# Foundations of Cryptography

## (Volume 2 – Basic Applications)

Oded Goldreich

Department of Computer Science and Applied Mathematics
Weizmann Institute of Science, Rehovot, Israel.

August 4, 2002

to Dana

# Preface

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

Eng. Isidor Goldreich (1906–1995)

Cryptography is concerned with the construction of schemes that withstand any abuse: Such schemes are constructed so to maintain a desired functionality, even under malicious attempts aimed at making them deviate from their prescribed functionality.

The design of cryptography 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 typically 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 about the environment in which a scheme is to operate, yet a cryptographic scheme has to operate in a maliciously selected environment which typically transcends the designer's view.

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

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

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

## Computational Difficulty

The specific constructs mentioned above (as well as most constructs in this area) can exist only if some sort of computational hardness exists. Specifically, all these problems and tools require (either explicitly or implicitly) the ability to generate instances of hard problems. Such ability is captured in the definition of one-way functions (see further discussion in Section 2.1). Thus, one-way functions is the very minimum needed for doing most sorts of cryptography. As we shall see, they actually suffice for doing much of cryptography (and the rest can be done by augmentations and extensions of the assumption that one-way functions exist).

Our current state of understanding of efficient computation does not allow us to prove that one-way functions exist. In particular, the existence of one-way functions implies that $\mathcal{NP}$ is not contained in $\mathcal{BPP} \supseteq \mathcal{P}$ (not even "on the average"), which would resolve the most famous open problem of computer science. Thus, we have no choice (at this stage of history) but to assume that one-way functions exist. As justification to this assumption we may only offer the combined believes of hundreds (or thousands) of researchers. Furthermore, these believes concern a simply stated assumption, and their validity follows from several widely believed conjectures which are central to some fields (e.g., the conjecture that factoring integers is hard is central to computational number theory).

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

## Structure and Prerequisites

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

Volume 1: Introduction and Basic Tools
- Chapter 1: Introduction
- Chapter 2: Computational Difficulty (One-Way Functions)
- Chapter 3: Pseudorandom Generators
- Chapter 4: Zero-Knowledge Proofs

Volume 2: Basic Applications
- Chapter 5: Encryption Schemes
- Chapter 6: Signature Schemes
- Chapter 7: General Cryptographic Protocols

Volume 3: Beyond the Basics
- . . .

Figure 0.1: Organization of this book

**Organization of the book.** The book is organized in three parts (see Figure 0.1): *Basic Tools*, *Basic Applications*, and *Beyond the Basics*. The first volume contains an introductory chapter as well as the first part (Basic Tools). This part contains chapters on computational difficulty (one-way functions), pseudorandomness and zero-knowledge proofs. These basic tools will be used for the Basic Applications of the second part, which consist of Encryption, Signatures, and General Cryptographic Protocols.

The partition of the book into three parts is a logical one. Furthermore, it offers the advantage of publishing the first part without waiting for the completion of the other parts. Similarly, we hope to complete the second part within a couple years, and publish it without waiting for the third part.

**The current manuscript.** The current manuscript consists of fragments of a chapter on encryption schemes. These fragments provide a draft of the first three sections of this chapter, covering the basic setting, definitions and constructions. Also included is a plan of the fourth section (i.e., *beyond eavesdropping security*), fragments for the Miscellaneous section of this chapter, and the above extracts from the preface of Volume 1.

# Part II

# Basic Applications

# Appendix C

# Corrections and Additions to Volume 1

In this appendix we list a few corrections and additions to the previous chapters of this work, which appeared in [124].

## C.1 Enhanced Trapdoor Permutations

Recall that a collection of trapdoor permutations, as defined in Definition 2.4.5, is a collection of permutations, $\{p_\alpha\}_\alpha$, armed with four probabilistic polynomial-time algorithms, denoted here by $I, S, F$ and $B$ (for *index*, *sample*, *forward* and *backward*), such that the following (syntactic) conditions hold

1. On input $1^n$, algorithm $I$ selects a random $n$-bit long *index* $\alpha$ of a permutation $p_\alpha$, along with a corresponding trapdoor $\tau$;
2. On input $\alpha$, algorithm $S$ *samples* the domain of $p_\alpha$, returning a random element in it;
3. For $x$ in the domain of $p_\alpha$, given $\alpha$ and $x$, algorithm $F$ returns $p_\alpha(x)$ (i.e., $F(\alpha, x) = p_\alpha(x)$);
4. For $y$ in the range of $p_\alpha$ if $(\alpha, \tau)$ is a possible output of $I(1^n)$ then, given $\tau$ and $y$, algorithm $B$ returns $p_\alpha^{-1}(y)$ (i.e., $B(\tau, y) = p_\alpha^{-1}(y)$).

The hardness condition in Definition 2.4.5 refers to the difficulty of inverting $p_\alpha$ on a random element of its range, when given only the range-element and $\alpha$. That is, let $I_1(1^n)$ denote the first element in the output of $I(1^n)$ (i.e., the index), then for every probabilistic polynomial-time algorithm $A$ (resp., every non-uniform family of polynomial-size circuit $A = \{A_n\}_n$), every polynomial $p$ and all sufficiently large $n$'s

$$\mathsf{Pr}[A(I_1(1^n), p_{I_1(1^n)}(S(I_1(1^n))) = S(I_1(1^n))] < \frac{1}{p(n)} \qquad \text{(C.1)}$$

Namely, $A$ (resp., $A_n$) fails to invert $p_\alpha$ on $p_\alpha(x)$, where $\alpha$ and $x$ are selected by $I$ and $S$ as above. An equivalent way of writing Eq. (C.1) is

$$\mathsf{Pr}[A(I_1(1^n), S'(I_1(1^n), R_n)) = p_{I_1(1^n)}(S'(I_1(1^n)), R_n)] \;<\; \frac{1}{p(n)} \qquad \text{(C.2)}$$

where $S'$ is the residual two-input (deterministic) algorithm obtained from $S$ when treating the coins of the latter as an auxiliary input, and $R_n$ denote the distribution of the coins of $S$ on $n$-bit inputs.

Although the above definition suffices for many applications, in some cases we will need an enhanced hardness condition. Specifically, we will require that it is hard to invert $f_\alpha$ on a random input $x$ (in the domain of $f_\alpha$) even when given the coins used by $S$ in the generation of $x$. (Note that given these coins (and the index $\alpha$), the resulting domain element $x$ is easily determined.)

**Definition C.1.1** (enhanced trapdoor permutations): *Let $\{f_\alpha : D_\alpha \to D_\alpha\}$ be a collection of trapdoor permutations as in Definition 2.4.5. We say that this collection is* enhanced (and call it an *enhanced collection of trapdoor permutations*) *if for every probabilistic polynomial-time algorithm $A$ every polynomial $p$ and all sufficiently large $n$'s*

$$\mathsf{Pr}[A(I_1(1^n), R_n) = p_{I_1(1^n)}(S'(I_1(1^n)), R_n)] \;<\; \frac{1}{p(n)} \qquad \text{(C.3)}$$

*where $S'$ is as above. The non-uniform version is defined analogously.*

We comment that the RSA collection (presented in Section **??** and further discussed in Section **??**) is in fact an *enhanced* collection of trapdoor permutations, provided that RSA is hard to invert in the same sense as assumed in Section **??**. In contrast, the Rabin Collection (as defined in Section 2.4.3), does not satisfy Definition C.1.1 (because the coins of the samling algorithm give away a modular square root of the domain element). Still, the Rabin Collection can be easily modify to yield an *enhanced* collection of trapdoor permutations, provided that factoring is hard (in the same sense as assumed in Section 2.4.3). Actually, we present two such possible modifications:

1. *Modifying the functions.* Rather than squaring modulo the composite $N$, we consider the function of raising to the power of 4 modulo $N$. It can be shown that the resulting permuations over the quadratic residues modulo $N$ satisfy Definition C.1.1, provided that factoring is hard. Specifically, given $N$ and a random $r \in Z_N$, ability to extract the 4th root of $r^2 \bmod N$ (modulo $N$), yields ability to factor $N$, where the algorithm is similar to the one used in order to establish the intractability of extracting square roots.

2. *Changing the domains.* Rather than considering the permutation induced (by the modoluar squaring function) on the set $Q_n$ of the quadratic residues modulo $N$, we consider the permulations induced on the set $M_n$, where

> $M_n$ contains all integers is $\{1, ..., N/2\}$ that have Jacobi symbol modulo $N$ that equals 1. Note that, as in case of $Q_n$, each quadratic residue has a unique square root in $M_n$ (because exactly two square roots have Jacobi symbol that equals 1 and their sum equals $N$).[1] However, unlike $Q_N$, membership in $M_N$ can be determined in polynomial-time (when given $N$ without its factorization). Thus, sampling $M_N$ can be done in probabilistic polynomial-time.
>
> Actually, squaring modulo $N$ is a 1-1 mapping of $M_N$ to $Q_N$. In order to obtain a permutation over $M_N$, we modify the function a little such that is the result of modular squaring is bigger than $N/2$ then we use its additive inverse (i.e., rather than $y > N/2$, we output $N - y$).

We comment that the special case of Definition 2.4.5 in which the domain of $f_\alpha$ equals $\{0,1\}^{|\alpha|}$ is a special case of Definition C.1.1 (because, without loss of generality, the sampling algorithm may satisfy $S'(\alpha, r) = r$). Clearly, the above examples can be slightly modified to fit this special case.

**Correction to Volume 1:** Theorems 4.10.10, 4.10.14 and 4.10.16 (which in turn are based on Remark 4.10.6) refer to the existence of certain non-interactive zero-knowledge proofs. The claimed non-interactive zero-knowledge proof systems can be constructed assuming the existence of an *enhanced* collection of trapdoor permutations. However, in contrast to the original text, it is not known how to derive these proof systems based on the existence of a (regular) collection of trapdoor permutations.

## C.2 Recent developments regarding zero-knowledge

A recent result by Barak [12] calls for re-evaluation of the significance of all negative results regarding black-box zero-knowledge[2] (as defined in Definition 4.5.10). In particular, relying on standard intractability assumptions, Barak presents round-efficient public-coin zero-knowledge arguments for $\mathcal{NP}$ (using non-black-box simulators), whereas only $\mathcal{BPP}$ can have such black-box zero-knowledge arguments (see comment following Theorem 4.5.11). Interestingly, Barak's simulator works in strict (rather than expected) probabilistic polynomial-time, addressing an open problem mentioned in Section 4.12.3. Barak's result is further described in Section C.2.2

In Section C.2.1, we report on recent progress achieved with respect to preservation of zero-knowledge under concurrent composition. We seize the oppertunity to provide a wider perspective on the question of preservation of zero-knowledge under various forms of protocol composition operations.

[1] As in case of $Q_n$, we use the fact that $-1$ has Jacobi symbol 1.

[2] Specifically, one should reject the interpretations of these results, which were offered in Sections 4.5.0, 4.5.4.0 and 4.5.4.2, by which such results indicate inherent limitations of zero-knowledge.

We mention that the two problems discussed in this section (i.e., the "preservation of security under various forms of protocol composition" and the "use of of the adversary's program within the proof of security") arise also with respect to the security of other cryptographic primitives. Thus, the study of zero-knowledge proofs serve as a good bench-mark for the study of various problems regarding cryptographic protocols.

### C.2.1 Composing zero-knowledge protocols

A natural question regarding zero-knowledge proofs (and arguments) is whether the zero-knowledge condition is preserved under a variety of composition operations. Three types of composition operation were considered in the literature: *sequential composition*, *parallel composition* and *concurrent composition*. We note that the preservation of zero-knowledge under these forms of composition is not only interesting on its own sake, but rather also sheds light of the preservation of the security of general protocols under these forms of composition.

We stress that when we talk of composition of protocols (or proof systems) we mean that the honest users are supposed to follow the prescribed program (specified in the protocol description) that refers to a single execution. That is, the actions of honest parties in each execution are independent of the messages they received in other executions. The adversary, however, may coordinate the actions it takes in the various executions, and in particular its actions in one execution may depend also on messages it received in other executions.

Let us motivate the asymmetry between the independence of executions assumed of honest parties but not of the adversary. Coordinating actions in different executions is typically difficult but not impossible. Thus, it is desirable to use composition (as defined above) rather than to use protocols that include inter-execution coordination-actions, which require users to keep track of all executions that they perform. Actually, trying to coordinate honest executions is even more problematic than it seems because one may need to coordinate executions of *different* honest parties (e.g., all employees of a big cooperation or an agency under attack), which in many cases is highly unrealistic. On the other hand, the adversary attacking the system may be willing to go into the extra trouble of coordinating its attack in the various executions of the protocol.

For $T \in \{\texttt{sequential}, \texttt{parallel}, \texttt{concurrent}\}$, we say that a protocol is $T$-zero-knowledge if it is zero-knowledge under a composition of type $T$. The definitions of $T$-zero-knowledge are derived from the standard definition by considering appropriate adversaries (i.e., adversarial verifiers); that is, adversaries that can initiate a polynomial number of interactions with the prover, where these interactions are scheduled according to the type $T$.[3] The corresponding simulator (which, as usual, interacts with nobody) is required to produce an

[3]Without loss of generality, we may assume that the adversary never violates the scheduling condition; it may instead send an illegal message at the latest possible adequate time. Furthermore, without loss of generality, we may assume that all the adversary's messages are delivered at the latest possible adequate time.

output that is computationally indistinguishable from the output of such a type $T$ adversary.

### C.2.1.1 Sequential Composition

In this case, the protocol is invoked (polynomially) many times, where each invocation follows the termination of the previous one. At the very least, security (e.g., zero-knowledge) should be preserved under sequential composition, or else the applicability of the protocol is highly limited (because one cannot safely use it more than once).

We mention that whereas the "simplified" version (i.e., without auxiliary inputs, as in Definition 4.3.2) is not closed under sequential composition (cf. [131]), the actual version (i.e., with auxiliary inputs, as in Definition 4.3.10) is closed under sequential composition (see Section 4.3.4). We comment that the same phenomena arises when trying to use a zero-knowledge proof as a sub-protocol inside larger protocols. Indeed, it is for these reasons that the augmentation of the "most basic" definition by auxiliary inputs was adopted in all subsequent works.[4]

### C.2.1.2 Parallel Composition

In this case, (polynomially) many instances of the protocol are invoked at the same time and proceed at the same pace. That is, we assume a synchronous model of communication, and consider (polynomially) many executions that are totally synchronized so that the $i$th message in all instances is sent exactly (or approximately) at the same time. (Natural variants on this model are discussed below as well as at the end of Section C.2.1.3.)

It turns out that, in general, zero-knowledge is not closed under parallel composition. A simple counter-example (to the "parallel composition conjecture") is depicted in Figure C.1. This counter-example, which is adapted from [131], consists of a simple protocol that is zero-knowledge (in a strong sense), but is not closed under parallel composition (not even in a very weak sense).

We comment that, at the 1980's, the study of parallel composition was interpreted mainly in the context of *round-efficient error reduction* (cf. [100, 131]); that is, the construction of full-fledge zero-knowledge proofs (of negligible soundness error) by composing (in parallel) a basic zero-knowledge protocol of high (but bounded away from 1) soundness error. Since alternative ways of constructing constant-round zero-knowledge proofs (and arguments) were found (cf. [130, 99, 58]), interest in parallel composition (of zero-knowledge protocols) has died. In retrospect, this was a conceptual mistake, because parallel composition (and mild extensions of this notion) capture the preservation of security in a fully synchronous (or almost-fully synchronous) communication network. We

[4]Interestingly, the preliminary version of Goldwasser, Micali and Rackoff's work [151] used the "most basic" definition, whereas the final version of this work used the augmented definition. In some works, the "most basic" definition is used for simplicity, but typically one actually needs the augmented definition.

Consider a party $P$ holding a random (or rather pseudorandom) function $f:\{0,1\}^{2n} \rightarrow \{0,1\}^n$, and willing to participate in the following protocol (with respect to security parameter $n$). The other party, called $A$ for adversary, is supposed to send $P$ a binary value $v \in \{1,2\}$ specifying which of the following cases to execute:

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

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): 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[a] 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 the pair $(\beta, f(\alpha\beta))$ to the first session of $P$, this pair passes the check, and so $A$ obtains the desired secret.

[a] Dummy messages may be added (in both cases) in order to obtain the above scheduling in the perfectly parallel case.

Figure C.1: A counter-example (adapted from [131]) to the parallel repetition conjecture for zero-knowledge protocols.

note that the almost-fully synchronous communication model is quite realistic in many settings, although it is certainly preferable not to assume even weak synchronism.

Although, in general, zero-knowledge is not closed under parallel composition, under standard intractability assumptions (e.g., the intractability of factoring), there exists zero-knowledge proofs for $\mathcal{NP}$ that are closed under parallel composition. Furthermore, these protocols have a constant number of rounds (cf. [125] for proofs and [90] for arguments).[5] Both results extend also to concurrent composition in a synchronous communication model, where the extension is in allowing protocol invocations to start at different (synchronous) times (and in particular executions may overlap but not run simultaneously).

We comment that parallel composition is problematic also in the context of reducing the soundness error of arguments (cf. [30]), but our focus here is on

[5] In case of parallel-zero-knowledge *proofs*, there is no need to specify the soundness error because it can always be reduced via parallel composition. As mentioned above, this is not the case with respect to arguments.

the zero-knowledge aspect of protocols regardless if they are proofs, arguments or neither.

#### C.2.1.3 Concurrent Composition (with and without timing)

Concurrent composition generalizes both sequential and parallel composition. Here (polynomially) many instances of the protocol are invoked at arbitrary times and proceed at arbitrary pace. That is, we assume an asynchronous (rather than synchronous) model of communication.

In the 1990's, when extensive two-party (and multi-party) computations became a reality (rather than a vision), it became clear that it is (at least) desirable that cryptographic protocols maintain their security under concurrent composition (cf. [86]). In the context of zero-knowledge, concurrent composition was first considered by Dwork, Naor and Sahai [90]. Actually, two models of concurrent composition were considered in the literature, depending on the underlying model of communication (i.e., a *purely asynchronous model* and an *asynchronous model with timing*). Both models cover sequential and parallel composition as special cases.

**Concurrent composition in the pure asynchronous model.** Here we refer to the standard model of asynchronous communication. In comparison to the timing model, the pure asynchronous model is a simpler model and using it requires no assumptions about the underlying communication channels, but it seems harder to construct concurrent zero-knowledge protocols for this model. In particular, for a while it was not known whether concurrent zero-knowledge proofs for $\mathcal{NP}$ exist at all (in this model). Under standard intractability assumptions (e.g., the intractability of factoring), this question was affirmatively resolved by Richardson and Kilian [226]. Following their work, research has focused on determining the round-complexity of concurrent zero-knowledge proofs for $\mathcal{NP}$. This question is still opened, and the current state of the art regarding it is as follows:

- Under standard intractability assumptions, every language in $\mathcal{NP}$ has a concurrent zero-knowledge proof with *almost-logarithmically* many rounds (cf. [218], building upon [172], which in turn builds over [226]). Furthermore, the zero-knowledge property can be demonstrated using a black-box simulator (see definition in Section 4.5.4.2 and C.2.2).

- Black-box simulator cannot demonstrated the concurrent zero-knowledge property of non-trivial proofs (or arguments) having significantly less than logarithmically-many rounds (cf. Canetti *et. al.* [66]).[6]

[6]By *non-trivial* proof systems we mean ones for languages outside $\mathcal{BPP}$, whereas by *significantly less than logarithmic* we mean any function $f : \mathbb{N} \to \mathbb{N}$ satisfying $f(n) = o(\log n / \log\log n)$. In contrast, by *almost-logarithmically* we mean any function $f$ satisfying $f(n) = \omega(\log n)$.

- Recently, Barak [12] demonstrated that the "black-box simulation barrier" can be bypassed. With respect to concurrent zero-knowledge he only obtains partial results: constant-round zero-knowledge arguments (rather than proofs) for $\mathcal{NP}$ that maintain security as long as an a-priori bounded (polynomial) number of executions take place concurrently. (The length of the messages in his protocol grows linearly with this a-priori bound.)

Thus, it is currently unknown whether or not *constant-round* arguments for $\mathcal{NP}$ may be concurrent zero-knowledge (in the pure asynchronous model).

**Concurrent composition under the timing model:** A model of naturally-limited synchronousness (which certainly covers the case of parallel composition) was introduced by Dwork, Naor and Sahai [90]. Essentially, they assume that each party holds a local clock such that the relative clock rates are bounded by an a-priori known constant, and consider protocols that employ time-driven operations (i.e., `time-out` in-coming messages and `delay` out-going messages). The benefit of the timing model is that it is known to construct concurrent zero-knowledge protocols for it. Specifically, using standard intractability assumptions, *constant-round* arguments and proofs that are concurrent zero-knowledge under the timing model do exist (cf. [90] and [125], respectively). The disadvantages of the timing model are discussed next.

The timing model consists of the *assumption* that talking about the actual timing of events is meaningful (at least in a weak sense) and of the *introduction of time-driven operations*. The timing assumption amounts to postulating that each party holds a local clock and knows a global bound, denoted $\rho \geq 1$, on the relative rates of the local clocks.[7] Furthermore, it is postulated that the parties know a (pessimistic) bound, denoted $\Delta$, on the message-delivery time (which also includes the local computation and handling times). In our opinion, these timing assumptions are most reasonable, and are unlikely to restrict the scope of applications for which concurrent zero-knowledge is relevant. We are more concerned about the effect of the time-driven operations introduced in the timing model. Recall that these operations are the `time-out` of in-coming messages and the `delay` of out-going messages. Furthermore, typically the delay period is at least as long as the time-out period, which in turn is at least $\Delta$ (i.e., the time-out period must be at least as long as the pessimistic bound on message-delivery time so not to disrupt the proper operation of the protocol). This means that the use of these time-driven operations yields slowing down the execution of the protocol (i.e., running it at the rate of the pessimistic message-delivery time rather than at the rate of the actual message-delivery time, which is typically much faster). Still, in absence of more appealing alternatives (i.e., a constant-round concurrent zero-knowledge protocol for the pure asynchronous model), the use of the timing model may be considered reasonable. (We comment than other alternatives to the timing-model include various set-up assumptions; cf. [65, 81].)

[7]The rate should be computed with respect to reasonable intervals of time; for example, for $\Delta$ as defined below, one may assume that a time period of $\Delta$ units is measured as $\Delta'$ units of time on the local clock, where $\Delta/\rho \leq \Delta' \leq \rho\Delta$.

**Back to parallel composition:** Given our opinion about the timing model, it is not surprising that we consider the problem of parallel composition almost as important as the problem of concurrent composition in the timing model. Firstly, it is quite reasonable to assume that the parties' local clocks have approximately the same rate, and that drifting is corrected by occasional clock synchronization. Thus, it is reasonable to assume that the parties have approximately-good estimate of some global time. Furthermore, the global time may be partitioned into phases, each consisting of a constant number of rounds, so that each party wishing to execute the protocol just delays its invocation to the beginning of the next phase. Thus, concurrent execution of (constant-round) protocols in this setting amounts to a sequence of (time-disjoint) almost-parallel executions of the protocol. Consequently, proving that the protocol is parallel zero-knowledge suffices for concurrent composition in this setting.

**Relation to resettable zero-knowledge.** Going to the other extreme, we mention that there is a model of zero-knowledge that is even stronger than concurrent zero-knowledge (even in the pure asynchronous model). Specifically, "resettable zero-knowledge" as defined in [65], implies concurrent zero-knowledge.

### C.2.2 Using the adversary's program in the proof of security

Recall that the definition of zero-knowledge proofs states that whatever an efficient adversary can compute after interacting with the prover, can actually be efficiently computed from scratch by a so-called *simulator* (which works without interacting with the prover). Although the simulator may depend arbitrarily on the adversary, the need to present a simulator for each feasible adversary seems to require the presentation of a universal simulator that is given the adversary's strategy (or program) as another auxiliary input. The question addressed in this section is how can the universal simulator use the adversary's program.

The adversary's program (or strategy) is actually a function determining for each possible view of the adversary (i.e., its input, random choices and the message it has received so far) which message will be sent next. Thus, we identify the adversary's program with this next-message function. As stated above, until very recently, all universal simulators (constructed towards demonstrating zero-knowledge properties) have used the adversary's program (or rather its next-message function) as a black-box (i.e., the simulator invoked the next-message function on a sequence of arguments of its choice). Furthermore, in view of the presumed difficulty of "reverse engineering" programs, it was commonly believed that nothing is lost by restricting attention to simulators, called black-box simulators, that only make black-box usage of the adversary's program. Consequently, Goldreich and Krawczyk conjectured that impossibility results regarding black-box simulation represent inherent limitations of zero-knowledge itself, and studied the limitations of the former [131].

In particular, they showed that parallel composition of the protocol of Construction 4.4.7 (as well as of any constant-round public-coin protocol) *cannot be proven to be zero-knowledge using a black-box simulator*, unless the language (i.e., 3-Colorability) is in $\mathcal{BPP}$. In fact their result refers to any constant-round public-coin protocol with negligible soundness error, regardless of how such a protocol is obtained. This result was taken as strong evidence towards the conjecture that constant-round public-coin protocol with negligible soundness error *cannot be zero-knowledge* (unless the language is in $\mathcal{BPP}$).

Similarly, as mentioned in Section C.2.1.3, it was shown that protocols of sub-logarithmic number of rounds *cannot be proven to be concurrent zero-knowledge via a black-box simulator* [66], and this was taken as evidence towards the conjecture that such protocols cannot be *concurrent zero-knowledge.*

In contrast to these conjectures and supportive evidence, Barak showed how to constructed non-black-box simulators and obtained several results that were known to be unachievable via black-box simulators [12]. In particular, under standard intractability assumption (see also [14]), he presented constant-round public-coin zero-knowledge arguments with negligible soundness error for any language in $\mathcal{NP}$. (Moreover, the simulator runs in strict polynomial-time, which is impossible for black-box simulators of non-trivial constant-round protocols [16].) Furthermore, this protocol preserves zero-knowledge under a fixed[8] polynomial number of concurrent executions, in contrast to the result of [66] (regarding black-box simulators) that holds also in that restricted case. Thus, Barak's result calls for the re-evaluation of many common believes. Most concretely, it says that results regarding black-box simulators do not reflect inherent limitations of zero-knowledge (but rather an inherent limitation of a natural way of demonstrating the zero-knowledge property). Most abstractly, it says that there are meaningful ways of using a program other than merely invoking it as a black-box.

Does this means that a method was found to "reverse engineer" programs or to "understand" them? We believe that the answer is negative. Barak [12] is using the adversary's program in a significant way (i.e., more significant than just invoking it), without "understanding" it. *So how does he use the program?*

The key idea underlying Barak's argument system [12] is to have the prover prove that either the original NP-assertion is valid or that he (i.e., the prover) "knows the verifier's residual strategy" (in the sense that it can predict the next verifier message). Indeed, in a real interaction (with the honest verifier), it is infeasible for the prover to predict the next verifier message, and so computational-soundness of the protocol follows. However, a simulator that is given the code of the verifier's strategy (and not merely oracle access to that code), can produce a valid proof of the disjunction by properly executing the sub-protocol using its

[8]The protocol depends on the polynomial bounding the number of executions, and thus is not known to be concurrent zero-knowledge (because the latter requires to fix the protocol and then consider any polynomial number of concurrent executions).

knowledge of an NP-witness for the second disjunctive. The simulation is computational indistinguishable from the real execution, provided that one cannot distinguish an execution of the sub-protocol in which one NP-witness (i.e., an NP-witness for the original assertion) is used from an execution in which the second NP-witness (i.e., an NP-witness for the auxiliary assertion) is use. That is, the sub-protocol should be a *witness indistinguishable* argument system (see Sections 4.6 and 4.8). We warn the reader that the actual implementation of the above idea requires overcoming several technical difficulties (cf. [12, 14]).

**Perspective.** In retrospect, taking a wide perspective, it should not come as a surprise that the program's code yields extra power beyond black-box access to it. Feeding a program with its own code (or part of it) is the essence of the diagonalization technique, and this too is done without "reverse engineering". Furthermore, various non-black-box techniques have appeared before in the cryptographic setting, but they were used in the more natural context of devising an attack on an (artificial) insecure scheme (e.g., towards proving the failure of the "Random Oracle Methodology" [64] and the impossibility of software obfuscation [15]). In contrast, in [12] (and [13]) the code of the adversary is being used within a sophisticated proof of security. What we wish to highlight here is that non-black-box usage of programs is relevant also to proving (rather than to disproving) the security of systems.

### Digest: Witness Indistinguishability and the FLS-Technique

The above description (of [12]), as well as several other sophisticated constructions of zero-knowledge protocols (e.g., [98, 226]), makes crucial use of a technique introduced by Feige, Lapidot and Shamir [98], which in turn is based on the notion of witness indistinguishability (introduced by Feige and Shamir [100]). Below, we will refer to strong witness indistinguishable protocols as defined in Definition 4.6.2. This technique, hereafter referred to as the FLS-technique, was used in Construction 4.10.12, but we wish to further discuss it below.

Following is a sketchy description of a special case of the FLS-technique, whereas the abovementioned application uses a more general version (which refers to proofs of knowledge, as defined in Section 4.7).[9] In this special case, the technique consists of the following construction schema, which uses (strong) witness indistinguishable protocols for $\mathcal{NP}$ in order to obtain zero-knowledge protocols for $\mathcal{NP}$. On common input $x \in L$, where $L = L_R$ is the NP-set defined by the witness relation $R$, the following two steps are performed:

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

[9]In the general case, the generation protocol may generate instances $x'$ in $L'$, but it is infeasible for the prover to obtain a corresponding witness (i.e., a $w'$ such that $(x', w') \in R'$). In the second step, the sub-protocol in use ought to be a proof of knowledge, and computational-soundness of the main protocol will follows (because otherwise the prover, using a knowledge extractor, can obtain a witness for $x' \in L'$).

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

(b) Loosely speaking, it is feasible to generate a transcript of the generation protocol that is computationally indistinguishable from the real interaction *along with an NP-witness for the outcome of the protocol.*

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

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

## C.3 Miscellaneous

### C.3.1 Additional Corrections

1. In Definition 4.10.15, the *adaptive zero-knowledge* condition should be quantified only over efficiently computable input-selection strategies. The revised form is presented in Definition 5.4.22.

2. Regarding Constriction 4.10.7 and the proof of Proposition 4.10.9: The current description in terms of two mappings $\pi_1, \pi_2$ is confusing and even inaccurate. Instead one should identify the rows (resp., columns) of $H$ with $[n]$ and use one permutation $\pi$ over $[n]$ (which supposedly maps the vertices of $G$ to those of $H$). Alternatively, one may compose this permutation $\pi$ with the two (1-1) mappings $\psi_i$'s (where $\psi_i : [n] \to [n^3]$), and obtain related $\pi_i$'s (i.e., $\pi_i(v) = \psi_i(\pi(v))$), which should be used as in the original text.

### C.3.2 More on Remark 4.10.6

In continuation to Remark 4.10.6 and following [37], we briefly discuss the issues that arise when wishing to extend the construction to arbitrary trapdoor permutations. Recall that Remark 4.10.6 focuses on a family of trapdoor permutations of the form $\{f_\alpha : \{0,1\}^{|\alpha|} \to \{0,1\}^{|\alpha|}\}_{\alpha\in\overline{I}}$, where $\overline{I}$ is efficiently recognizable. Unfortunately, no such family is known, and thus we first extend the treatment to the case in which $\overline{I}$ is not necessarily efficiently recognizable. The problem we encounter is that the prover may select (and send along) a function that is not in the family (i.e., an $\alpha$ not in $\overline{I}$). In such a case, the function is not necessarily 1-1, and consequently, the soundness property may be violated. This concern can be addressed by using a simple non-interactive (zero-knowledge) proof that the function is "typically 1-1" (or, equivalently, is "almost onto the designated range"). The proof proceeds by presenting inverses (under the function) of random elements specified in the reference string. Note that, for any fixed polynomial $p$, we can only prove that the function is 1-1 on at least a $1-(1/p(n))$ of the designated range, but this suffices for moderate soundness of the entire proof system (which in turn can be amplified by repetitions). For further details, consult [37].

Although the known candidate trapdoor permutations can be modified to fit the above form, we wish to further generalize the result so that any *enhanced* trapdoor permutation (as defined in Definition C.1.1) can be used. This can be done by letting the reference string consist of the coin-sequences used by the domain-sampling algorithm (rather than of elements of the function's domain). By virtue of the enhanced hardness condition (i.e., Eq. (C.3)), the security of the hardcore is preserved, and so is the zero-knowledge property.

As stated at the end of Section C.1, in contrast to what was claimed in Remark 4.10.6, we do not known how to extend the construction to arbitrary (rather than enhanced) trapdoor permutation.

### C.3.3 Additional Comments

1. In continuation to Sections 4.7 and 4.9.2, we mention that the round-efficient argument system of [99] is actually an "argument of knowledge" (with negligible error).

2. We mention that the notions of *strong* witness indistinguishability (Definition 4.6.2) and *strong* proofs of knowledge (Section 4.7.6), and the Hidden Bit Model (Section 4.10.2) have first appeared in early versions of this work.

### C.3.4 Typos etc

1. In the guideline for Exercise 11 of Chapter 2, the term $\mathsf{E}\text{cyc}_f(U_n)]$ should be $\mathsf{E}[\text{cyc}_f(U_n)]$. In the exercise itself, one should also address the case in which $\text{cyc}_f(x)$ is undefined for some $x$'s.

# Index