance to practice because they characterize a wide class of security problems that are solvable in principle, and provide techniques that may be useful also towards constructing reasonable solutions to specific problems.

7.3 Construction Paradigms and Two Simple Protocols

We briefly sketch a couple of paradigms used in the construction of secure multi-party protocols. We focus on the construction of secure protocols for the model of computationally-bounded and non-adaptive adversaries [75, 129, 76]. These constructions proceed in two steps (see details in [65, 68]). First a secure protocol is presented for the model of passive adversaries (for any number of dishonest parties), and next such a protocol is "compiled" into a protocol that is secure in one of the two models of active adversaries (i.e., either in a model allowing the adversary to control only a minority of the parties or in a model in which premature suspension of the execution is not considered a violation of security). These two steps are presented in the following two corresponding subsections, in which we also present two relatively simple protocols for two specific tasks, which are used extensively in the general protocols.

Recall that in the model of passive adversaries, all parties follow the prescribed protocol, but at termination the adversary may alter the outputs of the dishonest parties depending on all their intermediate internal states (during the execution). Below, we refer to protocols that are secure in the model of passive (resp., active) adversaries by the term passively-secure (resp., actively-secure).

7.3.1 Passively-secure computation with shares

For any $m \geq 2$, suppose that m parties, each having a private input, wish to obtain the value of a predetermined m-argument function evaluated at their sequence of inputs. Below, we outline a passively-secure protocol for achieving this goal. We mention that the design of passively-secure multi-party protocol for any functionality (allowing different outputs to different parties as well as handling also randomized computations) reduces easily to the aforementioned task.

We assume that the parties hold a circuit for computing the value of the function on inputs of the adequate length, and that the circuit contains only `and` and `not` gates. The key idea is to have each party "secretly share" its input with everybody else, and "secretly transform" shares of the input wires of the circuit into shares of the output wires of the circuit, thus obtaining shares of the outputs (which allows for the reconstruction of the actual outputs). The value of each wire in the circuit is shared in a way such that all shares yield the value, whereas lacking even one of the shares keeps the value totally undetermined. That is, we use a simple secret sharing scheme (cf. [122]) such that a bit b is shared by a random sequence of m bits that sum-up to $b \bmod 2$. First, each party shares each of its input bits with all parties (by secretly sending each party a random value and setting its own share accordingly). Next, all parties jointly scan the circuit from its input wires to the output wires, processing each gate as follows:

- When encountering a gate, the parties already hold shares of the values of the wires entering the gate, and their aim is to obtain shares of the value of the wires exiting the gate.

- For a `not`-gate this is easy: the first party just flips the value of its share, and all other parties maintain their shares.

- Since an `and`-gate corresponds to multiplication modulo 2, the parties need to securely compute the following randomized functionality (in which the x_i's denote shares of one entry-wire, the y_i's denote shares of the second entry-wire, the z_i's denote shares of the exit-wire, and the shares indexed by i belongs to Party i):

$$\begin{aligned} ((x_1,y_1),...,(x_m,y_m)) \quad \mapsto \quad & (z_1,...,z_m) && (1)\\ & \text{where } \textstyle\sum_{i=1}^m z_i = (\sum_{i=1}^m x_i)\cdot(\sum_{i=1}^m y_i). && (2)\end{aligned}$$

  That is, the z_i's are random subject to Eq. (2).

Finally, the parties send their shares of each circuit-output wire to the designated party, which reconstructs the value of the corresponding bit. Thus, the parties have propagated shares of the input wires into shares of the output wires, by repeatedly conducting privately-secure computation of the m-ary functionality of Eq. (1) & (2). That is, securely evaluating the entire (arbitrary) circuit "reduces" to securely conducting a specific (very simple) multi-party computation. But things get even simpler: the key observation is that

$$\left(\sum_{i=1}^m x_i\right)\cdot\left(\sum_{i=1}^m y_i\right) \;=\; \sum_{i=1}^m x_i y_i + \sum_{1\le i<j\le m}(x_i y_j + x_j y_i) \qquad (3)$$

Thus, the m-ary functionality of Eq. (1) & (2) can be computed as follows (where all arithmetic operations are mod 2):

1. Each Party i locally computes $z_{i,i} \stackrel{\text{def}}{=} x_i y_i$.

2. Next, each pair of parties (i.e., Parties i and j) securely compute random shares of $x_i y_j + y_i x_j$. That is, Parties i and j (holding (x_i,y_i) and (x_j,y_j), respectively), need to securely compute the randomized two-party functionality $((x_i,y_i),(x_j,y_j)) \mapsto (z_{i,j},z_{j,i})$, where the z's are random subject to $z_{i,j}+z_{j,i} = x_i y_j + y_i x_j$. Equivalently, Party j uniformly selects $z_{j,i} \in \{0,1\}$,

and Parties i and j securely compute the deterministic functionality $((x_i, y_i), (x_j, y_j, z_{j,i})) \mapsto (z_{j,i} + x_i y_j + y_i x_j, \lambda)$, where λ denotes the empty string.

The latter simple two-party computation can be securely implemented using a 1-out-of-4 Oblivious Transfer (cf. [80] and [68, Sec. 7.3.3]), which in turn can be implemented using enhanced trapdoor permutations (see below). Loosely speaking, a 1-out-of-k Oblivious Transfer is a protocol enabling one party to obtain one of k secrets held by another party, without the second party learning which secret was obtained by the first party. That is, we refer to the two-party functionality

$$(i, (s_1, ..., s_k)) \mapsto (s_i, \lambda) \tag{4}$$

Note that any function $f : [k] \times \{0,1\}^* \to \{0,1\}^*$ can be privately-computed by invoking a 1-out-of-k Oblivious Transfer on inputs i and $(f(1, y), ..., f(k, y))$, where i (resp., y) is the initial input of the first (resp., second) party.

3. Finally, for every $i = 1, ..., m$, summing-up all the $z_{i,j}$'s yields the desired share of Party i.

The above construction is analogous to a construction that was briefly described in [76]. A detailed description and full proofs appear in [65, 68].

We mention that an analogous construction has been subsequently used in the private channel model and withstands computationally unbounded active (resp., passive) adversaries that control less than one third (resp., a minority) of the parties [26]. The basic idea is to use a more sophisticated secret sharing scheme; specifically, via a low degree polynomial [122]. That is, the Boolean circuit is viewed as an arithmetic circuit over a finite field having more than m elements, and a secret element s of the field is shared by selecting uniformly a polynomial of degree $d = \lfloor (m-1)/3 \rfloor$ (resp., degree $d = \lfloor (m-1)/2 \rfloor$) having a free-term equal to s, and handing each party the value of this polynomial evaluated at a different (fixed) point (e.g., party i is given the value at point i). Addition is emulated by (local) point-wise addition of the (secret sharing) polynomials representing the two inputs (using the fact that for polynomials p and q, and any field element e (and in particular $e = 0, 1, ..., m$), it holds that $p(e) + q(e) = (p + q)(e)$). The emulation of multiplication is more involved and requires interaction (because the product of polynomials yields a polynomial of higher degree, and thus the polynomial representing the output cannot be the product of the polynomials representing the two inputs). Indeed, the aim of the interaction is to turn the shares of the product polynomial into shares of a degree d polynomial that has the same free-term as the product polynomial (which is of degree $2d$). This can be done using the fact that the coefficients of a polynomial are a linear combination of its values at sufficiently many arguments (and the other way around), and the fact that one can privately-compute any linear combination (of secret values). For details see [26, 63].

A passively-secure 1-out-of-k Oblivious Transfer. Using a collection of enhanced trapdoor permutations, $\{f_\alpha : D_\alpha \to D_\alpha\}_{\alpha \in \overline{I}}$, we outline a passively-secure implementation of the functionality of Eq. (4). The implementation originates in [56] (and a full description is provided in [68, Sec. 7.3.2]).

Inputs: The sender has input $(\sigma_1, \sigma_2, ..., \sigma_k) \in \{0,1\}^k$, the receiver has input $i \in \{1, 2, ..., k\}$.

Step S1: The sender selects at random a permutation f_α along with a corresponding trapdoor, denoted t, and sends the permutation f_α (i.e., its index α) to the receiver.

Step R1: The receiver uniformly and independently selects $x_1, ..., x_k \in D_\alpha$, sets $y_i = f_\alpha(x_i)$ and $y_j = x_j$ for every $j \neq i$, and sends $(y_1, y_2, ..., y_k)$ to the sender.

Thus, the receiver knows $f_\alpha^{-1}(y_i) = x_i$, but cannot predict $b(f_\alpha^{-1}(y_j))$ for any $j \neq i$. Of course, the last assertion presumes that the receiver follows the protocol (i.e., is semi-honest).

Step S2: Upon receiving $(y_1, y_2, ..., y_k)$, using the inverting-with-trapdoor algorithm and the trapdoor t, the sender computes $z_j = f_\alpha^{-1}(y_j)$, for every $j \in \{1, ..., k\}$. It sends the k-tuple $(\sigma_1 \oplus b(z_1), \sigma_2 \oplus b(z_2), ..., \sigma_k \oplus b(z_k))$ to the receiver.

Step R2: Upon receiving $(c_1, c_2, ..., c_k)$, the receiver locally outputs $c_i \oplus b(x_i)$.

We first observe that the above protocol correctly computes 1-out-of-k Oblivious Transfer; that is, the receiver's local output (i.e., $c_i \oplus b(x_i)$) indeed equals $(\sigma_i \oplus b(f_\alpha^{-1}(f_\alpha(x_i)))) \oplus b(x_i) = \sigma_i$. Next, we offer some intuition as to why the above protocol constitutes a privately-secure implementation of 1-out-of-k Oblivious Transfer. Intuitively, the sender gets no information from the execution because, for any possible value of i, the senders sees the same distribution; specifically, a sequence of k uniformly and independently distributed elements of D_α. (Indeed, the key observation is that applying f_α to a uniformly distributed element of D_α yields a uniformly distributed element of D_α.) Intuitively, the receiver gains no computational knowledge from the execution because, for $j \neq i$, the only information that the receiver has regarding σ_j is the triplet $(\alpha, x_j, \sigma_j \oplus b(f_\alpha^{-1}(x_j)))$, where x_j is uniformly distributed in D_α, and from this information it is infeasible to predict σ_j better than by a random guess. The latter intuition presumes that sampling D_α is trivial (i.e., that there is an easily computable correspondence between the coins used for sampling and the resulting sample), whereas in general the coins used for sampling may be hard to compute from the corresponding outcome (which is the reason that an enhanced hardness assumption is used in the general analysis of the the above protocol). (See [68, Sec. 7.3.2] for a detailed proof of security.)

7.3.2 Compilation of passively-secure protocols into actively-secure ones

We show how to transform any passively-secure protocol into a corresponding actively-secure protocol. The communication model in both protocols consists of a single broadcast channel. Note that the messages of the original protocol may be assumed to be sent over a broadcast channel, because the adversary may see them anyhow (by tapping the point-to-point channels), and because a broadcast channel is trivially implementable in the case of passive adversaries. As for the resulting actively-secure protocol, the broadcast channel it uses can be implemented via an (authenticated) Byzantine Agreement protocol [54, 98], thus providing an emulation of this model on the standard point-to-point model (in which a broadcast channel does not exist). We mention that authenticated Byzantine Agreement is typically implemented using a signature scheme (and assuming that each party knows the verification-key corresponding to each of the other parties).

Turning to the transformation itself, the main idea is to use zero-knowledge proofs (as described in Section 4.3) in order to force parties to behave in a way that is consistent with the (passively-secure) protocol. Actually, we need to confine each party to a unique consistent behavior (i.e., according to some fixed local input and a sequence of coin tosses), and to guarantee that a party cannot fix its input (and/or its coins) in a way that depends on the inputs of honest parties. Thus, some preliminary steps have to be taken before the step-by-step emulation of the original protocol may start. Specifically, the compiled protocol (which like the original protocol is executed over a broadcast channel) proceeds as follows:

1. *Committing to the local input*: Prior to the emulation of the original protocol, each party commits to its input (using a commitment scheme [104]). In addition, using a zero-knowledge proof-of-knowledge [83, 20, 75], each party also proves that it knows its own input; that is, that it can decommit to the commitment it sent. (These zero-knowledge proof-of-knowledge are conducted sequentially to prevent dishonest parties from setting their inputs in a way that depends on inputs of honest parties; a more round-efficient method was presented in [46].)

2. *Generation of local random tapes*: Next, all parties jointly generate a sequence of random bits for each party such that only this party knows the outcome of the random sequence generated for it, but everybody gets a commitment to this outcome. These sequences will be used as the random-inputs (i.e., sequence of coin tosses) for the original protocol. Each bit in the random-sequence generated for Party X is determined as the exclusive-or of the outcomes of instances of an (augmented) coin-tossing protocol (cf. [28] and [68, Sec. 7.4.3.5]) that Party X plays with each of the other parties. The latter protocol provides the other parties with a commitment to the outcome obtained by Party X.

3. *Effective prevention of premature termination*: In addition, when compiling (the passively-secure protocol to an actively-secure protocol) *for the model that allows the adversary to control only a minority of the parties*, each party shares its input and random-input with all other parties using a "Verifiable Secret Sharing" (VSS) protocol (cf. [44] and [68, Sec. 7.5.5.1]). Loosely speaking, a VSS protocol allows to share a secret in a way that enables each participant to verify that the share it got fits the publicly posted information, which includes (on top of the commitments posted in Steps 1 and 2) commitments to all shares. The use of VSS guarantees that if Party X prematurely suspends the execution, then the honest parties can together reconstruct all Party X's secrets and carry on the execution while playing its role. This step effectively prevents premature termination, and is not needed in a model that does not consider premature termination a breach of security.

4. *Step-by-step emulation of the original protocol*: After all the above steps were completed, we turn to the main step in which the new protocol emulates the original one. In each step, each party augments the message determined by the original protocol with a zero-knowledge proof that asserts that the message was indeed computed correctly. Recall that the next message (as determined by the original protocol) is a function of the sender's own input, its random-input, and the messages it has received so far (where the latter are known to everybody because they were sent over a broadcast channel). Furthermore, the sender's input is determined by its commitment (as sent in Step 1), and its random-input is similarly determined (in Step 2). Thus, the next message (as determined by the original protocol) is a function of publicly known strings (i.e., the said commitments as well as the other messages sent over the broadcast channel). Moreover, the assertion that the next message was indeed computed correctly is an NP-assertion, and the sender knows a corresponding NP-witness (i.e., its own input and random-input as well as the corresponding decommitment information). Thus, the sender can prove in zero-knowledge (to each of the other parties) that the message it is sending was indeed computed according to the original protocol.

The above compilation was first outlined in [75, 76]. A detailed description and full proofs appear in [65, 68].

A secure coin-tossing protocol. Using a commitment scheme (see Section 4.3), we outline a secure (ordinary as opposed to augmented) coin-tossing protocol, which originates in [28].

Step C1: Party 1 uniformly selects $\sigma \in \{0,1\}$ and sends Party 2 a commitment, denoted c, to σ.

Step C2: Party 2 uniformly selects $\sigma' \in \{0,1\}$, and sends σ' to Party 1.

Step C3: Party 1 outputs the value $\sigma \oplus \sigma'$, and sends σ along with the decommitment information, denoted d, to Party 2.

Step C4: Party 2 checks whether or not (σ, d) fit the commitment c it has obtained in Step 1. It outputs $\sigma \oplus \sigma'$ if the check is satisfied and halts with output $\perp$ otherwise (indicating that Party 1 has essentially aborted the protocol prematurely).

Outputs: Party 1 always outputs $b \stackrel{\text{def}}{=} \sigma \oplus \sigma'$, whereas Party 2 either outputs b or $\perp$.

Intuitively, Steps C1–C2 may be viewed as "tossing a coin into the well". At this point (i.e., after Step C2) the value of the coin is determined (essentially as a random value), but only one party (i.e., Party 1) "can see" (i.e., knows) this value. Clearly, if both parties are honest then they both output the same uniformly chosen bit, recovered in Steps C3 and C4, respectively. Intuitively, each party can guarantee that the outcome is uniformly distributed, and Party 1 can cause premature termination by improper execution of Step 3. Formally, we have to show how the effect of every real-model adversary can be simulated by an adequate ideal-model adversary (which is allowed premature termination). This is done in [68, Sec. 7.4.3.1].

7.4 Concurrent execution of protocols

The definitions and results surveyed so far refer to a setting in which, at each time, only a single execution of a cryptographic protocol takes place (or only one execution may be controlled by the adversary). In contrast, in many distributed settings (e.g., the Internet), many executions are taking place concurrently (and several of them may be controlled by the same adversary). Furthermore, it is undesirable (and sometimes even impossible) to coordinate these executions (so to effectively enforce a single-execution setting). Still, the definitions and results obtained in the single-execution setting serve as a good starting point for the study of security in the setting of concurrent executions.

As in the case of stand-alone security, the notion of zero-knowledge provides a good test case for the study of concurrent security. Indeed, in order to demonstrate the security issues arising from concurrent execution of protocols, we consider the concurrent execution of zero-knowledge protocols. Specifically, we consider a party P holding a random (or rather pseudorandom) function $f:\{0,1\}^{2n} \to \{0,1\}^n$, and willing to participate in the following protocol (with respect to security parameter n).[28] The other party, called A for adversary, is supposed to send P a binary value $v \in \{1,2\}$ specifying which of the following cases to execute:

For $v = 1$: Party P uniformly selects $\alpha \in \{0,1\}^n$, and sends it to A, which is supposed to reply with a pair of n-bit long strings, denoted (β, γ). Party P checks whether or not $f(\alpha\beta) = \gamma$. In case equality holds, P sends A some secret information (e.g., the secret-key corresponding to P's public-key).

For $v = 2$: Party A is supposed to uniformly select $\alpha \in \{0,1\}^n$, and sends it to P, which selects uniformly $\beta \in \{0,1\}^n$, and replies with the pair $(\beta, f(\alpha\beta))$.

[28]In fact, assuming that P shares a pseudorandom function f with his friends (as explained in Section 3.3), the above protocol is an abstraction of a natural "mutual identification" protocol. (The example is adapted from [73].)

Observe that P's strategy (in each case) is zero-knowledge (even w.r.t auxiliary-inputs as defined in Definition 4.1): Intuitively, if the adversary A chooses the case $v = 1$, then it is infeasible for A to guess a passing pair (β, γ) with respect to a random α selected by P. Thus, except with negligible probability (when it may get secret information), A does not obtain anything from the interaction. On the other hand, if the adversary A chooses the case $v = 2$, then it obtains a pair that is indistinguishable from a uniformly selected pair of n-bit long strings (because β is selected uniformly by P, and for any α the value $f(\alpha\beta)$ looks random to A). In contrast, if the adversary A can conduct two concurrent executions with P, then it may learn the desired secret information: In one session, A sends $v = 1$ while in the other it sends $v = 2$. Upon receiving P's message, denoted α, in the first session, A sends it as its own message in the second session, obtaining a pair $(\beta, f(\alpha\beta))$ from P's execution of the second session. Now, A sends the pair $(\beta, f(\alpha\beta))$ to the first session of P, this pair passes the check, and so A obtains the desired secret.

An attack of the above type is called a *relay attack*: During such an attack the adversary just invokes two executions of the protocol and relays messages between them (without any modification). However, in general, the adversary in a concurrent setting is not restricted to relay attacks. For example, consider a minor modification to the above protocol so that, in case $v = 2$, party P replies with (say) the pair $(\beta, f(\overline{\alpha}\beta))$, where $\overline{\alpha} = \alpha \oplus 1^{|\alpha|}$, rather than with $(\beta, f(\alpha\beta))$. The modified strategy P is zero-knowledge and it also withstands a relay attack, but it can be "abused" easily by a more general concurrent attack.

The above example is merely the tip of an iceberg, but it suffices for introducing the main lesson: *an adversary attacking several concurrent executions of the same protocol may be able to cause more damage than by attacking a single execution* (or several sequential executions) of the same protocol. One may say that a protocol is concurrently secure if whatever the adversary may obtain by invoking and controlling parties in real concurrent executions of the protocol is also obtainable by a corresponding adversary that controls corresponding parties making concurrent functionality calls to a trusted party (in a corresponding ideal model).[29] More generally, one may consider concurrent executions of many sessions of *several* protocols, and say that a *set of protocols* is concurrently secure if whatever the adversary may obtain by invoking and controlling such real concurrent executions is also obtainable by a corresponding adversary that invokes and controls concurrent calls to a trusted party (in a corresponding ideal model). Consequently, a protocol is said to be secure with respect to concurrent compositions if adding this protocol to *any set* of concurrently secure protocols yields a set of concurrently secure protocols.

A much more appealing approach was recently suggested by Canetti [37]. Loosely speaking, Canetti suggests to consider a protocol to be secure (called *environmentally-secure* (or *Universally Composable secure* [37])) only if it remains secure when executed within any (feasible) environment. Following the simulation paradigm, we get the following definition:

Definition 7.2 (environmentally-secure protocols [37] – a rough sketch): *Let f be an m-ary functionality and Π be an m-party protocol, and consider the following real and ideal models.*

In the real model *the adversary controls some of the parties in an execution of Π and all parties can communicate with an arbitrary probabilistic polynomial-time process, which is called an* environment (*and possibly represents other executions of various protocols that are taking place*

[29] One specific concern (in such a concurrent setting) is the ability of the adversary to "non-trivially correlate the outputs" of concurrent executions. This ability, called *malleability*, was first investigated by Dolev, Dwork and Naor [52]. We comment that providing a general definition of what "correlated outputs" means seems very challenging (if at all possible). Indeed the focus of [52] is on several important special cases such as encryption and commitment schemes.

concurrently). Honest parties only communicate with the environment before the execution starts and when it ends; they merely obtain their inputs from the environment and pass their outputs to it. In contrast, dishonest parties may communicate freely with the environment, concurrently to the entire execution of Π.

In the ideal model *the* (simulating) *adversary controls the same parties, which use an ideal* (trusted-party) *that behaves according to the functionality* f *(as in Section 7.1.2). All parties can communicate with the* (same) *environment* (as in the real model)*. Indeed, the dishonest parties may communicate extensively with the environment before and after their single communication with the trusted party.*

We say that Π is an environmentally-secure protocol for computing f *if for every probabilistic polynomial-time adversary* A *in the real model there exists a probabilistic polynomial-time adversary* A' *controlling the same parties in the ideal model such that no probabilistic polynomial-time environment can distinguish the case in which it is accessed by the parties in the real execution from the case it is accessed by parties in the ideal model.*

As hinted above, the environment may account for other executions of various protocols that are taking place concurrently to the main execution being considered. The definition requires that such environments cannot distinguish the real execution from an ideal one. This means that anything that the real adversary (i.e., operating in the real model) gains from the execution and some environment, can be also obtained by an adversary operating in the ideal model and having access to the same environment. Indeed, Canetti proves that environmentally-secure protocols are secure with respect to concurrent compositions [37].

It is known is that environmentally-secure protocols for any functionality can be constructed for settings in which more than two-thirds of the active parties are honest [37]. This holds unconditionally for the private channel model, and under standard assumptions (e.g., allowing the construction of public-key encryption schemes) for the standard model (i.e., without private channel). The immediate consequence of this result is that general environmentally-secure multi-party computation is possible, provided that more than two-thirds of the parties are honest.

In contrast, general environmentally-secure *two-party* computation is not possible (in the standard sense).[30] Still, one can salvage general environmentally-secure two-party computation in the following reasonable model: Consider a network that contains servers that are willing to participate (as "helpers", possibly for a payment) in computations initiated by a set of (two or more) users. Now, suppose that two users wishing to conduct a secure computation can agree on a set of servers so that each user believes that more than two-thirds of the servers (in this set) are honest. Then, with the active participation of this set of servers, the two users can compute any functionality in an environmentally-secure manner.

Other reasonable models where general environmentally-secure *two-party* computation is possible include the common random-string (CRS) model [42] and variants of the public-key infrastructure (PKI) model [9]. In the CRS model, all parties have access to a universal random string (of length related to the security parameter). We stress that the entity trusted to post this universal random string is not required to take part in any execution of any protocol, and that all executions of all protocols may use the same universal random string. The PKI models considered in [9] require that each party deposits a public-key with a trusted center, while proving knowledge of a corresponding private-key. This proof may be conducted in zero-knowledge during special epochs in which no other activity takes place.

[30] Of course, some specific two-party computations do have environmentally-secure protocols. See [37] for several important examples (e.g., key exchange).

7.5 Concluding Remarks

In Sections 7.1-7.2 we have mentioned a host of definitions of security and constructions for multi-party protocols (especially for the case of more than two parties). Furthermore, some of these definitions are incomparable to others (i.e., they neither imply the others nor are implies by them), and there seems to be no single definition that may be crowned as the central one.

For example, in Sections 7.1.2 and 7.1.3, we have presented two alternative definitions of "secure multi-party protocols", one requiring an honest majority and the other allowing abort. These definitions are incomparable and there is no generic reason to prefer one over the other. Actually, as mentioned in Section 7.1.2, one could formulate a natural definition that implies both definitions (i.e., waiving the bound on the number of dishonest parties in Definition 7.1). Indeed, the resulting definition is free of the annoying restrictions that were introduced in each of the two aforementioned definitions; the "only" problem with the resulting definition is that it cannot be satisfied (in general). Thus, for the first time in this primer, we have reached a situation in which a natural (and general) definition cannot be satisfied, and we are forced to choose between two weaker alternatives, where each of these alternatives carries fundamental disadvantages.

In general, Section 7 carries a stronger flavor of compromise (i.e., recognizing inherent limitations and settling for a restricted meaningful goal) than previous sections. In contrast to the impression given in other parts of this primer, it is now obvious that we cannot get all that we may want (see Section 7.4). Instead, we should study the alternatives, and go for the one that best suits our real needs.

Indeed, as stated in Section 1.1, the fact that we can define a cryptographic goal does not mean that we can satisfy it as defined. In case we cannot satisfy the initial definition, we should search for relaxations that can be satisfied. These relaxations should be defined in a clear manner so that it would be obvious what they achieve (and what they fail to achieve). Doing so will allow a sound choice of the relaxation to be used in a specific application. This seems to be a good point to end the current primer.

A good compromise is one in which
the most important interests
of all parties are satisfied.

Adv. Klara Goldreich-Ingwer (1912–2004)

Acknowledgments

I wish to thank Minh-Huyen Nguyen for carefully reading this manuscript and pointing out various difficulties and errors. I also wish to thank Madhu Sudan and an anonymous referee for their comments.

References

[1] W. Aiello and J. Håstad. Perfect Zero-Knowledge Languages can be Recognized in Two Rounds. In *28th IEEE Symposium on Foundations of Computer Science*, pages 439–448, 1987.

[2] W. Alexi, B. Chor, O. Goldreich and C.P. Schnorr. RSA/Rabin Functions: Certain Parts are As Hard As the Whole. *SIAM Journal on Computing*, Vol. 17, April 1988, pages 194–209.

[3] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy. Proof Verification and Intractability of Approximation Problems. *Journal of the ACM*, Vol. 45, pages 501–555, 1998. Preliminary version in *33rd FOCS*, 1992.

[4] S. Arora and S. Safra. Probabilistic Checkable Proofs: A New Characterization of NP. *Journal of the ACM*, Vol. 45, pages 70–122, 1998. Preliminary version in *33rd FOCS*, 1992.

[5] L. Babai, L. Fortnow, L. Levin, and M. Szegedy. Checking Computations in Polylogarithmic Time. In *23rd ACM Symposium on the Theory of Computing*, pages 21–31, 1991.

[6] L. Babai, L. Fortnow, N. Nisan and A. Wigderson. BPP has Subexponential Time Simulations unless EXPTIME has Publishable Proofs. *Complexity Theory*, Vol. 3, pages 307–318, 1993.

[7] B. Barak. How to Go Beyond the Black-Box Simulation Barrier. In *42nd IEEE Symposium on Foundations of Computer Science*, pages 106–115, 2001.

[8] B. Barak. Constant-Round Coin-Tossing with a Man in the Middle or Realizing the Shared Random String Model. In *43th IEEE Symposium on Foundations of Computer Science*, pages 345–355, 2002.

[9] B. Barak, R. Canetti and J.B. Nielsen. Universally composable protocols with relaxed set-up assumptions. In *45th IEEE Symposium on Foundations of Computer Science*, pages 186–195, 2004.

[10] B. Barak and O. Goldreich, Universal arguments and their applications. In the *17th IEEE Conference on Computational Complexity*, pages 194–203, 2002.

[11] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (im)possibility of software obfuscation. In *Crypto01*, Springer-Verlag Lecture Notes in Computer Science (Vol. 2139), pages 1–18.

[12] B. Barak and Y. Lindell. Strict Polynomial-time in Simulation and Extraction. *SIAM Journal on Computing*, Vol. 33 (4), pages 783–818, May 2004.

[13] D. Beaver. Foundations of Secure Interactive Computing. In *Crypto91*, Springer-Verlag Lecture Notes in Computer Science (Vol. 576), pages 377–391.

[14] D. Beaver. Secure Multi-Party Protocols and Zero-Knowledge Proof Systems Tolerating a Faulty Minority. *Journal of Cryptology*, Vol. 4, pages 75–122, 1991.

[15] D. Beaver, S. Micali and P. Rogaway. The Round Complexity of Secure Protocols. In *22nd ACM Symposium on the Theory of Computing*, pages 503–513, 1990. See details in [118].

[16] M. Bellare. Electronic Commerce and Electronic Payments. Webpage of a course. `http://www-cse.ucsd.edu/users/mihir/cse291-00/`

[17] M. Bellare, R. Canetti and H. Krawczyk. Keying Hash Functions for Message Authentication. In *Crypto96*, Springer Lecture Notes in Computer Science (Vol. 1109), pages 1–15.

[18] M. Bellare, R. Canetti and H. Krawczyk. A Modular Approach to the Design and Analysis of Authentication and Key-Exchange Protocols. In *30th ACM Symposium on the Theory of Computing*, pages 419–428, 1998.

[19] M. Bellare, A. Desai, D. Pointcheval and P. Rogaway. Relations among notions of security for public-key encryption schemes. In *Crypto98*, Springer Lecture Notes in Computer Science (Vol. 1462), pages 26–45.

[20] M. Bellare and O. Goldreich. On Defining Proofs of Knowledge. In *Crypto92*, Springer-Verlag Lecture Notes in Computer Science (Vol. 740), pages 390–420.

[21] M. Bellare, R. Impagliazzo and M. Naor. Does Parallel Repetition Lower the Error in Computationally Sound Protocols? In *38th IEEE Symposium on Foundations of Computer Science*, pages 374–383, 1997.

[22] M. Bellare and P. Rogaway. Random Oracles are Practical: a Paradigm for Designing Efficient Protocols. In *1st Conf. on Computer and Communications Security*, ACM, pages 62–73, 1993.

[23] C.H. Bennett, G. Brassard and J.M. Robert. Privacy Amplification by Public Discussion. *SIAM Journal on Computing*, Vol. 17, pages 210–229, 1988. Preliminary version in *Crypto85*, titled "How to Reduce your Enemy's Information".

[24] M. Ben-Or, R. Canetti and O. Goldreich. Asynchronous Secure Computation. In *25th ACM Symposium on the Theory of Computing*, pages 52–61, 1993. See details in [35].

[25] M. Ben-Or, O. Goldreich, S. Goldwasser, J. Håstad, J. Kilian, S. Micali and P. Rogaway. Everything Provable is Probable in Zero-Knowledge. In *Crypto88*, Springer-Verlag Lecture Notes in Computer Science (Vol. 403), pages 37–56, 1990.

[26] M. Ben-Or, S. Goldwasser and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In *20th ACM Symposium on the Theory of Computing*, pages 1–10, 1988.

[27] M. Ben-Or, B. Kelmer and T. Rabin. Asynchronous Secure Computations with Optimal Resilience. In *13th ACM Symposium on Principles of Distributed Computing*, pages 183–192, 1994.

[28] M. Blum. Coin Flipping by Phone. *IEEE Spring COMPCOM*, pages 133–137, February 1982. See also *SIGACT News*, Vol. 15, No. 1, 1983.

[29] M. Blum, A. De Santis, S. Micali, and G. Persiano. Non-Interactive Zero-Knowledge Proof Systems. *SIAM Journal on Computing*, Vol. 20, No. 6, pages 1084–1118, 1991. (Considered the journal version of [30].)

[30] M. Blum, P. Feldman and S. Micali. Non-Interactive Zero-Knowledge and its Applications. In *20th ACM Symposium on the Theory of Computing*, pages 103–112, 1988. See [29].

[31] M. Blum and S. Goldwasser. An Efficient Probabilistic Public-Key Encryption Scheme which hides all partial information. In *Crypto84*, Lecture Notes in Computer Science (Vol. 196) Springer-Verlag, pages 289–302.

[32] M. Blum and S. Micali. How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits. *SIAM Journal on Computing*, Vol. 13, pages 850–864, 1984. Preliminary version in *23rd FOCS*, 1982.

[33] G. Brassard, D. Chaum and C. Crépeau. Minimum Disclosure Proofs of Knowledge. *Journal of Computer and System Science*, Vol. 37, No. 2, pages 156–189, 1988. Preliminary version by Brassard and Crépeau in *27th FOCS*, 1986.

[34] G. Brassard and C. Crépeau. Zero-Knowledge Simulation of Boolean Circuits. In *Crypto86*, Springer-Verlag Lecture Notes in Computer Science (Vol. 263), pages 223–233, 1987.

[35] R. Canetti. *Studies in Secure Multi-Party Computation and Applications*. Ph.D. Thesis, Department of Computer Science, Weizmann Institute of Science, Rehovot, Israel, June 1995. Available from `http://www.wisdom.weizmann.ac.il/~oded/PS/ran-phd.ps`.

[36] R. Canetti. Security and Composition of Multi-party Cryptographic Protocols. *Journal of Cryptology*, Vol. 13, No. 1, pages 143–202, 2000.

[37] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145, 2001. Full version (with different title) is available from *Cryptology ePrint Archive*, Report 2000/067.

[38] R. Canetti, U. Feige, O. Goldreich and M. Naor. Adaptively Secure Multi-party Computation. In *28th ACM Symposium on the Theory of Computing*, pages 639–648, 1996.

[39] R. Canetti, O. Goldreich and S. Halevi. The Random Oracle Methodology, Revisited. In *30th ACM Symposium on the Theory of Computing*, pages 209–218, 1998.

[40] R. Canetti and A. Herzberg. Maintaining Security in the Presence of Transient Faults. In *Crypto94*, Springer-Verlag Lecture Notes in Computer Science (Vol. 839), pages 425–439.

[41] R. Canetti, J. Kilian, E. Petrank and A. Rosen. Black-Box Concurrent Zero-Knowledge Requires $\tilde{\Omega}(\log n)$ Rounds. In *33rd ACM Symposium on the Theory of Computing*, pages 570–579, 2001.

[42] R. Canetti, Y. Lindell, R. Ostrovsky and A. Sahai. Universally Composable Two-Party and Multi-Party Secure Computation. In *34th ACM Symposium on the Theory of Computing*, pages 494–503, 2002.

[43] D. Chaum, C. Crépeau and I. Damgård. Multi-party unconditionally Secure Protocols. In *20th ACM Symposium on the Theory of Computing*, pages 11–19, 1988.

[44] B. Chor, S. Goldwasser, S. Micali and B. Awerbuch. Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults. In *26th IEEE Symposium on Foundations of Computer Science*, pages 383–395, 1985.

[45] B. Chor and E. Kushilevitz. A Zero-One Law for Boolean Privacy. *SIAM J. on Disc. Math.*, Vol. 4, pages 36–47, 1991.

[46] B. Chor and M.O. Rabin. Achieving independence in logarithmic number of rounds. In *6th ACM Symposium on Principles of Distributed Computing*, pages 260–268, 1987.

[47] R. Cleve. Limits on the Security of Coin Flips when Half the Processors are Faulty. In *18th ACM Symposium on the Theory of Computing*, pages 364–369, 1986.

[48] I. Damgård. Collision Free Hash Functions and Public Key Signature Schemes. In *EuroCrypt87*, Springer-Verlag, Lecture Notes in Computer Science (Vol. 304), pages 203–216.

[49] I. Damgard and J. B. Nielsen. Improved non-committing encryption schemes based on general complexity assumption. In *Crypto00*, Springer-Verlag Lecture Notes in Computer Science (Vol. 1880), pages 432–450.

[50] A. De Santis, G. Di Crescenzo, R. Ostrovsky, G. Persiano, A. Sahai. Robust Non-interactive Zero-Knowledge. In *Crypto01*, Springer Lecture Notes in Computer Science (Vol. 2139), pages 566–598.

[51] W. Diffie, and M.E. Hellman. New Directions in Cryptography. *IEEE Trans. on Info. Theory*, IT-22 (Nov. 1976), pages 644–654.

[52] D. Dolev, C. Dwork, and M. Naor. Non-Malleable Cryptography. *SIAM Journal on Computing*, Vol. 30, No. 2, pages 391–437, 2000. Preliminary version in *23rd STOC*, 1991.

[53] D. Dolev, C. Dwork, O. Waarts, and M. Yung. Perfectly secure message transmission. *Journal of the ACM*, Vol. 40 (1), pages 17–47, 1993.

[54] D. Dolev and H.R. Strong. Authenticated Algorithms for Byzantine Agreement. *SIAM Journal on Computing*, Vol. 12, pages 656–666, 1983.

[55] C. Dwork, M. Naor, and A. Sahai. Concurrent Zero-Knowledge. In *30th ACM Symposium on the Theory of Computing*, pages 409–418, 1998.

[56] S. Even, O. Goldreich, and A. Lempel. A Randomized Protocol for Signing Contracts. *Communications of the ACM*, Vol. 28, No. 6, 1985, pages 637–647.

[57] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Approximating Clique is almost NP-complete. *Journal of the ACM*, Vol. 43, pages 268–292, 1996. Preliminary version in *32nd FOCS*, 1991.

[58] U. Feige, D. Lapidot, and A. Shamir. Multiple Non-Interactive Zero-Knowledge Proofs Under General Assumptions. *SIAM Journal on Computing*, Vol. 29 (1), pages 1–28, 1999.

[59] U. Feige and A. Shamir. Witness Indistinguishability and Witness Hiding Protocols. In *22nd ACM Symposium on the Theory of Computing*, pages 416–426, 1990.

[60] A. Fiat and A. Shamir. How to Prove Yourself: Practical Solution to Identification and Signature Problems. In *Crypto86*, Springer-Verlag Lecture Notes in Computer Science (Vol. 263), pages 186–189, 1987.

[61] L. Fortnow, The Complexity of Perfect Zero-Knowledge. In *19th ACM Symposium on the Theory of Computing*, pages 204–209, 1987.

[62] P.S. Gemmell. An Introduction to Threshold Cryptography. In *CryptoBytes*, RSA Lab., Vol. 2, No. 3, 1997.

[63] R. Gennaro, M. Rabin and T. Rabin. Simplified VSS and Fast-track Multiparty Computations with Applications to Threshold Cryptography. In *17th ACM Symposium on Principles of Distributed Computing*, pages 101–112, 1998.

[64] O. Goldreich. A Uniform Complexity Treatment of Encryption and Zero-Knowledge. *Journal of Cryptology*, Vol. 6, No. 1, pages 21–53, 1993.

[65] O. Goldreich. *Secure Multi-Party Computation.* Working draft, June 1998. Available from `http://www.wisdom.weizmann.ac.il/~oded/pp.html`.

[66] O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness.* Algorithms and Combinatorics series (Vol. 17), *Springer*, 1998.

[67] O. Goldreich. *Foundations of Cryptography – Basic Tools.* Cambridge University Press, 2001.

[68] O. Goldreich. *Foundations of Cryptography – Basic Applications.* Cambridge University Press, 2004.

[69] O. Goldreich. Concurrent Zero-Knowledge With Timing, Revisited. In *34th ACM Symposium on the Theory of Computing*, pages 332–340, 2002.

[70] O. Goldreich, S. Goldwasser, and S. Micali. How to Construct Random Functions. *Journal of the ACM*, Vol. 33, No. 4, pages 792–807, 1986.

[71] O. Goldreich and J. Håstad. On the Complexity of Interactive Proofs with Bounded Communication. *IPL*, Vol. 67 (4), pages 205–214, 1998.

[72] O. Goldreich and A. Kahan. How to Construct Constant-Round Zero-Knowledge Proof Systems for NP. *Journal of Cryptology*, Vol. 9, No. 2, pages 167–189, 1996.

[73] O. Goldreich and H. Krawczyk. On the Composition of Zero-Knowledge Proof Systems. *SIAM Journal on Computing*, Vol. 25, No. 1, February 1996, pages 169–192.

[74] O. Goldreich and L.A. Levin. Hard-core Predicates for any One-Way Function. In *21st ACM Symposium on the Theory of Computing*, pages 25–32, 1989.

[75] O. Goldreich, S. Micali and A. Wigderson. Proofs that Yield Nothing but their Validity or All Languages in NP Have Zero-Knowledge Proof Systems. *Journal of the ACM*, Vol. 38, No. 1, pages 691–729, 1991. Preliminary version in *27th FOCS*, 1986.

[76] O. Goldreich, S. Micali and A. Wigderson. How to Play any Mental Game – A Completeness Theorem for Protocols with Honest Majority. In *19th ACM Symposium on the Theory of Computing,* pages 218–229, 1987. See details in [65].

[77] O. Goldreich and Y. Oren. Definitions and Properties of Zero-Knowledge Proof Systems. *Journal of Cryptology*, Vol. 7, No. 1, pages 1–32, 1994.

[78] O. Goldreich, A. Sahai, and S. Vadhan. Honest-Verifier Statistical Zero-Knowledge equals general Statistical Zero-Knowledge. In *30th ACM Symposium on the Theory of Computing*, pages 399–408, 1998.

[79] O. Goldreich, S. Vadhan and A. Wigderson. On interactive proofs with a laconic provers. *Computational Complexity*, Vol. 11, pages 1–53, 2002.

[80] O. Goldreich and R. Vainish. How to Solve any Protocol Problem – An Efficiency Improvement. In *Crypto87*, Springer Verlag, Lecture Notes in Computer Science (Vol. 293), pages 73–86.

[81] S. Goldwasser and L.A. Levin. Fair Computation of General Functions in Presence of Immoral Majority. In *Crypto90*, Springer-Verlag Lecture Notes in Computer Science (Vol. 537), pages 77–93.

[82] S. Goldwasser and S. Micali. Probabilistic Encryption. *Journal of Computer and System Science*, Vol. 28, No. 2, pages 270–299, 1984. Preliminary version in *14th STOC*, 1982.

[83] S. Goldwasser, S. Micali and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on Computing*, Vol. 18, pages 186–208, 1989. Preliminary version in *17th STOC*, 1985.

[84] S. Goldwasser, S. Micali, and R.L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM Journal on Computing*, April 1988, pages 281–308.

[85] S.W. Golomb. *Shift Register Sequences*. Holden-Day, 1967. (Aegean Park Press, revised edition, 1982.)

[86] R. Greenstadt. Electronic Voting Bibliography, 2000. `http://theory.lcs.mit.edu/~cis/voting/greenstadt-voting-bibliography.html`.

[87] J. Håstad, R. Impagliazzo, L.A. Levin and M. Luby. A Pseudorandom Generator from any One-way Function. *SIAM Journal on Computing*, Volume 28, Number 4, pages 1364–1396, 1999.

[88] M. Hirt and U. Maurer. Complete characterization of adversaries tolerable in secure multi-party computation. *Journal of Cryptology*, Vol. 13, No. 1, pages 31–60, 2000.

[89] R. Impagliazzo, L.A. Levin and M. Luby. Pseudorandom Generation from One-Way Functions. In *21st ACM Symposium on the Theory of Computing*, pages 12–24, 1989.

[90] R. Impagliazzo and M. Luby. One-Way Functions are Essential for Complexity Based Cryptography. In *30th IEEE Symposium on Foundations of Computer Science*, pages 230–235, 1989.

[91] R. Impagliazzo and M. Yung. Direct Zero-Knowledge Computations. In *Crypto87*, Springer-Verlag Lecture Notes in Computer Science (Vol. 293), pages 40–51, 1987.

[92] J. Katz and M. Yung. Complete Characterization of Security Notions for Probabilistic Private-Key Encryption. In *32nd ACM Symposium on the Theory of Computing*, pages 245–254, 2000.

[93] J. Kilian. A Note on Efficient Zero-Knowledge Proofs and Arguments. In *24th ACM Symposium on the Theory of Computing*, pages 723–732, 1992.

[94] J. Kilian and E. Petrank Concurrent and Resettable Zero-Knowledge in Poly-logarithmic Rounds In *33rd ACM Symposium on the Theory of Computing*, pages 560–569, 2001.

[95] D.E. Knuth. *The Art of Computer Programming*, Vol. 2 (*Seminumerical Algorithms*). Addison-Wesley Publishing Company, Inc., 1969 (first edition) and 1981 (second edition).

[96] H. Krawczyk. LFSR-based Hashing and Authentication. In *Crypto94*, Lecture Notes in Computer Science (Vol. 839), Springer-Verlag, pages 129–139.

[97] Y. Lindell. Parallel Coin-Tossing and Constant-Round Secure Two-Party Computation. In *Crypto01*, Springer Lecture Notes in Computer Science (Vol. 2139), pages 171–189, 2001.

[98] Y. Lindell, A. Lysyanskaya and T. Rabin. On the Composition of Authenticated Byzantine Agreement. In *34th ACM Symposium on the Theory of Computing*, pages 514–523, 2002.

[99] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic Methods for Interactive Proof Systems. *Journal of the ACM*, Vol. 39, No. 4, pages 859–868, 1992. Preliminary version in *31st FOCS*, 1990.

[100] A.J. Menezes, P.C. van Oorschot and S.A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[101] R.C. Merkle. Protocols for public key cryptosystems. In *Proc. of the 1980 Symposium on Security and Privacy.*

[102] S. Micali. Computationally Sound Proofs. *SIAM Journal on Computing*, Vol. 30 (4), pages 1253–1298, 2000. Preliminary version in *35th FOCS*, 1994.

[103] S. Micali and P. Rogaway. Secure Computation. In *Crypto91*, Springer-Verlag Lecture Notes in Computer Science (Vol. 576), pages 392–404. Ellaborated working draft available from the authors.

[104] M. Naor. Bit Commitment using Pseudorandom Generators. *Journal of Cryptology*, Vol. 4, pages 151–158, 1991.

[105] M. Naor and K. Nissim. Communication preserving protocols for secure function evaluation. In *33rd ACM Symposium on the Theory of Computing*, 2001, pages 590–599.

[106] M. Naor and M. Yung. Universal One-Way Hash Functions and their Cryptographic Application. In *21st ACM Symposium on the Theory of Computing*, 1989, pages 33–43.

[107] M. Naor and M. Yung. Public-Key Cryptosystems Provably Secure Against Chosen Ciphertext Attacks. In *22nd ACM Symposium on the Theory of Computing*, pages 427-437, 1990.

[108] National Institute for Standards and Technology. *Digital Signature Standard* (DSS). *Federal Register*, Vol. 56, No. 169, August 1991.

[109] R. Ostrovsky and A. Wigderson. One-Way Functions are essential for Non-Trivial Zero-Knowledge. In *2nd Israel Symp. on Theory of Computing and Systems*, IEEE Comp. Soc. Press, pages 3–17, 1993.

[110] R. Ostrovsky and M. Yung. How to Withstand Mobile Virus Attacks. In *10th ACM Symposium on Principles of Distributed Computing*, pages 51–59, 1991.

[111] M. Prabhakaran, A. Rosen and A. Sahai. Concurrent Zero-Knowledge Proofs in Logarithmic Number of Rounds. In *43rd IEEE Symposium on Foundations of Computer Science*, pages 366–375, 2002.

[112] M.O. Rabin. Digitalized Signatures. In *Foundations of Secure Computation* (R.A. DeMillo et. al. eds.), Academic Press, 1977.

[113] M.O. Rabin. Digitalized Signatures and Public Key Functions as Intractable as Factoring. MIT/LCS/TR-212, 1979.

[114] M.O. Rabin. How to Exchange Secrets by Oblivious Transfer. Tech. Memo TR-81, Aiken Computation Laboratory, Harvard U., 1981.

[115] T. Rabin and M. Ben-Or. Verifiable Secret Sharing and Multi-party Protocols with Honest Majority. In *21st ACM Symposium on the Theory of Computing*, pages 73–85, 1989.

[116] R. Richardson and J. Kilian. On the Concurrent Composition of Zero-Knowledge Proofs. In *EuroCrypt99*, Springer Lecture Notes in Computer Science (Vol. 1592), pages 415–413.

[117] R. Rivest, A. Shamir and L. Adleman. A Method for Obtaining Digital Signatures and Public Key Cryptosystems. *Communications of the ACM*, Vol. 21, Feb. 1978, pages 120–126

[118] P. Rogaway. The Round Complexity of Secure Protocols. MIT Ph.D. Thesis, June 1991. Available from `http://www.cs.ucdavis.edu/~rogaway/papers`.

[119] J. Rompel. One-way Functions are Necessary and Sufficient for Secure Signatures. In *22nd ACM Symposium on the Theory of Computing*, 1990, pages 387–394.

[120] A. Sahai. Improved Constructions Achieving Chosen-Ciphertext Security. In preparation, 2001. See [50].

[121] A. Sahai and S. Vadhan. A Complete Promise Problem for Statistical Zero-Knowledge. *Journal of the ACM*, Vol. 50, No. 2, pages 1–54, April 2003.

[122] A. Shamir. How to Share a Secret. *Communications of the ACM*, Vol. 22, Nov. 1979, pages 612–613.

[123] A. Shamir. IP = PSPACE. *Journal of the ACM*, Vol. 39, No. 4, pages 869–877, 1992. Preliminary version in *31st FOCS*, 1990.

[124] C.E. Shannon. Communication Theory of Secrecy Systems. *Bell Sys. Tech. J.*, Vol. 28, pages 656–715, 1949.

[125] M. Sipser. A Complexity Theoretic Approach to Randomness. In *15th ACM Symposium on the Theory of Computing*, pages 330–335, 1983.

[126] S. Vadhan. A Study of Statistical Zero-Knowledge Proofs. PhD Thesis, Department of Mathematics, MIT, 1999. Available from `http://www.eecs.harvard.edu/~salil/papers/phdthesis-abs.html`.

[127] S. Vadhan. An Unconditional Study of Computational Zero Knowledge. In *45th IEEE Symposium on Foundations of Computer Science*, pages 176–185, 2004.

[128] A.C. Yao. Theory and Application of Trapdoor Functions. In *23rd IEEE Symposium on Foundations of Computer Science*, pages 80–91, 1982.

[129] A.C. Yao. How to Generate and Exchange Secrets. In *27th IEEE Symposium on Foundations of Computer Science*, pages 162–167, 1986.