# Cryptography and Cryptographic Protocols

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

June 9, 2001

**Abstract**

We survey the paradigms, approaches and techniques used to conceptualize, define and provide solutions to natural cryptographic problems. We start by presenting some of the central tools (e.g., computational difficulty, pseudorandomness, and zero-knowledge proofs), and next turn to the treatment of encryption and signature schemes. We conclude with an extensive treatment of secure cryptographic protocols both as stand-alone and when many sessions of various protocols are concurrently executed and controlled by an adversary.

The survey is intended for researchers in distributed computing, and assumes no prior familiarity with cryptography.

# Contents


**1 Introduction** **1**
- 1.1 Cryptography . . . 1
- 1.2 Cryptographic Protocols . . . 2

**2 Preliminaries** **4**

**3 Basic Tools** **4**
- 3.1 Computational Difficulty and One-Way Functions . . . 4
- 3.2 Pseudorandomness . . . 6
  - 3.2.1 Computational Indistinguishability . . . 6
  - 3.2.2 Pseudorandom Generators . . . 7
  - 3.2.3 Pseudorandom Functions . . . 8
- 3.3 Zero-Knowledge . . . 8
  - 3.3.1 The Simulation Paradigm . . . 8
  - 3.3.2 The Actual Definition . . . 9
  - 3.3.3 Zero-Knowledge Proofs for all NP-assertions and their applications . . . 10

**4 Encryption and Signature Schemes** **11**
- 4.1 Encryption Schemes . . . 11
  - 4.1.1 Definitions . . . 13
  - 4.1.2 Constructions . . . 14
  - 4.1.3 Beyond eavesdropping security . . . 16
- 4.2 Signature and Message Authentication Schemes . . . 16
  - 4.2.1 Definitions . . . 18
  - 4.2.2 Constructions . . . 18

**5 Cryptographic Protocols (as stand-alone)** **19**
- 5.1 The Definitional Approach and Some Models . . . 19
  - 5.1.1 Some parameters used in defining security models . . . 20
  - 5.1.2 Example: Multi-party protocols with honest majority . . . 21
- 5.2 Some Known Results . . . 23
- 5.3 Construction Paradigms . . . 24
  - 5.3.1 Compilation of passively-secure protocols into actively-secure ones . . . 25
  - 5.3.2 Passively-secure computation with "encrypted circuits" . . . 26
  - 5.3.3 Passively-secure computation with shares . . . 27

**6 Cryptographic Protocols (under concurrent execution)** **28**
- 6.1 Some issues and some definitional approaches . . . 28
- 6.2 Some difficulties . . . 31
- 6.3 Some currently-known positive results . . . 31

**References** **32**

# 1 Introduction

The modern society is quite preoccupied with various statistics like the average, median and deviation of various attributes (e.g., salary) of it members. On the other hand, individuals often wish to keep their own attributes secret (although they are interested in the above statistics). Furthermore, on top of being suspicious of other people, individuals are growing to be suspicious of all (the society's) establishments and are unwilling to trust the latter with their secrets. Under these circumstances it is not clear whether there is a way for the members of the society to obtain various statistics (regarding all secrets) without revealing their individual secrets to other people.

The above question is a special case of a general problem. We are talking about computing some (predetermined) function of inputs that are scattered among different parties, without having these parties reveal their individual inputs. The mutually suspectful parties have to employ some distributed protocol in order to compute the function value, without revealing their inputs to other parties. Furthermore, in some settings, some of the parties may deviate from the protocol, and it is desired that such malfunctioning will not be of any advantage to them. At best, we would like to "emulate" a trusted party, and hope to do so in a distributed setting in which no trusted parties exist. This, in a nutshell, is what secure cryptographic protocols are all about.

The results presented in this survey describe a variety of reasonable models in which such an "emulation" is possible. In other words, we survey the main (general) results regarding secure cryptographic protocols. But before doing so, we shall discuss more specific cryptographic primitives and tools. This is done for three main reasons: Most importantly, we believe that these primitives and tools should be of interest to researchers in the area of distributed computing. Secondly, the area of secure cryptographic protocols is very complex, and we believe that some familiarity with the rest of cryptography is very helpful when studying cryptographic protocols. Lastly, many of the known results regarding secure cryptographic protocols rely on primitives and tools developed in the rest of cryptography.

## 1.1 Cryptography

Modern cryptography is concerned with the construction of systems that are robust against malicious attempts to make these systems deviate from their prescribed functionality. Indeed, the scope of modern cryptography is very broad, and stands in contrast to "classical" cryptography, which has been associated with the single problem of providing secret communication over insecure communication media.

The design of cryptographic schemes 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 above, 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, it is our opinion that 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. Consequently, the design of good cryptographic systems consists of two main steps:

1. A Definitional Step: The identification, conceptualization and rigorous definition of a cryptographic task that captures the intuitive security concern at hand; and

2. A Constructive Step: The design of cryptographic schemes satisfying the definition distilled in Step (1), possibly while relying on widely believed and well-understood intractability assumptions.

We note that most of modern cryptography relies on intractability assumptions, and that relying on such assumptions is unavoidable (in the sense discussed in Section 3.1). Still there is a huge difference between *relying on a simple, explicitly stated assumption* and just assuming (or rather hoping) that an ad-hoc construction satisfies some vaguely specified (or even unspecified) goals.

In Sections 3 and 4 we survey the foundations of cryptography. We shall highlight paradigms, approaches and techniques used to conceptualize, define and provide solutions to natural "security concerns". Specifically, we start our presentation with basic paradigms and tools such as computational difficulty (Section 3.1), pseudorandomness (Section 3.2) and zero-knowledge (Section 3.3). Once these are presented, we turn to encryption (Section 4.1) and signature schemes (Section 4.2). At that point, we will be ready to discuss cryptographic protocols.

## 1.2 Cryptographic Protocols

Cryptography, in the broad sense defined above, encompasses also the area of Cryptographic Protocols. Taking the opposite perspective, all of cryptography deals with cryptographic protocols (because any scheme or algorithm can be viewed as a (possibly) degenerated protocol). Still, we believe that it make sense to differentiate between basic primitives (which involve little interaction) like encryption and signature schemes on one hand, and general cryptographic protocols on the other hand.

In a nutshell, general results concerning secure (two-party and) multi-party computations 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.

In Section 5, we survey the above results. We stress that these results presuppose that, during the execution of the (secure) protocol, the parties that participate in the execution do not take part in any other protocol execution. That is, it is not guaranteed that the above mentioned protocols maintain their security when executed concurrently to other protocols (or even to other instances of the same protocol): conceivably, an adversary that controls parties in several concurrent executions, may gain some illegitimate advantage. Thus, it is desirable (and in some settings imperative) to design protocols that maintain their security also when executed concurrently to other protocols (or to other instances of themselves). In Section 6, we survey the known results regarding security under concurrent executions. At this point, we wish to make several comments:

- The issue of security under concurrent execution arises only if the adversary may initiate and *control* several concurrent executions. In contrast, concurrent executions that are not controlled by the same adversary (or set of coordinating adversaries) do not introduce any new security issue (beyond stand-alone security).

- Preservation of security under concurrent executions seems essential in settings such as the Internet, in which many (distributed) processes do take place concurrently and it is unreasonable to require these processes to coordinate their actions. We stress that although inter-process coordination cannot be required of the legitimate programs, it cannot be assumed that the adversary does not coordinate its attacks on the various processes. (Coordination is possible, but too expensive to require in normal operation. Still the adversary may be willing to invest the effort if, by coordinating its attack in the various processes, it can obtain substantial gain.)

- It is hasty to conclude that "stand-alone security" (as surveyed in Section 5) is worthless (i.e., unsatisfactory in all reasonable settings). We believe that "stand-alone security" may be sufficient in some (small) distributed systems. On one extreme, in distributed systems in which executions of secure multi-party computations are rare, these executions can be coordinated so that they do not take place concurrently. On the other extreme, in distributed systems in which executions of secure multi-party computations involving all (or most) parties take place all the time, it may be possible to "lump together" all these computations into a single multi-party computations (e.g., by implementing a secure distributed operating system that supports all these desired multi-party computations). As another (related) example, consider a (small) distributed system that operates under a single distributed operating system. The desired functionality of such an operating system can be casted as a (reactive) multi-party functionality, and as such one can design a secure implementation of it. This means that we obtain a secure distributed operating system that maintains its functionality even if some of the processors behave in a malicious way (e.g., are governed by an adversary).[1]

- The current state of knowledge regarding preservation of security under concurrent executions still lags behind what is known regarding the security of protocols as stand-alone. In particular, satisfactory positive results are currently known with respect to general multi-party protocols (in which more than 2/3 of the parties are honest), but not regarding general two-party protocols (being executed in parallel to other instances of themselves).

**Historical comment:** The first general results regarding secure multi-party computations [45, 76, 46] were presented at a time in which *intensive* electronic multi-party interactions seemed a remote possibility. This may explain the impression that, while generating considerable interest within the Theory of Computation community, these results went almost unnoticed by the Applied Cryptography community. But times have changed: intensive electronic multi-party interactions seems almost a reality, and the entire cryptographic community seems very much interested in a variety of natural problems that arise from such a reality.

**The communication model:** The choice of the communication model is often neglected in the cryptographic research. Most of cryptographic research is concerned with two-party computations, in which case the choice of model is not very important, and typically an (asynchronous) message-passing model is assumed (almost always implicitly). For multi-party cryptographic protocols, the choice of the communication model is more important. With the exception of a few work (most notably [10]), the model of choice for multi-party cryptographic protocols is a synchronous model consisting of either point-to-point channels or a single "broadcast channel" (or both [68]). With

[1] We comment that in a secure distributed operating system as suggested above, all (or most) parties will have to actively participate in each action taken by the system. Actually, if one assumes that at most $t$ parties may be controlled by the adversary then it suffices to have $O(t)$ parties participate in each action taken by the system.

the exception of a few work (most notably [33]), in case point-to-point channels are assumed, they are assumed to exist between every pair of processors. Indeed, implementing such results in an arbitrary communication network will require secure and reliable routing, which may be achieved (in principle) using encryption and message authentication (or signature) schemes.

# 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 (via an adversary) ought to be infeasible.

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). Here (i.e., when referring to the complexity of the legitimate user) 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. A common approach is to postulate that the latter are polynomial-time too, where the polynomial is not a-priori specified. In other words, the adversary is restricted to the class of efficient computations and anything beyond this is considered to be infeasible. Although many definitions explicitly refer to this convention, 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 adversaries of running-time bounded by any super-polynomial function (or class of functions). Still, for sake of concreteness and clarity, we shall use the former convention in our formal definitions.

Lastly we consider the notion of a negligible probability. The idea behind this notion is to have 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, if we consider any polynomial-time computation to be feasible then any function $\mu : \mathbb{N} \to \mathbb{N}$ so that $(1 - \mu(n))^{p(n)} > 0.99$, for any polynomial $p$, is considered negligible (i.e., $\mu$ is negligible if for any polynomial $p'$ the function $\mu(\cdot)$ is bounded above 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 $n$).

# 3 Basic Tools

In this section we survey three basic tools used in Modern Cryptography. The most basic tool is computational difficulty, which in turn is captured in 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 section, see our recent textbook [40].

## 3.1 Computational Difficulty and One-Way Functions

Modern Cryptography is concerned with the construction of schemes which are easy to operate (properly) but hard to foil. Thus, a complexity gap (i.e., between the complexity of proper usage and the complexity of defeating the prescribed functionality) lies in 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. One-way functions are functions that are easy to evaluate but hard (on the average) to invert. That is, a function $f:\{0,1\}^* \rightarrow \{0,1\}^*$ 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 3.1** (one-way functions): *A function $f:\{0,1\}^* \rightarrow \{0,1\}^*$ 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.)

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 composites is infeasible if and only if squaring modulo such composite is a one-way function (see [66]). 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 [69]: $x \mapsto x^e \bmod N$, 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 3.1):

**Definition 3.2** (collections of one-way functions and additional properties): *A collection of functions, $\{f_i : D_i \rightarrow \{0,1\}^*\}_{i \in \overline{I}}$, 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: *The output of algorithm $I$, on input $1^n$, is distributed over the set $\overline{I} \cap \{0,1\}^n$ (i.e., is an $n$-bit long index of some function). The output of algorithm $D$, on input (an index of a function) $i \in \overline{I}$, is distributed over the set $D_i$ (i.e., over the domain of the function). On input $i \in \overline{I}$ and $x \in D_i$, algorithm $F$* always *outputs $f_i(x)$.*

2. hard to invert; *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 of permutations if each of the $f_i$'s is a permutation over the corresponding $D_i$. Such a collection is call a* trapdoor permutation *if in addition to the above there are two 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 $I'(1^n)$ it holds that $F^{-1}(t, f_i(x)) = x$.* (That is, $t$ is a trapdoor that allows to invert $f_i$.)

Note that the hardness-to-invert 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.)

## 3.2 Pseudorandomness

In practice "pseudorandom" sequences are used instead of truly random sequences in many applications. 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 [56] or having large linear-complexity (as in [53]). In contrast, the above belief is an easy corollary of defining pseudorandom distributions as ones that are computationally indistinguishable from uniform distributions.

### 3.2.1 Computational Indistinguishability

A central notion in Modern Cryptography is that of "effective similarity" (introduced by Goldwasser, Micali and Yao [50, 75]). The underlying idea is that we do not care if objects are equal or not, all we care is whether 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, it will be our common practice below to 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 $n$. 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 3.3** (computational indistinguishability [50, 75]): *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, think of $D$ as of somebody who wishes to distinguish two distributions (based on a sample given to it), and think of 1 as of $D$'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 an efficient 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).

We comment that, for "efficiently constructible" distributions, indistinguishability by a single sample (as defined above) implies indistinguishability by multiple samples (see [40, Sec. 3.2.3]).

### 3.2.2 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 [17, 75], and are formally defined as follows.

**Definition 3.4** (pseudorandom generator [17, 75]): *Let $\ell : \mathbb{N} \to \mathbb{N}$ be so that $\ell(n) > n$, for all $n \in \mathbb{N}$. A* pseudorandom generator, *with* stretch function *$\ell$, 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 $\{0,1\}^m$.*

Thus, pseudorandom sequences can replace truly random sequences not only in "ordinary" computations 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. We start with the notion of a *hard-core predicate* of a (one-way) function: The predicate $b$ is a hard-core of the function $f$ if $b$ is easy to evaluate but $b(x)$ is hard to predict from $f(x)$. That is, it is infeasible, given $f(x)$ when $x$ is uniformly chosen, to predict $b(x)$ substantially better than with probability 1/2. We mention that by [44] (see proof in [40, Sec. 2.5.2]), for any one-way function $f$, the inner-product mod 2 of $x$ and $r$ is a hard-core of $f'(x,r) = (f(x), r)$.

**Theorem 3.5** ([17, 75], see [40, 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 $\ell$), 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 $x \mapsto x^2 \bmod N$, 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 $G_N(s) = \text{lsb}(s)\cdot\text{lsb}(s^2 \bmod N)\cdots\text{lsb}(s^{2^{\ell(|s|)-1}} \bmod N)$, where $\text{lsb}(x)$ is the least significant bit of $x$ (which is a hard-core of the modular squaring function [1]).

We conclude this subsection by mentioning that pseudorandom generators can be constructed from *any* one-way functions (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 3.6** [54]: *Pseudorandom generators exist if and only* (arbitrary) *one-way functions exist.*

### 3.2.3 Pseudorandom Functions

Pseudorandom generators allow to efficiently generate long pseudorandom sequences from short random seeds. Pseudorandom functions, introduced and constructed by Goldreich, Goldwasser and Micali [42], are even more powerful: they allow 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 which 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 responses of $f_s$, for a uniformly chosen $s \in \{0,1\}^n$, from the responses of a truly random function $F : \{0,1\}^n \to \{0,1\}^n$.

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 $v$ 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 3.7** ([42], see [40, Sec. 3.6.2]): *Pseudorandom functions can be constructed using any pseudorandom generator.*

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).[2]

## 3.3 Zero-Knowledge

Zero-knowledge proofs, introduced by Goldwasser, Micali and Rackoff [51], are a powerful tool in 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. 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.

### 3.3.1 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

[2]The above methodology is sound provided that the adversary does not get the description of the pseudorandom function in use, but rather only (possibly limited) oracle access to it. This is different from the so-called Random Oracle Methodology formulated in [9] and criticized in [22].

an honest user. Our approach is that the adversary *gains nothing* if whatever it can obtain by unrestricted adversarial behavior can 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 beyond the validity of the assertion (i.e., the benign behavior is any computation that is based (only) on the assertion itself). Other examples appear and are discussed in Sections 4.1 and 5.1.

A notable property of the above 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 considers also "non-harmful" gains of some "far fetched" adversaries. We warn against this impression. Firstly, there is nothing more dangerous to cryptography than to consider "reasonable" adversaries (a notion which is almost a contradiction in terms): typically, the adversaries will try exactly what the system designed has discarded as "far fetched". Secondly, it seems impossible to come up with definitions of security that distinguish "breaking the scheme in a harmful sense" from "breaking it in a non-harmful sense": 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".

### 3.3.2 The Actual Definition

Before defining zero-knowledge proofs, we have to define proofs. The standard notion of static (i.e., non-interactive) proofs will not do (because static zero-knowledge proofs exist only for sets that are easy to decide (i.e, are in $\mathcal{BPP}$) [47], 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 [51]). 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, whereas NO prover strategy may fool the verifier to accept false assertions. Both the above *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).

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 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 $C^*$ so that the following two probability ensembles are computationally indistinguishable[3]:

[3]Here we refer to a natural extension of Definition 3.3: 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 $Z_\alpha$ ranges over strings of length that is polynomially-related to the length of $\alpha$. 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 $\alpha \in S$,

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

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

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

The above definition does NOT account for auxiliary information that an adversary 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 [43, 47]). This is taken care of by a more strict notion called auxiliary-input zero-knowledge.

**Definition 3.8** (zero-knowledge [51], revisited [47]): *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. $\{(A,B^*(z))(x)\}_{x\in S\,,\,z\in\{0,1\}^{p(|x|)}} \stackrel{\rm def}{=}$ *the output of $B^*$ when having auxiliary-input $z$ and interacting with $A$ on common input $x \in S$; and*

2. $\{C^*(x,z)\}_{x\in S\,,\,z\in\{0,1\}^{p(|x|)}} \stackrel{\rm def}{=}$ *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. (The only exceptions are zero-knowledge proofs constructed on purpose in order to show a separation between these two notions [43].)

### 3.3.3 Zero-Knowledge Proofs for all NP-assertions and their applications

Assuming the existence of commitment schemes, which in turn exists if one-way functions exist [60, 54], there exist (auxiliary-input) zero-knowledge proofs for 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 [45] (and depicted in Figure 1), 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 proven). This makes zero-knowledge a very powerful tool in the design of cryptographic schemes and protocols.

In a typical cryptographic setting, a user referred to as $U$, has a secret and is supposed to take some steps depending on its secret. The question is how can other users verify that $U$ indeed took the correct steps (as determined by $U$'s secret and the publicly known information). Indeed, if $U$ discloses its secret then anybody can verify that $U$ took the correct steps. 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 actions without violating $U$'s interest in not revealing its secrets). That is, $U$ can prove in zero-knowledge that it took the correct steps. Note that $U$'s claim to having taken the correct steps is an NP-assertion (since $U$'s legal actions are 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 a NP-witness to the claim that the action fits the public information). Thus, by the above result, it is possible for $U$ to efficiently prove the correctness of its actions without yielding anything about its secret. Consequently, it is fair to ask $U$ to prove (in zero-knowledge) that it behaves properly, and so force $U$ to behave properly. Indeed, "forcing proper behavior" is the canonical application of zero-knowledge proofs (see also Section 5.3.1).

where the probabilities are taken over the relevant distribution (i.e., either $X_\alpha$ or $Y_\alpha$) and over the internal coin tosses of algorithm $D$.

Commitment schemes are digital analogies 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 to read the unique committed value (as when sending the key to the lock).
The following 4 steps are repeated $t \cdot |E|$ many times so to obtain $\exp(-t)$ soundness error.

**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 $\phi : V \to \{1, 2, 3\}$.

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

**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 1: The zero-knowledge proof of Graph 3-Colorability (of [45]). Zero-knowledge proofs for other NP-sets can be obtained using the standard reductions.

# 4 Encryption and Signature Schemes

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). For more details regarding the contents of the current section, see fragments of our forthcoming textbook [41].

## 4.1 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 which 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. 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 secret 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 ("classic") approach, introduced by Shannon [74], is *information theoretic*. It is concerned with the "information" about the plaintext which 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 [74]. 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 encryption schemes.

The second ("modern") approach, followed in the current text, is based on *computational complexity*. This approach is based on the observation 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 if the key is much shorter than the total length of the messages sent via the encryption scheme.

The computational complexity approach allows 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 [31]. 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 this 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 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, traditional encryption scheme in which the encryption-key equals the description-key are called *private-key* schemes, since in these schemes the encryption-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 key-generation (randomized) 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 consists of three efficient algorithms: a *key generation* algorithm denoted $G$, an *encryption* algorithm denoted $E$, and an *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 private-key encryption scheme. Below we focus on the public-key case.

### 4.1.1 Definitions

For simplicity we consider only the encryption of a single message (which, for further simplicity, is assumed to be of length $n$).[4] 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 as efficiently from the a-priori information alone. This definition (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, also when given the encryption-key. Both definitions were introduced by Goldwasser and Micali [50]:

**Definition 4.1** (semantic security (following [50])): *An 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 $|h(x)| = \text{poly}(|x|)$, 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*

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

*where $e$ is distributed according to $G(1^n)$ and $\mu$ is a negligible function.*

Note that no computational restrictions are made regarding the functions $h$ and $f$, and in particular it may be that $h(x) = (z_{|x|}, h'(x))$, where the sequence of $z_n$'s is possibly non-uniform. We stress that the above definition (as well as the next one) refers to public-key encryption schemes, and in case of private-key schemes algorithm $A$ is not given the encryption-key $e$.

**Definition 4.2** (indistinguishability of encryptions (following [50])): *An 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 $|z_n| = \text{poly}(n)$,*

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

*Again, $e$ is distributed according to $G(1^n)$, and $\mu$ 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 fix messages (such as the all-zero message and the all-ones message).

[4] In case of public-key schemes no generality is lost by these simplifying assumptions, but in case of private-key schemes one should consider the encryption of polynomially many messages.

**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. The same holds for *private-key* encryption schemes when considering the security of encrypting several messages (rather than a single message as done above). (Here, 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.) This explains the linkage between the above robust security definitions and the *randomization paradigm* (discussed below).

**Further discussion:** We stress that the (equivalent) definitions presented above 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.

### 4.1.2 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 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 3.2.2). An alternative and more flexible construction follows.

**Private-Key Encryption Scheme based on Pseudorandom Functions:** The key generation algorithm consists of selecting a seed, denoted $s$, for such a 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 3.2.3):

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 (since otherwise one could distinguish a pseudorandom function from a truly random one).

Note that we could have gotten rid of the randomization 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 a single party encrypts using the same key or that all parties that encrypt using the same key coordinate their actions (i.e., maintain a joint state (e.g., counter)). Indeed, 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.

**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 $T = (P, Q, N, c_P, d_P, c_Q, d_Q)$ is the decryption-key.

**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 $y = x \oplus \sigma_1\sigma_2 \cdots \sigma_{\ell(n)}$.

**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 $y \oplus \sigma_1\sigma_2 \cdots \sigma_{\ell(n)}$.

Comment: for every $s$, it holds that $(s^2)^{(P+1)/4} \equiv s \pmod{P}$, and so $(s^{2^{\ell(n)}})^{d_P} = s$.

Figure 2: The Blum–Goldwasser Public-Key Encryption Scheme [16]. For simplicity we assume that $\ell$, which is polynomially bounded (e.g., $\ell(n) = n$), is known at key-generation time.

We comment that the use of a counter (or any other state) in the encryption process is not reasonable in 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). Thus, probabilistic encryption plays even a more important role in case of public-key encryption schemes (than in case of private-key schemes). Following Goldwasser and Micali [50], 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 are going to use the RSA scheme [69] as a trapdoor permutation (rather than using it directly as an encryption scheme).[5] The RSA scheme has an instance-generating algorithm that randomly selects two primes, $p$ and $q$, computes their product $N = p \cdot q$, and selects at random a pair $(e, d)$ so that $e \cdot d \equiv 1 \pmod{\phi(N)}$, where $\phi(N) \stackrel{\rm def}{=} (p-1) \cdot (q-1)$. (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 (resp., decryption-key) is set to $(N, e)$ (resp., $(N, d)$), just as in "plain RSA". To encrypt *a single bit* $\sigma$ (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 $r$. To decrypt the ciphertext $(y, \tau)$ (using the decryption-key $(N, d)$), the decryption algorithm just computes $\tau \oplus \mathrm{lsb}(y^d \bmod N)$. The above scheme is quite wasteful in bandwidth; however, the paradigm underlying its construction is valuable in practice. For example, assuming the intractability of factoring large integers one may derive a secure public-key encryption scheme with efficiency comparable to that of RSA (see [16] and Figure 2). The ideas underlying both schemes can be applied using any trapdoor permutation (see [41]).

#### 4.1.3 Beyond eavesdropping security

The above definitions refer only to a "passive" attack in which the adversary merely eavesdrops 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. Loosely speaking, in such an attack, the adversary may obtain the decryption of any ciphertexts of its choice, and is deemed successful if it learns something regarding the plaintext that corresponds to some other ciphertext (see [55, 7]). Private-key and public-key encryption schemes secure against such attacks can be constructed under the same assumptions that suffice for the construction of the corresponding passive schemes. Specifically:

**Theorem 4.3** (folklore, see [41]): *Assuming the existence of one-way functions, there exist private-key encryption schemes that are secure against chosen ciphertext attack.*

**Theorem 4.4** ([32] and [15, 36], see [72]): *Assuming the existence of trapdoor permutations, there exist private-key encryption schemes that are secure against chosen ciphertext attack.*

Security against chosen ciphertext attack is related to the notion of *non-malleability* of the encryption scheme (cf. [32]). 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 of the form $1x$, it is infeasible to produce a ciphertext to the plaintext $0x$). For further discussion see [32, 7, 55].

### 4.2 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 verifiable by parties that are also able to generate them.

[5] 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.

**Signature Schemes:** The need to discuss "digital signatures" has arise with the introduction of computer communication in business environment (in which parties need to commit themselves to proposals and/or declarations they make) [31, 65]. 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 his own signature* on documents of his 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 himself, a universally agreed verification procedure, and the belief (or assertion) that it is infeasible (or at least hard) to forge signatures in a manner that pass the verification procedure. It is not clear to what extent do handwritten signatures meet these requirements. In contrast, our discussion of digital signatures provides precise statements concerning the extend by which digital signatures meet the above requirements. Furthermore, unforgeable digital signature schemes can be constructed based on some reasonable computational assumptions (i.e., 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 his 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. That is, only the 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 (only 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.

Formally speaking, both signature schemes and message authentication schemes consist of three efficient algorithms: *key generation*, *signing* and *verification*. As in 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 case of signature scheme, the adversary is given the verification-key, whereas in case of message authentication scheme 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 functionality (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.)

### 4.2.1 Definitions

We focus on the definition of secure signature schemes. Following Goldwasser, Micali and Rivest [52], 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 consider a breaking of the scheme).

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 succeeds (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.

We stress that *plain* RSA (alike plain versions of Rabin's scheme [66] and the DSS [63]) 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. Thus, the randomization paradigm seems pivotal here too.

### 4.2.2 Constructions

Secure *message authentication schemes* can be constructed using pseudorandom functions [42]. Specifically, the key-generation algorithm consists of selecting a seed $s \in \{0,1\}^n$ for such a function, denoted $f_s$, and the (only valid) tag of message $x$ with respect to the key $s$ is $f_s(x)$. We comment that an *extensive* usage of pseudorandom functions seem an overkill for achieving message authentication, and more efficient schemes may be obtained based on other cryptographic primitives (cf. [6]).

Constructing secure *signature schemes* seems more difficult than constructing message authentication schemes. Three central paradigms in the construction of *signature schemes* are the "refreshing" of the "effective" signing-key, the usage of an "authentication tree" and the "hashing paradigm". The first paradigm is aimed at limiting the potential dangers of a chosen message attack by signing the actual document using a newly (randomly) generated instance of the signature scheme, and authenticating (the verification-key of) this random instance relative to the fixed public-key. A natural way of carrying-on the authentication of the many newly generated keys is by using an "authentication tree" [58]. Finally, the hashing paradigm refers to the common practice of signing documents via a two stage process: First the actual document is hashed into 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 *Universal One-Way Hash Functions* (cf. [62]). We conclude by mentioning that secure signature schemes can be constructed based on any one-way function. Furthermore:

**Theorem 4.5** ([62, 71], see [41]): *The following three conditions are equivalent.*

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

# 5 Cryptographic Protocols (as stand-alone)

A general framework for casting cryptographic ($m$-party protocol) problems consists of specifying a random process that maps $m$ inputs to $m$ outputs.[6] The inputs to the process are to be thought of as local inputs of $m$ parties, and the $m$ outputs are their corresponding local outputs. The random process describes the desired functionality. That is, if the $m$ parties were to trust each other (or trust some outside 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 mentioned in the introduction and 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").

## 5.1 The Definitional Approach and Some Models

Before describing these 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 [49, 59, 3, 4, 18, 19])[7]. The approach can be traced back to the definition of zero-knowledge

[6] That is, we consider a process $F$, which 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 $\ell'$ depending on $\ell$), and $F'$ is a function mapping $(m+1)$-long sequences to $m$-long sequences.

[7] We refer the reader to Canetti's work [19], which provides a relatively simple, flexible and comprehensive treatment of the definitions of secure multi-party computation. Canetti's work may be viewed as a minimalistic instantiation of the definitional approach of Micali and Rogaway [59], where minimality refers to possible augmentations of the

(cf. [51]), and even to the definition of secure encryption (cf. [38], rephrasing [50]). The underlying paradigm (called the simulation paradigm (cf. §3.3.1)) is that a scheme is secure if whatever a feasible adversary can obtain after attacking it, is also feasibly attainable "from scratch". In 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 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 adversaries can feasibly obtain in the former actual setting can also be feasibly obtained in the latter ideal setting then the protocol "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.[8] We first discuss some of the parameters used in defining various models, and next demonstrate the application of this approach in one (specific) important model. For further detail, see [19].

### 5.1.1 Some parameters used in defining security models

The following parameters are described in terms of the actual computation. In some cases, the corresponding definition of security is obtained by some restrictions or provisions applied to 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 (as the real-model adversary).

- *The communication channels*: 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. [11, 25]). In addition, one may postulate the existence of a broadcast channel (cf. [68]). Each of these postulates may be justified in some settings. Furthermore, each postulate may be view as a useful abstraction that provide a clean model for study and development of secure protocols. In this respect, it is important to mention that in case of non-adaptive adversaries (see below), both types of channels can be easily emulated by ordinary "tapped channels" (by using encryption schemes and signature schemes, respectively).

  As mentioned in the introduction, most work in the area assume that communication is synchronous and that point-to-point channels exist between every pair of processors. However, one may also consider asynchronous communication (cf. [10]) and arbitrary networks of point-to-point channels (cf. [33]).

high-level approach as presented there. Beaver's papers [3, 4] have a similar approach. The approach of Goldwasser and Levin [49] is more general: it avoids the definition of security (w.r.t a given functionality) and instead defines a notion of protocol robustness.

[8] 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 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 functionality that are computable within any given (time-constructible) time bound, using adequate padding.

- *Computational limitations*: Typically, we consider computationally-bounded adversaries (e.g., probabilistic polynomial-time adversaries). However, the private-channel model allows us also to (meaningfully) consider computationally-unbounded adversaries.

  We stress that, also in the latter case, security should be defined by saying 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). Thus, results in the computationally-unbounded adversary model trivially imply results for computationally-bounded adversaries.

- *Restricted adversarial behavior*: 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 do so based on partial information it has gathered so far (cf., [21]). A somewhat more restricted model, which seems adequate in many setting, 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.

  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., [45, 46, 39] and Section 5.3). Below we refer to the adversary of the unrestricted model as to *active*; another commonly used name is *malicious*.

- *Restricted notions of security*: One example is the willingness to tolerate "unfair" protocols in which the execution can be suspended 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.)

- *Upper bounds on the number of dishonest parties*: In some models secure multi-party computation is possible only if a majority of the parties are honest (cf., [11, 27]). Sometimes even a special majority (e.g., 2/3) is required.

- *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 [64], 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., [23]).

### 5.1.2 Example: Multi-party protocols with honest majority

Here we consider a non-adaptive, active, 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. Thus, the basic paradigm underlying the definitions of *secure multi-party computations* amounts to saying that the only situations that may occur in the real execution of a secure protocol, are those that can also occur in an 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 decided 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.

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 simulate 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 5.1** (protocol security definition – a sketch): *Let $f$ be an $m$-ary functionality and $\Pi$ be an $m$-party protocol operating in the real model.*

- *For a real-model adversary $A$, controlling some minority of the parties, and an $m$-sequence $\overline{x}$, we denote by $\mathrm{REAL}_{\Pi,A}(\overline{x})$ the sequence of $m$ outputs resulting from the execution of $\Pi$ 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 $\mathrm{IDEAL}_{f,A'}(\overline{x})$ 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 $\Pi$* 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 $\{\mathrm{REAL}_{\Pi,A}(\overline{x})\}_{\overline{x}}$ and $\{\mathrm{IDEAL}_{f,A'}(\overline{x})\}_{\overline{x}}$ are computationally indistinguishable (as in Footnote 3).*

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, since they can actually reproduce those by themselves.)

The fact that Definition 5.1 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 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 stated (during the execution). In the corresponding ideal-model, the adversary is not allowed to modify the *inputs* of dishonest parties (in Step 1).

We comment that a definition analogous to Definition 5.1 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 [29]. Indeed, secure multi-party computation with dishonest majority is possible if premature suspension of the execution is not considered a breach of security (which is formulated by allowing the ideal-model adversary to suspend the execution after the trusted party has handed local outputs to some (but not all) of the parties).

### 5.2 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 given functionality). We mention that the first results of this type were obtained by Goldreich, Micali, Wigderson and Yao [45, 76, 46].

- *Assuming the existence of trapdoor permutations*, secure multi-party computation is possible in the following models (cf. [45, 76, 46] and details in [39]):

  1. Passive adversary, for any number of dishonest parties (cf. [39, Sec. 3.2]).
  2. Active adversary that may control only a minority of the parties (cf. [39, Sec. 3.3.3]).
  3. Active adversary, for any number of bad parties, provided that suspension of execution is not considered a violation of security (i.e., see restricted notion of security above). (Cf. [39, Sec. 3.3.2].)

  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).

- Making no computational assumptions and allowing computationally-unbounded adversaries, but *assuming private channels*, secure multi-party computation is possible in the following models (cf. [11, 25]):

  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.[9]

  In both cases the adversary may be adaptive (cf. [11, 21]).

- 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 [64]. 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 *adaptive* and active computationally-bounded adversary that may control only less than one third of the parties [21, 30]. 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 [10, 12] and [33], respectively.

**Secure reactive computation:** All the above results 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 from the party corresponding outputs. That is, a reactive system goes through (an 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 $j$th iteration.

A more general formulation allows the outputs of each iteration to depend also on a global state, which is possibly updated by 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 [39, Sec. 4.1].

**Efficiency considerations:** 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 case of multi-party computations with honest majority (cf. [5]) and in case of two-party computations allowing abort (cf. [57]).

### 5.3 Construction Paradigms

We briefly sketch three 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 [45, 76, 46]. These constructions proceed in two steps (see details in [39]). First

[9] Fault-tolerance can be increased to a regular minority if broadcast channels exists [68].

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).

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

### 5.3.1 Compilation of passively-secure protocols into actively-secure ones

This is achieved mainly via using zero-knowledge proofs to force parties to behave in a way consistent with the protocol, as described in Section 3.3.3. Actually, we need to force 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 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 can take place. Specifically, the compiled protocol (which is executed over a broadcast channel) proceeds as follows:

1. Prior to the emulation of the original protocol, each party commits to its input (using a commitment scheme [60]). In addition, using a zero-knowledge proof-of-knowledge [51, 8, 45], 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 [28].)

2. Next, all parties jointly generate a sequence of random bits for each party so that only this party knows the outcome of the random sequence generated for it, but everybody gets a commitment to this outcome. This sequence will be used as the random-input (i.e., sequence of coin tosses) for the original protocol. Each bit in the sequence is determined as the exclusive-or of the outcomes of instances of an (augmented) coin-tossing protocol (cf. [13] and [39, Sec. 2.3.2.1]) that each party plays with each of all the other parties.

3. In addition, when compiling to 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 protocol (cf. [26] and [39, Sec. 3.3.3.1]). This will guarantee that if some party prematurely suspends the execution, then all the parties can together reconstruct all its secrets and carry-on the execution while playing its role.

4. 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 that asserts that the message was indeed computed correctly. An important detail to note here is that all 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). 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. However, the sender's input is determined by its commitment (as sent in Step 1), its random-input is similarly determined (in Step 2), and the messages it has

received are publicly known. Thus, the next message as determined by the original protocol is a function of publicly known strings. Furthermore, the assertion that the next message was indeed computed correctly is an NP-assertion, and the sender knows a corresponding NP-witness. Thus, it 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 [45, 46]. A detailed description and full proofs appear in [39].

### 5.3.2 Passively-secure computation with "encrypted circuits"

This technique refers mainly to two-party computation. Suppose that two parties, each having a private input, wish to obtain the value of a predetermine two-argument function evaluated at their input pair. Further suppose that the two parties hold a circuit that computes the value of the function on inputs of the adequate length. The idea is to have one party construct an "encrypted" form of the circuit so that the other party can propagate encrypted values through the "encrypted gates" and obtain the output in cleartext (while all intermediate values remain secret). Note that the roles of the two parties are not symmetric, and recall that we are describing a protocol that is secure (only) with respect to passive adversaries. An implementation of this idea proceeds as follows:

- The the first party will construct an "encrypted" form of the original circuit with its own input wired-in. Thus, the value of this "encrypted" circuit depends only on the input of the second party. The first party sends this "encrypted" circuit to the second party.

- The "encrypted" circuit consists of *pairs of keys* that correspond to each wire of the original circuit and *templates* that correspond to the gates of the original circuit. These templates are constructed so that if one knows a key that correspond to each wire entering any gate in the circuit then one can obtain a key corresponding to the wire that exits the gate. Furthermore, there is a *random correspondence* between the key pairs and the possible Boolean values of the corresponding wires, and the reconstruction of keys using each template respects the functionality of the corresponding gate (i.e., if one knows the key that corresponds to the 1-value of one entry-wire and the key that corresponds to the 0-value of the other entry-wire, and the gate is an OR-gate, then one obtains the key that corresponds to the 1-value of exit-wire).

  Each template may consist of 4 gadgets that are presented in random order, where each gadget corresponds to one of the 4 possible values of the two entry-wires. A gadget may be merely a double encryption of the key that corresponds to the appropriate output value, where encryption uses the two keys that correspond to the input values.

- In addition, the first party uses an Oblivious Transfer protocol (cf. [67, 35, 45] and [39, Sec. 2.2.2]) in order to hand the second party keys corresponding to the second party's input bits (without the first party learning anything about these bits).

- Finally, the second party "evaluates" the "encrypted" circuit gate-by-gate, starting from the top (circuit-input) gates (for which it knows all keys) and ending at the bottom (circuit-output) gates (for which, by construction, the correspondence of keys to values is known). Thus, the second party obtains the output value of the circuit (but nothing else), and sends it to the first party.

The above description is based on oral presentations of Yao's work [76]. More detailed descriptions appeared in several subsequent papers (e.g., [5, 61]).

### 5.3.3 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 predetermine $m$-argument function evaluated at their sequence of inputs. Further suppose that the parties hold a circuit that computes the value of the function on inputs of the adequate length, and that the gate only consists of AND and NOT gates. Again, the idea is to propagate information from the top gates to the bottom gates, but this time the information is different, and the propagation is done simultaneously by all parties. The idea is to share the value of each wire in the circuit so 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. [73]) so that a bit $b$ is shared by a random sequence of $m$ bits that sum-up to $b$ mod 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 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 wire 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 $i$):

$$((x_1, y_1), ..., (x_m, y_m)) \mapsto (z_1, ..., z_2) \tag{1}$$
$$\text{where } \textstyle\sum_{i=1}^m z_i = (\sum_{i=1}^m x_i) \cdot (\sum_{i=1}^m y_i). \tag{2}$$

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

Thus, 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

$$\left(\sum_{i=1}^m x_i\right) \cdot \left(\sum_{i=1}^m y_i\right) = \sum_{i=1}^m x_i y_i + \sum_{1 \leq i < j \leq m} (x_i y_j + x_j y_i) \tag{3}$$
$$\equiv (1 - (m-1)) \cdot \sum_{i=1}^m x_i y_i + \sum_{1 \leq i < j \leq m} (x_i + x_j) \cdot (y_i + y_j) \mod 2 \tag{4}$$

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 $z_{i,i} \stackrel{\text{def}}{=} m \cdot x_i y_i$. (Note that $2 - m \equiv m \pmod 2$.)
2. Next, each pair of parties (i.e., Parties $i$ and $j$) securely compute random shares of $(x_i + x_j) \cdot (y_i + y_j)$. 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 $z_{i,j} + z_{j,i} = (x_i + x_j) \cdot (y_i + y_j)$. The latter secure two-party computation can be reduced to (a 1-out-of-4) Oblivious Transfer (cf. [48] and [39, Sec. 2.2.3]).

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 [46]. A detailed description and full proofs appear in [39].

We mention that an analogous construction has been subsequently used in the private channel model and withstands computationally unbounded adversaries that control less than one third of the parties [11]. The basic idea is to use a more sophisticated secret sharing scheme; specifically, via a low degree polynomials [73]. 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 $d = \lfloor (m-1)/3 \rfloor - 1$ having 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). For details see [11, 37].

# 6 Cryptographic Protocols (under concurrent execution)

The definitions and results surveyed in Section 5 refer to a setting in which, at each time, only a single execution 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 enforce a single execution setting). Still, the definitions and results obtained in the single execution setting serves as a good starting point for the study of security in the setting of concurrent executions.

The study of security in the setting of concurrent executions is currently at its early stages: Central issues and difficulties have been identified, and approaches addressing them have been suggested. On the other hand, there are fundamental gaps in our current understanding, and so the current time seems inadequate for a conclusive description of this area. We thus confine ourselves to discussing some of the issues and difficulties involved and presenting some of the positive results that were achieved. We believe (and hope) that in a few years the current discussion will be only of historical value.

## 6.1 Some issues and some definitional approaches

As in 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 $n$).[10] 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 uniformly selected pair of $n$-bit long strings, denoted $(\beta, \gamma)$. 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 $(\beta, f(\alpha\beta))$.

Observe that $P$'s strategy is zero-knowledge (even w.r.t auxiliary-inputs as defined in Definition 3.8): Intuitively, if the adversary $A$ chooses the case $v = 1$, then it is infeasible for $A$ to guess a passing pair $(\beta, \gamma)$ with respect to a random $\alpha$ 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 $\beta$ is selected uniformly by $P$, and for any $\alpha$ 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 $\alpha$, 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 $(\beta, f(\alpha\beta))$ 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 $(\beta, f(\alpha\beta) \oplus \alpha \oplus \beta)$, rather than with $(\beta, f(\alpha\beta))$. Clearly, 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 the 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).[11] 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.

[10] In fact, assuming that $P$ shares a pseudorandom function $f$ with his friends (as explained in Section 3.2.3), the above protocol is an abstraction of a natural "mutual identification" protocol. (The example is adapted from [43].)

[11] 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 [32]. We comment that providing a general definition of what "correlated outputs" means seems very challenging (if at all possible). Indeed the focus of [32] is on several important special cases such as encryption and commitment schemes.

A much more appealing approach was recently suggested by Canetti [20]. Loosely speaking, Canetti suggests to consider a protocol to be secure (called *environmentally-secure*) only if it remains secure when executed within any (feasible) environment. Following the simulation paradigm, we get the following definition:

**Definition 6.1** (environmentally-secure protocols [20] – a rough sketch): *Let $f$ be an $m$-ary functionality and $\Pi$ 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 $\Pi$ 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 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 $\Pi$.*

in the ideal model *the* (simulating) *adversary controls the same parties, which use an ideal* (trusted-party) *that behaves according to the functionality $f$. All parties can communicate with the 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* $\Pi$ 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 accounts 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 any 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 [20]. We wonder whether the reverse direction holds.

**Timing assumptions.** The above issues (as well as the specific example used) remain valid also in a synchronous environment, but are indeed amplified in an asynchronous environment. Thus, one approach may be to rely some kind of weak timing assumption (as done in [34] (in the context of concurrent zero-knowledge)). However, using timing assumptions to achieve concurrent (rather than stand-alone) security typically means imposing a significant delay in each execution (rather than delaying only executions that are actively tampered by an adversary).[12]

[12] A typical use of a timing assumption in the context of stand-alone security is to delay some operation until either a specific message arrives or some large amount of time (i.e., larger than typical message delays) has elapsed. Clearly, this effects only executions in which non-typical message delays (typically caused by the adversary) occur. However, in the context of concurrent security, the delay is typically used with respect to operations taking place in other sessions, and so the delay has to be imposed on the current execution unconditionally (because the relevant message arrival event belongs to a different session and so cannot be tested in the current session).

## 6.2 Some difficulties

As stated above, 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 [43, 24] point out inherent limitation of the "standard proof methods" (used to establish zero-knowledge) when applied to the concurrent setting, where [43] treats the synchronous case and [24] 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 (i.e., the machine $C^*$ in Item 2 of Definition 3.8 is a universal oracle machine that has oracle access to the strategy of $B^*$).

Note that since Definition 3.8 refers to all possible adversaries, it seems that a demonstration that it holds must be universal in the sense that it should provide a (general) procedure for constructing a simulator for each possible adversary (i.e., transforming any $B^*$ to a corresponding $C^*$). Furthermore, this procedure cannot be expected to "reverse engineer" the adversary $B^*$, it seems natural to assume that the simulator must use $B^*$ as an oracle (or, as a "black-box"). However, this assumption has be refuted recently by Barak, who has shown that the simulator may use the code of $B^*$ in a meaningful way [2] (and not just by invoking it on inputs of its choice). (On the other hand, arguably, the simulator constructed in [2] does not "reverse engineer" $B^*$.) Indeed, Barak [2] presents several results (regarding zero-knowledge) that are impossible to achieve by simulators that only obtain oracle access to the adversary strategy (rather than its code). In particular, his result regarding concurrent zero-knowledge goes beyond the impossibility result of [24] (which refers to black-box simulations).

In general, simulation via oracle access to the adversary is very problematic in the context of concurrent executions (much more than in the context of stand-alone security). Thus, the newly discovered (non-"black-box") simulation techniques of Barak [2] carry great promise for obtaining stronger results regarding concurrent security. On the other hand, the recent definitional approach of Canetti [20] relies heavily on oracle access (to an environment). In fact, it seems that general secure two-party computation is not possible under his definitions (except as discussed in the next subsection). It remains to be see if this conflict can be reconciled.

## 6.3 Some currently-known positive results

The main positive result known is that environmentally-secure protocols for any functionality can be constructed for settings in which two-thirds of the active parties are honest [20]. 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 two-thirds of the parties are honest.

In contrast, general environmentally-secure *two-party* computation is not possible (in the standard sense).[13] 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 at least two-thirds of the servers (in this set) are honest. Then, with the active participation of this set of servers, the users can compute any functionality in an environmentally-secure manner.

[13] Of course, some specific two-party computations do have environmentally-secure protocols. See [20] for several important examples (e.g., key exchange).

## Acknowledgments

I wish to thank Ran Canetti and Yehuda Lindell for helpful discussions.

## References

[1] 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.

[2] B. Barak. How to Go Beyond the Black-Box Simulation Barrier. Submitted to *42nd IEEE Symposium on Foundations of Computer Science*, 2001.

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

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

[5] 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 [70].

[6] 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.

[7] 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.

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

[9] 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.

[10] 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 [18].

[11] 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.

[12] 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.

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

[14] 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 [15].)

[15] 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 [14].

[16] 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.

[17] 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 IEEE Symposium on Foundations of Computer Science*, 1982.

[18] 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://theory.lcs.mit.edu/~tcryptol/BOOKS/ran-phd.html`.

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

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

[21] 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.

[22] 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.

[23] 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.

[24] 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 xxx–xxx, 2001.

[25] 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.

[26] 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.

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

[28] 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.

[29] 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.

[30] 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.

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

[32] D. Dolev, C. Dwork, and M. Naor. Non-Malleable Cryptography. In *23rd ACM Symposium on the Theory of Computing*, pages 542–552, 1991.

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

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

[35] S. Even, O. Goldreich, and A. Lempel. A Randomized Protocol for Signing Contracts. *CACM*, Vol. 28, No. 6, 1985, pages 637–647.

[36] 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.

[37] 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.

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

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

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

[41] O. Goldreich. *Foundation of Cryptography – Volume 2*. Working drafts for chapters regarding encryption schemes and signature schemes, 2000.
Available from `http://www.wisdom.weizmann.ac.il/~oded/foc-vol2.html`.

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

[43] 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.

[44] 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.

[45] 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 IEEE Symposium on Foundations of Computer Science*, 1986.

[46] 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 [39].

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

[48] 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.

[49] 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.

[50] 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 ACM Symposium on the Theory of Computing*, 1982.

[51] 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 ACM Symposium on the Theory of Computing*, 1985.

[52] 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.

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

[54] 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.

[55] 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.

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

[57] Y. Lindell. Parallel Coin-Tossing and Constant-Round Secure Two-Party Computation. In *Crypto01*, Springer-Verlag Lecture Notes in Computer Science (Vol. xxxx), 2001.

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

[59] 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.

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

[61] M. Naor, B. Pinkas, and R. Sumner. Privacy Preserving Auctions and Mechanism Design. In *Proc. of the 1st ACM conf. on Electronic Commerce*, Denver, Colorado, November 1999.

[62] 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.

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

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

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

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

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

[68] 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.

[69] 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

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

[71] 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.

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

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

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

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

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