

A Brief Introduction to Zero-Knowledge (by Oded Goldreich)

Zero-knowledge proofs, introduced by Goldwasser, Micali and Rackoff [72], are fascinating and extremely useful constructs. Their fascinating nature is due to their seemingly contradictory definition; zero-knowledge proofs are both convincing and yet yield nothing beyond the validity of the assertion being proven. Their applicability in the domain of cryptography is vast: Following the results of Goldreich, Micali and Wigderson [65], zero-knowledge proofs are typically used to force malicious parties to behave according to a predetermined protocol. In addition to their direct applicability in cryptography, zero-knowledge proofs serve as a good benchmark for the study of various problems regarding cryptographic protocols (e.g., the “preservation of security under various forms of protocol composition” and the “use of the adversary’s program within the proof of security”).

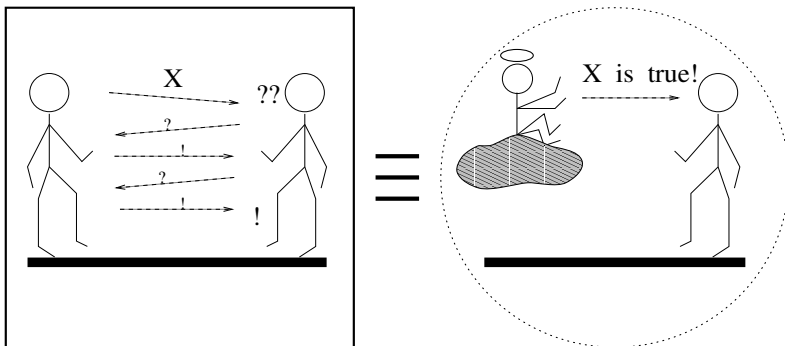


Fig. 1.1. Zero-knowledge proofs – an illustration.

We wish to highlight also the indirect impact of zero-knowledge on the definitional approach underlying the foundations of cryptography (cf. Sect. 1.2.1). In addition, zero-knowledge has served as a source of inspiration for complex-

ity theory. In particular, it served as the main motivation towards the introduction of interactive proof systems [72] and multiprover interactive proof systems [17] (which, in turn, led to the exciting developments regarding PCP and the complexity of approximation [44, 4, 3]).

A Very Brief Summary of this Chapter. 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 (by a so-called simulator). Variants on the basic definition include (see Sect. 1.2.3):

- consideration of auxiliary inputs;
- mandating of universal and black-box simulations;
- restricting attention to honest (or rather semi-honest) verifiers;
- the level of similarity required of the simulation.

It is well known (see Sect. 1.3) that *zero-knowledge proofs exist for any NP-set*, provided that one-way functions exist. This result is a powerful tool in the design of cryptographic protocols, because it can be used to force parties to behave according to a predetermined protocol (i.e., the protocol requires parties to provide zero-knowledge proofs of the correctness of their secret-based actions, without revealing these secrets).

A natural question regarding zero-knowledge proofs is whether the zero-knowledge condition is preserved under a variety of composition operations. Indeed, most of this book is devoted to the study of concurrent composition of zero-knowledge proofs, but more restricted types of composition are also of interest. The main facts regarding composition of zero-knowledge protocols are:

- Zero-knowledge is closed under sequential composition.
- In general, zero-knowledge is not closed under parallel composition. Yet, some zero-knowledge proofs (for NP) preserve their security when many copies are executed in parallel. Furthermore, some of these protocols use a constant number of rounds, and this result extends to restricted forms of concurrent composition (i.e., “bounded asynchronicity”).
- Some zero-knowledge proofs (for NP) preserve their security when many copies are executed concurrently, but such a result is not known for constant-round protocols. Indeed, most of this book is devoted to the study of the round-complexity of concurrent zero-knowledge protocols.

Organization of this Chapter. In this chapter we present the basic definitions and results regarding zero-knowledge protocols. We start with some preliminaries (Sect. 1.1), which are central to the “mind-set” of the notion of zero-knowledge. In particular, we review the definitions of interactive proofs

and arguments as well as the definitions of computational indistinguishability (which underlies the definition of general zero-knowledge) and of one-way functions (which are used in constructions). We then turn to the definitional treatment of zero-knowledge itself, provided in Sect. 1.2. In Sect. 1.3 we review the main feasibility result regarding zero-knowledge proofs and its typical applications. We conclude this chapter with a brief survey of results regarding sequential, parallel and concurrent composition (Sect. 1.4).

Suggestions for Further Reading. A brief account of other developments regarding zero-knowledge protocols is provided in Chap. 9. For further details regarding the material presented in Sect. 1.1–1.3, the reader is referred to [57, Chap. 4]. For a wider perspective on probabilistic proof systems, the reader is referred to [56, Chap. 2].

1.1 Preliminaries

Modern cryptography is concerned with the construction of *efficient* schemes for which it is *infeasible* to violate the security feature. The same concern underlies the main definitions of zero-knowledge. Thus, for starters, 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 to 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 users) we are in the same situation as in any algorithmic setting.

Things are different when referring to our assumptions regarding the computational resources of the adversary. 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 treatment.

Actually, in order to simplify our exposition, we will often consider as **infeasible** any computation that cannot be conducted by a (possibly non-uniform) family of polynomial-size circuits. For simplicity we consider families of circuits $\{C_n\}$, where for some polynomials p and q , each C_n has exactly $p(n)$ input bits and has size at most $q(n)$.

Randomized computations play a central role in the definition of zero-knowledge (as well as in cryptography at large). That is, we allow the legitimate users to employ randomized computations, and likewise we consider

adversaries that employ randomized computations. This brings up the issue of success probability: typically, we require that legitimate users succeed (in fulfilling their legitimate goals) with probability 1 (or negligibly close to this), whereas adversaries succeed (in violating the security features) with negligible probability. Thus, the notion of a **negligible probability** plays an important role in our exposition. One feature required of the definition of *negligible probability* is to yield a robust notion of rareness: A rare event should occur rarely even if we repeat the experiment for a feasible number of times. Likewise, we consider two events to occur “as frequently” if the absolute difference between their corresponding occurrence probabilities is negligible. For concreteness, we consider as **negligible** any function $\mu: N \rightarrow [0, 1]$ that vanishes faster than the reciprocal of any polynomial (i.e., for every positive polynomial p and all sufficiently big n , it holds that $\mu(n) < 1/p(n)$).

1.1.1 Interactive Proofs and Argument Systems

A proof is whatever convinces me.

Shimon Even, answering a student’s question
in his Graph Algorithms class (1978)

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 BPP) [67], 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 by Goldwasser, Micali and Rackoff [72]). 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).

We comment that interactive proofs emerge naturally when associating the notion of efficient verification, which underlies the notion of a proof system, with probabilistic and interactive polynomial-time computations. This association is quite natural in light of the growing acceptability of randomized and distributed computations. Thus, a “proof” in this context is not a fixed and static object, but rather a randomized and dynamic (i.e., interactive) process in which the verifier interacts with the prover. Intuitively, one may think of this interaction as consisting of “tricky” questions asked by the verifier, to which the prover has to reply “convincingly”. The above discussion, as well as the following definition, makes explicit reference to a prover, whereas a prover is only implicit in the traditional definitions of proof systems (e.g., NP-proofs).

Loosely speaking, an interactive proof is a game between a computationally bounded verifier and a computationally unbounded prover whose goal is to convince the verifier of the validity of some assertion. Specifically, the verifier is probabilistic polynomial-time. It is required that if the assertion holds then the verifier always accepts (i.e., when interacting with an appropriate prover strategy). On the other hand, if the assertion is false then the verifier must reject with “noticeable” probability, no matter what strategy is being employed by the prover. Indeed, the error probability (in the soundness condition) can be reduced by (either sequential or parallel) repetitions.

Definition 1.1.1 (Interactive proof systems and the class \mathcal{IP} [72]) *An interactive proof system for a set S is a two-party game, between a verifier executing a probabilistic polynomial-time strategy (denoted V) and a prover which executes a computationally unbounded strategy (denoted P), satisfying:*

- **Completeness:** *For every $x \in S$ the verifier V always accepts after interacting with the prover P on common input x .*
- **Soundness:** *For some polynomial p , it holds that for every $x \notin S$ and every potential strategy P^* , the verifier V rejects with probability at least $1/p(|x|)$, after interacting with P^* on common input x .*

The class of problems having interactive proof systems is denoted \mathcal{IP} .

Note that by repeating such a proof system for $O(p(|x|)^2)$ times, we may decrease the probability that V accepts a false statement (from $1 - (1/p(|x|))$) to $2^{-p(|x|)}$. Thus, when constructing interactive proofs we sometimes focus on obtaining a noticeable rejection probability for NO-instances (i.e., obtaining a soundness error bounded away from 1), whereas when using interactive proofs we typically assume that their soundness error is negligible.

Variants. Arthur–Merlin games (a.k.a. *public-coin* proof systems), introduced by Babai [5], are a special case of interactive proofs in which the verifier must send the outcome of any coin it tosses (and thus need not send any other information). Yet, as shown in [74], this restricted case has essentially the same power as the general case (introduced by Goldwasser, Micali and Rackoff [72]). Thus, in the context of interactive proof systems, *asking random questions is as powerful as asking “tricky” questions*. (As we shall see, this does not necessarily hold in the context of zero-knowledge proofs.) Also, in some sources interactive proofs are defined so that two-sided error probability is allowed (rather than requiring “perfect completeness” as done above); yet, this does not increase their power [52].

Arguments (or Computational Soundness). A fundamental variant on the notion of interactive proofs was introduced by Brassard, Chaum and Crépeau [23], who relaxed the soundness condition so that it only refers to feasible ways of trying to fool the verifier (rather than to all possible ways). Specifically, the soundness condition was replaced by the following **computational soundness** condition that asserts that it is infeasible to fool the verifier into accepting false statements. That is:

For every polynomial p , every prover strategy that is implementable by a family of polynomial-size circuits $\{C_n\}$, and every sufficiently large $x \in \{0, 1\}^ \setminus S$, the probability that V accepts x when interacting with $C_{|x|}$ is less than $1/p(|x|)$.*

We warn that although the computational-soundness error can always be reduced by sequential repetitions, it is not true that this error can always be reduced by parallel repetitions (cf. [15]). Protocols that satisfy the computational-soundness condition are called **arguments**.¹ We mention that argument systems may be more efficient than interactive proofs (see [80, 61]).

Terminology. Whenever we wish to blur the distinction between proofs and arguments, we will use the term protocols. We will consider such a protocol **trivial** if it establishes membership in a BPP-set (because membership in such a set can be determined by the verifier itself). On the other hand, we will sometimes talk about “protocols for \mathcal{NP} ”, when what we actually mean is protocols for each set in \mathcal{NP} . (The latter terminology is quite common in the area; see [10] for further discussion of the distinction.)

1.1.2 Computational Difficulty and One-Way Functions

Most positive results regarding zero-knowledge proofs are based on intractability assumptions. Furthermore, the very notion of a zero-knowledge proof is interesting only in case the assertion being proven to be valid is hard to verify in probabilistic polynomial-time. Thus, our discussion always assumes (at least implicitly) that \mathcal{IP} is not contained in \mathcal{BPP} , and often we explicitly assume more than that.

In general, modern cryptography is concerned with the construction of schemes that 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 at the heart of modern cryptography. However, gaps as required for modern cryptography are not known to exist; they are only widely believed to exist. Indeed, almost all of modern cryptography rises or falls with the question of whether one-way functions exist. One-way functions are functions that are easy to evaluate but hard (on the average) to invert (cf. [37]). 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 (possibly non-uniform) families of polynomial-size circuits, we obtain the following definition.

¹ A related notion not discussed here is that of CS-proofs, introduced by Miceli [86].

Definition 1.1.2 (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 exists a polynomial-time algorithm A such that $A(x) = f(x)$ for every $x \in \{0, 1\}^*$.*
2. Hard to invert: *For every family of polynomial-size circuits $\{C_n\}$, every polynomial p , and all sufficiently large n ,*

$$\Pr[C_n(f(x)) \in f^{-1}(f(x))] < \frac{1}{p(n)}$$

where the probability is taken uniformly over all the possible choices of $x \in \{0, 1\}^n$.

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.

Terminology. Some of the known (positive) results (regarding zero-knowledge) require stronger forms of one-way functions (e.g., one-way permutations with (or without) trapdoor [57, Sect. 2.4.4] and claw-free permutation pairs [57, Sect. 2.4.5]). Whenever we wish to avoid the specific details, we will talk about standard intractability assumptions. In all cases, the conjectured intractability of factoring will suffice.

1.1.3 Computational Indistinguishability

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

The Principle of Identity of Indiscernibles²
G.W. Leibniz (1646–1714)

A central notion in modern cryptography is that of “effective similarity” (introduced by Goldwasser, Micali and Yao [71, 102]). The underlying thesis is that we do not care whether or not objects are equal, all we care is whether or not a difference between the objects can be observed by a feasible computation. In case the answer is negative, the two objects are equivalent as far as any practical application is concerned. Indeed, like in many other cryptographic definitions, in the definition of general/computational zero-knowledge we will freely interchange such (computationally indistinguishable) objects.

The asymptotic formulation of computational indistinguishability refers to (pairs of) probability ensembles, which are infinite sequences of finite distributions, rather than to (pairs of) finite distributions. Specifically, we consider

²Leibniz admits that counterexamples to this principle are conceivable but will not occur in real life because God is much too benevolent.

sequences indexed by strings, rather than by integers (in unary representation). For $S \subseteq \{0, 1\}^*$, we consider the probability ensembles $X = \{X_\alpha\}_{\alpha \in S}$ and $Y = \{Y_\alpha\}_{\alpha \in S}$, where each X_α (resp., Y_α) is a distribution that ranges over strings of length polynomial in $|\alpha|$. We say that X and Y are **computationally indistinguishable** if for every feasible algorithm A the difference $d_A(n) \stackrel{\text{def}}{=} \max_{\alpha \in \{0,1\}^n} \{|\Pr[A(X_\alpha) = 1] - \Pr[A(Y_\alpha) = 1]|\}$ is a negligible function in $|\alpha|$. That is:

Definition 1.1.3 (Computational indistinguishability [71, 102]) *We say that $X = \{X_\alpha\}_{\alpha \in S}$ and $Y = \{Y_\alpha\}_{\alpha \in S}$ are computationally indistinguishable if for every family of polynomial-size circuits $\{D_n\}$, every polynomial p , all sufficiently large n and every $\alpha \in \{0, 1\}^{\text{poly}(n)} \cap S$,*

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

where probabilities are taken over the relevant distribution (either X_n or Y_n).

That is, we think of $D = \{D_n\}$ 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 indistinguishability by a single sample (as defined above) implies indistinguishability by multiple samples. Also note that the definition would not have been stronger if we were to provide the distinguisher (i.e., D) with the index (i.e., α) of the distribution pair being tested.³

1.2 Definitional Issues

A: Please.
 B: Please.
 A: I insist.
 B: So do I.
 A: OK then, thank you.
 B: You are most welcome.

A protocol for two Italians to pass through a door.⁴

Source: Silvio Micali, 1985.

³ Furthermore, the definition would not have been stronger if we were to consider a specialized polynomial-size circuit for each $\alpha \in S$ (i.e., consider the difference $|\Pr[D_\alpha(X_\alpha) = 1] - \Pr[D_\alpha(Y_\alpha) = 1]|$ for any set of circuits $D = \{D_\alpha\}_{\alpha \in S}$ such that the size of D_α is polynomial in $|\alpha|$).

⁴The protocol is zero-knowledge because it can be simulated without knowing any of the secrets of these Italians; in fact, the execution is independent of their secrets as well as of anything else.

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 can be feasibly obtained 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.

1.2.1 The Simulation Paradigm

In defining zero-knowledge proofs, we view the verifier as a potential adversary that tries to gain knowledge from the (prescribed) prover. We wish to state that no (feasible) adversary strategy for the verifier can gain anything from the prover (beyond conviction in the validity of the assertion). Let us consider the desired formulation from a wide perspective.

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 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, while assuming that the latter is valid). Thus, in a zero-knowledge proof no feasible adversarial strategy for the verifier can obtain more than a “benign verifier”, which believes the assertion, can obtain from the assertion itself. We comment that the simulation paradigm, which was first developed in the context of zero-knowledge [72], is pivotal also to the definition of the security of encryption schemes (cf. [58, Chap. 5]) and cryptographic protocols (cf. [26] and [58, Chap. 7]).

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

respect to a specific application, it is typically very hard to classify the set of “harmful breakings”.

1.2.2 The Basic Definition

Zero-knowledge is a property of some prover strategies. More generally, zero-knowledge is a property of some interactive machines. Fixing an interactive machine (e.g., a prescribed prover), we consider what can be computed by an *arbitrary* feasible adversary (e.g., a verifier) that interacts with the fixed machine on a common input taken from a predetermined set (in our case the set of valid assertions). This is compared against what can be computed by an *arbitrary* feasible algorithm that is only given the input itself. An interactive strategy A is **zero-knowledge** on (inputs from) the set S if, for every feasible (interactive) strategy B^* , there exists a feasible (non-interactive) computation C^* such that the following two probability ensembles are computationally indistinguishable:

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

We stress that the first ensemble represents an actual execution of an interactive protocol, whereas the second ensemble represents the computation of a stand-alone procedure (called the “simulator”), which does not interact with anybody. Thus, whatever can be feasibly extracted from interaction with A on input $x \in S$ can also be feasibly extracted from x itself. This means that nothing was gained by the interaction itself (beyond confidence in the assertion $x \in S$).

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 [63, 67]). This is taken care of by a more strict notion called **auxiliary-input zero-knowledge**.⁵

Definition 1.2.1 (Zero-knowledge [72], revisited [67]) *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:*

⁵ We note that Definition 1.2.1 seems stronger than merely allowing the verifier and simulator to be arbitrary polynomial-size circuits. The issue is that the latter formulation does not guarantee that the simulator can be easily derived from the cheating verifier nor that the length of the simulator’s description is related to the length of the description of the verifier. Both issues are important when trying to use zero-knowledge proofs as subprotocols inside larger protocols or to compose them (even sequentially). For further discussion, see Sect. 1.4.

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

An interactive proof (resp., an argument) system for S is called **auxiliary-input zero-knowledge** if the prescribed prover strategy is auxiliary-input zero-knowledge on inputs from S .⁶

The more basic definition of zero-knowledge is obtained by eliminating the auxiliary-input z from Definition 1.2.1. We comment that almost all known zero-knowledge proofs are in fact auxiliary-input zero-knowledge. (Notable exceptions are zero-knowledge proofs constructed on purpose in order to show a separation between these two notions (e.g., in [63]) and protocols having only “non-black-box simulators” (see warm-up in [8]).) As hinted above, *auxiliary-input zero-knowledge is preserved under sequential composition* [67].

We stress that the zero-knowledge property of an interactive proof (resp., argument) refers to all feasible *adversarial strategies that the verifier may employ* (in an attempt to extract knowledge from the prescribed prover that tries to convince the verifier to accept a valid assertion). In contrast, the soundness property of an interactive proof (resp., the computational-soundness property of an argument) refers to all possible (resp., feasible) *adversarial strategies that the prover may employ* (in an attempt to fool the prescribed verifier to accept a false assertion). Finally, the completeness property (only) refers to the behavior of both prescribed strategies (when given, as common input, a valid assertion).

1.2.3 Variants

The reader may skip the current subsection and return to it whenever encountering (especially in Chap. 9) a notion that was not defined above.

Universal and Black-Box Simulation

We have already discussed two variants of the basic definition (i.e., with or without auxiliary-inputs). Further strengthening of Definition 1.2.1 is obtained by requiring the existence of a **universal simulator**, denoted C , that is given the program of the verifier (i.e., B^*) as an auxiliary-input; that is, in terms of Definition 1.2.1, one should replace $C^*(x, z)$ by $C(x, z, \langle B^* \rangle)$, where

⁶ Note that the *prescribed* verifier strategy (which is a probabilistic polynomial-time strategy that only depends on the common input) is always auxiliary-input zero-knowledge. In contrast, typical prover strategies are implemented by probabilistic polynomial-time algorithms that are given an auxiliary input (which is not given to the verifier), but not by probabilistic polynomial-time algorithms that are only given the common input.

$\langle B^* \rangle$ denotes the description of the program of B^* (which may depend on x and on z).⁷ That is, we effectively restrict the simulation by requiring that it be a uniform (feasible) function of the verifier’s program (rather than arbitrarily depend on it). This restriction is very natural, because it seems hard to envision an alternative way of establishing the zero-knowledge property of a given protocol.

Taking another step, one may argue that since it seems infeasible to reverse-engineer programs, the simulator may as well just use the verifier strategy as an oracle (or as a “black-box”). This reasoning gave rise to the notion of **black-box simulation**, which was introduced and advocated in [62] and further studied in numerous works (see, e.g., [30]). The belief was that impossibility results regarding black-box simulation represent inherent limitations of zero-knowledge itself. However, this belief has been refuted recently by Barak [8]. For further discussion, see Sect. 9.1.

Knowledge Tightness

Intuitively, knowledge tightness is a refinement of zero-knowledge that is aimed at measuring the “actual security” of the proof system; namely, how much harder does the verifier need to work, when not interacting with the prover, in order to compute something that it can compute after interacting with the prover. Thus, knowledge tightness is the ratio between the running time of the simulator and the running time of the verifier in the real interaction simulated by the simulator. (For more details, see [57, Sect. 4.4.4.2].)

Note that black-box simulators guarantee that the underlying zero-knowledge protocol has knowledge tightness that is bounded by some fixed polynomial. In fact, in some cases, the knowledge tightness can be bounded by a constant (e.g., 2). In contrast, the general definition of zero-knowledge (i.e., Definition 1.2.1) does not guarantee that the knowledge tightness can be bounded by some fixed polynomial. In fact, the non-black-box simulators of Barak [8] seem to have a running time that is polynomially (but not linearly) related to the running time of the verifier that they simulate.

Honest Verifier Versus General Cheating Verifier

The (general) definition of zero-knowledge (i.e., Definition 1.2.1) refers to all feasible verifier strategies. This choice is most natural since zero-knowledge is supposed to capture the robustness of the prover under *any feasible* (i.e., adversarial) attempt to gain something by interacting with it. Thus, we typically view the verifier as an adversary that is trying to cheat.

⁷ Actually, we may incorporate x and z in $\langle B^* \rangle$, and thus replace $C(x, z, \langle B^* \rangle)$ by $C(\langle B^* \rangle)$.

A weaker and still interesting notion of zero-knowledge refers to what can be gained by an “honest verifier” (or rather a semi-honest verifier)⁸ that interacts with the prover as directed, with the exception that it may maintain (and output) a record of the entire interaction (i.e., even if directed to erase all records of the interaction). Although such a weaker notion is not satisfactory for standard cryptographic applications, it yields a fascinating notion from a conceptual as well as a complexity-theoretic point of view. Furthermore, as shown in [68], every public-coin proof system that is *zero-knowledge with respect to the honest-verifier* can be transformed into a *standard zero-knowledge* proof that maintains many of the properties of the original protocol (and without increasing the prover’s powers or using any intractability assumptions).

We stress that the definition of *zero-knowledge with respect to the honest-verifier* V is derived from Definition 1.2.1 by considering a single verifier strategy B that is equal to V except that B also maintains a record of the entire interaction (including its own coin tosses) and outputs this record at the end of the interaction. (In particular, the messages sent by B are identical to the corresponding messages that would have been sent by V .)

Statistical Versus Computational Zero-Knowledge

Recall that the definition of zero-knowledge postulates that for every probability ensemble of one type (i.e., representing the verifier’s output after interaction with the prover) there exists a “similar” ensemble of a second type (i.e., representing the simulator’s output). One key parameter is the interpretation of “similarity”. Three interpretations, yielding different notions of zero-knowledge, have been commonly considered in the literature (cf., [72, 50]):

1. **Perfect zero-knowledge (PZK)** requires that the two probability ensembles be identical.⁹
2. **Statistical zero-knowledge (SZK)** requires that these probability ensembles be statistically close (i.e., the variation distance between them is negligible).
3. **Computational (or rather general) zero-knowledge (CZK)** requires that these probability ensembles be computationally indistinguishable.

Indeed, computational zero-knowledge (CZK) is the most liberal notion, and is the notion considered in Definition 1.2.1 as well as in most of this book. (In particular, whenever we fail to qualify the type of zero-knowledge, we mean

⁸ The term “honest verifier” is more appealing when considering an alternative (equivalent) formulation of Definition 1.2.1. In the alternative definition, the simulator is “only” required to generate the verifier’s view of the real interaction, when the verifier’s view includes its inputs, the outcome of its coin tosses, and all messages it has received.

⁹ The actual definition of PZK allows the simulator to fail (while outputting a special symbol) with some probability that is bounded away from 1, and the output distribution of the simulator is conditioned on its not failing.

computational zero-knowledge.) The only exception is Sect. 9.5, which is devoted to a discussion of statistical (or almost-perfect) Zero-Knowledge (SZK). We note that the class SZK contains several problems that are considered intractable.

Strict Versus Expected Probabilistic Polynomial-Time

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

1. **Strict probabilistic polynomial-time.** That is, there exists a (polynomial in the length of the input) bound on the *number of steps in each possible run* of the machine, regardless of the outcome of its coin tosses.
2. **Expected probabilistic polynomial-time.** The standard approach is to look at the running time as a random variable and *bound its expectation* (by a polynomial in the length of the input). As observed by Levin [84] (cf. [54]), this definitional approach is quite problematic (e.g., it is not model-independent and is not closed under algorithmic composition), and an alternative treatment of this random variable is preferable.¹⁰

The notion of expected polynomial-time raises a variety of conceptual and technical problems. For that reason, whenever possible, one should prefer to use the more robust (and restricted) notion of strict (probabilistic) polynomial-time. Thus, with the *exception of constant-round* zero-knowledge protocols, whenever we talk of a probabilistic polynomial-time verifier (resp., simulator) we mean one in the strict sense. In contrast, with the exception of [8, 12],¹¹ all results regarding *constant-round* zero-knowledge protocols refer to a strict polynomial-time verifier and an expected polynomial-time simulator, which is indeed a small cheat. For further discussion, the reader is referred to [12].

¹⁰ Specifically, it is preferable to define expected polynomial-time as having running time that is polynomially related to a function that has linear expectation. That is, rather than requiring that $e[X_n] = \text{poly}(n)$, one requires that for some Y_n it holds that $X_n = \text{poly}(Y_n)$ and $e[Y_n] = O(n)$. The advantage of the latter approach is that if X_n is deemed *polynomial on the average* then so is X_n^2 , which is not the case under the former approach (e.g., $X_n = 2^n$ with probability 2^{-n} and $X_n = n$ otherwise).

¹¹ Specifically, in [8, 12] both the verifier and the simulator run in strict polynomial-time. We comment that, as shown in [12], the use of non-black-box is necessary for the non-triviality of constant-round zero-knowledge protocols under the strict definition.

1.3 Zero-Knowledge Proofs for Every NP-set

A question avoided so far is whether zero-knowledge proofs exist at all. Clearly, every set in \mathcal{P} (or rather in \mathcal{BPP})¹² has a “trivial” zero-knowledge proof (in which the verifier determines membership by itself); however, what we seek is zero-knowledge proofs for statements that the verifier cannot decide by itself.

1.3.1 Constructing Zero-Knowledge Proofs for NP-sets

Assuming the existence of commitment schemes,¹³ which in turn exist if one-way functions exist [87, 77], *there exist* (auxiliary-input) *zero-knowledge proofs of membership in any NP-set* (i.e., sets having efficiently verifiable static proofs of membership). These zero-knowledge proofs, first constructed by Goldreich, Micali and Wigderson [65] (and depicted in Figure 1.2), 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). That is:

Theorem 1.1 ([65], using [77, 87]) *If one-way functions exist then every set $S \in \mathcal{NP}$ has a zero-knowledge interactive proof. Furthermore, the prescribed prover strategy can be implemented in probabilistic polynomial-time, provided it is given as auxiliary-input an NP-witness for membership of the common input in S .*

Theorem 1.1 makes zero-knowledge a very powerful tool in the design of cryptographic schemes and protocols (see below). We comment that the intractability assumption used in Theorem 1.1 seems essential; see [91].

Analyzing the Protocol of Fig. 1.2. Let us consider a single execution of the main loop (and rely on the fact that zero-knowledge is preserved under sequential composition). Clearly, the prescribed prover is implemented in probabilistic polynomial-time, and always convinces the verifier (provided that it is given a valid 3-coloring of the common input graph). In case the graph is not 3-colorable then, no matter how the prover behaves, the verifier will reject with probability at least $1/|E|$ (because at least one of the edges must be improperly colored by the prover). We stress that the verifier selects uniformly which edge to inspect after the prover has committed to the colors of all vertices. Thus, Fig. 1.2 depicts an interactive proof system for Graph 3-colorability. As can be expected, the zero-knowledge property is the hardest

¹² Trivial zero-knowledge proofs for sets in $\mathcal{BPP} \setminus \text{coRP}$ require modifying the definition of interactive proofs so as to allow a negligible error also in the completeness condition. Alternatively, zero-knowledge proofs for sets in \mathcal{BPP} can be constructed by having the prover send a single message that is distributed almost uniformly (cf. [52]).

¹³ Loosely speaking, commitment schemes are digital analogues of non-transparent sealed envelopes. See further discussion in Fig. 1.2.

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 the receiver to read the uniquely committed value (as when sending the key to the lock).

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

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

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

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

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

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

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

Fig. 1.2. The zero-knowledge proof of graph 3-colorability (of [65]). Zero-knowledge proofs for other NP-sets can be obtained using the standard reductions.

to establish, and we will confine ourselves to presenting a simulator (which we hope will convince the reader without a detailed analysis). We start with three simplifying conventions (which are useful in general):

1. Without loss of generality, we may assume that the cheating verifier strategy is implemented by a *deterministic* polynomial-size circuit (or, equivalently, by a polynomial-time algorithm with an auxiliary input). This is justified by fixing any outcome of the verifier's coins, and observing that our (uniform) simulation of the various (residual) deterministic strategies yields a simulation of the original probabilistic strategy.
2. Without loss of generality, it suffices to consider cheating verifiers that (only) output their view of the interaction (i.e., their input, coin tosses, and the messages they received). This is justified by observing that the output of the original verifier can be computed by an algorithm of comparable complexity that is given the verifier's view of the interaction. Thus, it suffices to simulate the view that cheating verifiers have of the real interaction.
3. Without loss of generality, it suffices to construct a "weak simulator" that produces output with some noticeable probability. This is the case because, by repeatedly invoking this weak simulator (polynomially) many

times, we may obtain a simulator that fails to produce an output with negligible probability, whereas the latter yields a simulator that never fails (as required).

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

Zero-Knowledge Proofs for Other NP-Sets. By using the standard Karp-reductions to 3-colorability, the protocol of Fig. 1.2 can be used for constructing zero-knowledge proofs for any set in \mathcal{NP} . We comment that this is probably the first time that an NP-completeness result was used in a “positive” way (i.e., in order to construct something rather than in order to derive a hardness result). Subsequent positive uses of completeness results have appeared in the context of interactive proofs [85, 100], probabilistically checkable proofs [6, 44, 4, 3], “hardness versus randomness trade-offs” [7], and statistical zero-knowledge [99].

Efficiency Considerations. The protocol in Fig. 1.2 calls for invoking some constant-round protocol for a non-constant number of times. At first glance, it seems that one can derive a constant-round zero-knowledge proof system (of negligible soundness error) by performing these invocations in parallel (rather than sequentially). Unfortunately, as demonstrated in [62], this intuition is not sound. See further discussions in Sects. 1.4 and 9.1. We comment that the number of rounds in a protocol is commonly considered the most important efficiency criterion (or complexity measure), and typically one desires to have it be a constant. We mention that, under standard intractability assumptions (e.g., the intractability of factoring), constant-round zero-knowledge proofs (of negligible soundness error) exist for every set in \mathcal{NP} (cf. [62]).

1.3.2 Using Zero-Knowledge Proofs for NP-sets

We stress two important aspects regarding Theorem 1.1. Firstly, it provides a zero-knowledge proof for every NP-set, and secondly the prescribed prover

can be implemented in probabilistic polynomial-time when given an adequate NP-witness. These properties are essential to the wide applicability of zero-knowledge protocols.

A Generic Application. In a typical cryptographic setting, a user referred to as U has a secret and is supposed to take some action depending on its secret. The question is how can other users verify that U indeed took the correct action (as determined by U 's secret and the publicly known information). Indeed, if U discloses its secret then anybody can verify that U took the correct action. However, U does not want to reveal its secret. Using zero-knowledge proofs we can satisfy both conflicting requirements (i.e., having other users verify that U took the correct action without violating U 's interest in not revealing its secrets). That is, U can prove in zero-knowledge that it took the correct action. Note that U 's claim to having taken the correct action is an NP-assertion (since U 's legal action is determined as a polynomial-time function of its secret and the public information), and that U has an NP-witness to its validity (i.e., the secret is an NP-witness to the claim that the action fits the public information). Thus, by Theorem 1.1, it is possible for U to efficiently prove the correctness of its action without yielding anything about its secret. Consequently, it is fair to ask U to prove (in zero-knowledge) that it behaves properly, and so to force U to behave properly. Indeed, “forcing proper behavior” is the canonical application of zero-knowledge proofs (see [66, 55]).

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

Zero-Knowledge Proofs for All IP. We mention that under the same assumption used in the case of \mathcal{NP} , it holds that *any set that has an interactive proof also has a zero-knowledge interactive proof* (cf. [78, 16]).

1.4 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 for its own sake, but rather also sheds light on 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 possible but quite difficult. 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 to the extra trouble of coordinating its attack in the various executions of the protocol.

For $T \in \{\text{sequential}, \text{parallel}, \text{concurrent}\}$, 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 Definition 1.2.1 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 .¹⁴ The corresponding simulator (which, as usual, interacts with nobody) is required to produce an output that is computationally indistinguishable from the output of such a type T adversary.

1.4.1 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).

Referring to Definition 1.2.1, we mention that whereas the “simplified” version (i.e., without auxiliary inputs) is not closed under sequential composition (cf. [63]), the actual version (i.e., with auxiliary inputs) is closed under sequential composition (cf. [67]). We comment that the same phenomena arises when trying to use a zero-knowledge proof as a subprotocol inside larger

¹⁴ 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.

protocols. Indeed, it is for these reasons that the augmentation of the “most basic” definition by auxiliary inputs was adopted in all subsequent works.¹⁵

Bottom-Line. Every protocol that is zero-knowledge (under Definition 1.2.1) is sequential-zero-knowledge.

1.4.2 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 Sect. 1.4.3.)

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 Fig 1.3. This counter-example, which is adapted from [63], 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, in the 1980s, the study of parallel composition was interpreted mainly in the context of *round-efficient error reduction* (cf. [47, 63]); 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. [62, 48, 25]), 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 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 exist zero-knowledge protocols for \mathcal{NP} that are closed under parallel composition. Furthermore, these protocols have a constant number of rounds (cf. [59] for proofs and [39] for arguments).¹⁶ Both results extend also

¹⁵ Interestingly, the preliminary version of Goldwasser, Micali and Rackoff’s work [72] 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 and means the augmented definition.

¹⁶ 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, which were therefore defined to have negligible soundness error.

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 (β, γ) . 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 as defined in Definition 3.3.1): Intuitively, if the adversary A chooses the case $v = 1$, then it is infeasible for A to guess a passing pair (β, γ) with respect to the random α selected by P . Thus, except with negligible probability (when it may get secret information), A does not obtain anything from the interaction. On the other hand, if the adversary A chooses the case $v = 2$, then it obtains a pair that is indistinguishable from a uniformly selected pair of n -bit long strings (because β is selected uniformly by P , and for any α the value $f(\alpha\beta)$ looks random to A).

In contrast, if the adversary A can conduct two concurrent^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 α , in the first session, A sends α 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.

^aDummy messages may be added (in both cases) in order to make the above scheduling fit the perfectly parallel case.

Fig. 1.3. A counter-example (adapted from [62]) to the parallel repetition conjecture for zero-knowledge protocols.

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. [15]), but our focus here is on the zero-knowledge aspect of protocols regardless of whether they are proofs, arguments or neither.

Bottom-Line. Under standard intractability assumptions, every NP-set has a constant-round parallel-zero-knowledge proof.

1.4.3 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 1990s, when extensive two-party (and multiparty) 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. [38]). In the context of zero-knowledge, concurrent composition was first considered by Dwork, Naor and Sahai [39]. 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. However, 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 [94]. Following their work, research has focused on determining the round-complexity of concurrent zero-knowledge proofs for \mathcal{NP} . This question is still open, 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. [93], building upon [82], which in turn builds over [94]). Furthermore, the zero-knowledge property can be demonstrated using a black-box simulator (see definition in Sects. 1.2.3 and 9.1). This result is presented in Chap. 5.
- Black-box simulators cannot demonstrate the concurrent zero-knowledge property of non-trivial proofs (or arguments) having significantly less than logarithmically many rounds (cf. Canetti et al. [30]).¹⁷ This result is presented in Chap. 7.
- Recently, Barak [8] 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

¹⁷ By *non-trivial* proof systems we mean ones for languages outside \mathcal{BPP} , whereas by *significantly less than logarithmic* we mean any function $f : N \rightarrow N$ satisfying $f(n) = o(\log n / \log \log n)$. In contrast, by *almost-logarithmic* we mean any function f satisfying $f(n) = \omega(\log n)$.

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* protocols for \mathcal{NP} may be concurrent zero-knowledge (in the pure asynchronous model).

We comment that the result of Canetti et al. [30] was proven at a time when it was (falsely) believed that limitations concerning “black-box simulators” are inherent to zero-knowledge itself. This belief turned out to be wrong; see Sect. 9.1 for further discussion. Still black-box simulators are the natural way to demonstrate the zero-knowledge feature of protocols, and it is still important to determine the limits of “black-box” techniques. One reason is that asserting that some problem cannot be solved using “black-box” techniques means that, even in case it is solvable (by “non-black-box” techniques), this problem is inherently harder than others that can be solved using “black-box” techniques. Indeed, solutions that rely on “non-black-box” techniques tend to be more complex not only from a conceptual perspective but also in terms of the time and communication complexities of the resulting protocol. Furthermore, the latter tend to provide a lower level of security (e.g., in terms of knowledge tightness, as discussed in Sect. 1.2.3).

Concurrent Composition Under the Timing Model. A model of naturally-limited asynchronism (which certainly covers the case of parallel composition) was introduced by Dwork, Naor and Sahai [39]. 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** outgoing messages). The benefit of the timing model is that it is known to construct concurrent zero-knowledge protocols in it. Specifically, using standard intractability assumptions, *constant-round* arguments and proofs that are concurrent zero-knowledge under the timing model do exist (cf. [39] and [59], 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.¹⁸ Furthermore, it is postulated that the parties know a (pessimistic) bound, denoted Δ , 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

¹⁸ The rate should be computed with respect to reasonable intervals of time; for example, for Δ as defined below, one may assume that a time period of Δ units is measured as Δ' units of time on the local clock, where $\Delta/\rho \leq \Delta' \leq \rho\Delta$.

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 Δ (i.e., the time-out period must be at least as long as the pessimistic bound on message-delivery time as so not to disrupt the proper operation of the protocol). This means that the use of these time-driven operations yields a slowing down of 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 the 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 that other alternatives to the timing-model include various set-up assumptions; cf. [28, 34].)

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 exists a natural 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 Sect. 9.6, implies concurrent zero-knowledge.



<http://www.springer.com/978-3-540-32938-1>

Concurrent Zero-Knowledge
With Additional Background by Oded Goldreich
Rosen, A.
2006, XIV, 184 p., Hardcover
ISBN: 978-3-540-32938-1