the ensemble $\{G'(U_n)\}_{n\in\mathbb{N}}$ is pseudorandom, as defined in Definition 3.2.8. (If $G'$ is a pseudorandom generator, then it satisfies both conditions, but the converse is not true.)

Prove that the generalized function ensemble $\{f_s : \{0,1\}^{d(|s|)} \to \{0,1\}^{r(|s|)}\}_{s\in\{0,1\}^*}$, defined by $f_s(x) \stackrel{\text{def}}{=} G'(f'_s(x))$, is pseudorandom.

**Guideline:** See proof of Theorem 3.6.11.

**Exercise 30:** *Speeding up pseudorandom function constructions* (suggested by Leonid Levin): For some $d, r : \mathbb{N} \to \mathbb{N}$, consider a generalized pseudorandom function ensemble

$$F \stackrel{\text{def}}{=} \{f_s : \{0,1\}^{d(|s|)} \to \{0,1\}^{r(|s|)}\}_{s\in\{0,1\}^*}$$

as in Definition 3.6.9. Let $\text{Primes}_m$ denote the set of primes in the interval $(2^{m-1}, 2^m)$. For any $d' : \mathbb{N} \to \mathbb{N}$, consider a new function ensemble,

$$F' \stackrel{\text{def}}{=} \{f'_{s,p} : \{0,1\}^{d'(|s|)} \to \{0,1\}^{r(|s|)}\}_{s\in\{0,1\}^*, p\in\text{Primes}_{d(|s|)}}$$

such that $f'_{s,p}(x) \stackrel{\text{def}}{=} f_s(x \bmod p)$, where $\{0,1\}^{d'(|s|)}$ and $\{0,1\}^{d(|s|)}$ are associated with $\{0,\ldots,2^{d'(|s|)}-1\}$ and $\{0,\ldots,2^{d(|s|)}-1\}$, respectively.

The point is that the functions in $F'$ are computable in time related to the time-complexity of $F$. Whenever $d'(n) \gg d(n)$ (e.g., $d'(n) = n^2$ and $d(n) = \log_n^2 n$), this yields a speedup in the time-complexity of $F'$ (when compared with Construction 3.6.10).

**1.** Prove that if $d(n) = \omega(\log n)$, then $F'$ is pseudorandom.

**2.** Show that, on the other hand, if $d(n) = O(\log n)$ (and $d'(n) > d(n)$), then $F'$ is not pseudorandom.

Note that, in general, the "pseudorandomness" of $F'$ (as quantified with respect to the running time sufficient to see evidence that $F'$ is not random) depends on $d : \mathbb{N} \to \mathbb{N}$. Specifically, evidence that $F'$ is not random can be found in time exponential in $d$.

**Guideline (Part 2):** Going over all possible $p$'s, try to gather evidence that the target function indeed uses reduction modulo $p$. (Hint: For fixed $p$, any two distinct $x, y \in \{0,1\}^{d'(|s|)}$ such that $x \equiv y \pmod p$ yield such evidence.)

**Guideline (Part 1):** Consider applying the foregoing construction to the uniform function ensemble $H$, rather than to the pseudorandom ensemble $F$. The main issue is to show that the resulting ensemble $H'$ is pseudorandom. ($F'$ is indistinguishable from $H'$, or else we can distinguish $F$ from $H$.)

**Guideline (Part 1, extra hints):** We refer to the function ensemble $H' = \{H'_n\}_{n\in\mathbb{N}}$, where $H'_n$ is defined by uniformly selecting a function $h : \{0,1\}^{d(n)} \to \{0,1\}^{r(n)}$ and $p \in \text{Prime}_{d(n)}$ and letting $H'_n = h'_p$ such that $h'_p(x) = h(x \bmod p)$. If the distinct queries $x_1, \ldots, x_t \in \{0,1\}^{d'(n)}$ have distinct residues mod $p$, then the answers obtained from $h'_p$ are independently and uniformly distributed in $\{0,1\}^{r(n)}$. Thus, essentially, we need to lower-bound the probability of the former event for a uniformly selected $p \in \text{Prime}_{d(n)}$. We upper-bound the probability of the complementary event (i.e., $\exists i \neq j$ s.t. $x_i \equiv x_j \pmod p$). For distinct $x, y \in \{0,1\}^{d'(n)}$, it holds that $x \equiv y \pmod p$ iff $p$ divides $x - y$. At this stage the argument is simplified by the fact that $p$ is prime:[12] The probability that a uniformly

[12] What if the construction were to be modified so that $p$ was uniformly selected among all integers in $\{2^{d(n)-1}, \ldots, 2^{d(n)}-1\}$?

chosen $d(n)$-bit-long prime divides a $d'(n)$-bit-long integer is at most $\frac{d'(n)/d(n)}{|\mathrm{Prime}_{d(n)}|}$, which is $\Theta(d'(n) \cdot 2^{-d(n)})$.

**Exercise 31:** *An alternative construction for Exercise 30*: Let $F$ and let $d'$ be as in Exercise 30, and let $S^{d(n)}_{d'(n)}$ be a hashing family (as defined in Section 3.5.1.1). For every $s \in \{0,1\}^*$ and $h \in S^{d(|s|)}_{d'(|s|)}$, define $f'_{s,h}$ such that $f'_{s,h}(x) = f_s(h(x))$, and let

$$F' \stackrel{\text{def}}{=} \{ f'_{s,h} : \{0,1\}^{d'(|s|)} \to \{0,1\}^{r(|s|)} \}_{s \in \{0,1\}^*, h \in S^{d(|s|)}_{d'(|s|)}}$$

(This construction requires longer seeds than the one in Exercise 30; however, one can use much smaller families of functions that approximate the desired features.)

1. Prove that if $d(n) = \omega(\log n)$, then $F'$ is pseudorandom.
2. On the other hand, show that if $d(n) = O(\log n)$ and $r(n) > d(n)$, then $F'$ is not pseudorandom.

   **Guideline (Part 2):** For any distinct $x, y \in \{0,1\}^{d'(n)}$ and a uniformly selected function mapping $d'(n)$-bit-long strings to $r(n)$-bit-long string, the probability that $x$ and $y$ are mapped to the same image is $2^{-r(n)}$. However, the probability that $x$ and $y$ are mapped to the same image under a uniformly selected $f'_{s,h}$ is lower-bounded by $\Pr[h(x) = h(y)] = 2^{-d(n)}$.

**Exercise 32:** *An alternative definition of pseudorandom functions*: For the sake of simplicity, this exercise is stated in terms of ensembles of Boolean functions (analogously to Definition 3.6.9, with $d(n) = n$ and $r(n) = 1$). That is, we consider a Boolean-function ensemble $\{ f_s : \{0,1\}^{|s|} \to \{0,1\} \}_{s \in \{0,1\}^*}$ and let $F_n$ be uniformly distributed over the multi-set $\{ f_s \}_{s \in \{0,1\}^n}$. We say that the function ensemble $\{F_n\}_{n \in \mathbb{N}}$ is *unpredictable* if for every probabilistic polynomial-time oracle machine $M$, for every polynomial $p(\cdot)$, and for all sufficiently large $n$'s,

$$\Pr\left[\mathrm{corr}^{F_n}\left(M^{F_n}(1^n)\right)\right] < \frac{1}{2} + \frac{1}{p(n)}$$

where $M^{F_n}(1^n)$ assumes values of the form $(x, \sigma) \in \{0,1\}^n \times \{0,1\}$ such that $x$ is *not* a query appearing in the computation $M^{F_n}(1^n)$, and $\mathrm{corr}^f(x, \sigma)$ is defined as the predicate "$f(x) = \sigma$". Intuitively, after getting the values of $f$ on points of its choice, the machine $M$ outputs a new point (i.e., $x$) along with a guess (i.e., $\sigma$) for the value of $f$ on this point. The value of $\mathrm{corr}^f(x, \sigma)$ represents whether or not $M$ is correct in its guess.

Assuming that $F = \{F_n\}_{n \in \mathbb{N}}$ is efficiently computable, prove that $F$ is pseudorandom if and only if $F$ is unpredictable.

**Guideline:** The proof is analogous to the proof of Theorem 3.3.7

**Exercise 33:** *A mistaken "alternative" definition of pseudorandom functions*: Again, we consider ensembles of Boolean functions, as in Exercise 32. Consider the following definition of *weak unpredictability* of function ensembles. The predicting oracle machine $M$ is given a uniformly chosen $x \in \{0,1\}^n$ as input and should output a guess for $f(x)$, after querying the oracle $f$ on polynomially many other (than $x$) points of its choice. We require that for every probabilistic polynomial-time oracle machine $M$ that *does not* query

the oracle on its own input, for every polynomial $p(\cdot)$, and for all sufficiently large $n$'s,

$$\Pr\left[M^{F_n}(U_n) = F_n(U_n)\right] < \frac{1}{2} + \frac{1}{p(n)}$$

That is, unlike the formulation of Exercise 32, the predicting machine cannot select the point for which it has to predict the value of the function (but rather this point is random and is given as input).

1. Show that any pseudorandom function ensemble is weakly unpredictable.
2. Assuming that pseudorandom function ensembles exist, show that there exists a function ensemble that is weakly unpredictable, although it is not pseudorandom.

This exercise contradicts a flawed claim (which appeared in earlier versions of this manuscript). The flaw was pointed out by Omer Reingold.

**Guideline:** For Part 1, show that unpredictability, as defined in Exercise 32, implies weak unpredictability. Alternatively, provide a direct proof (as in Exercise 32). For Part 2, modify a pseudorandom function ensemble so that each $f$ in the range of $F_n$ satisfies $f(0^n) = 0$.

**Exercise 34:** *An unsuccessful attempt to strengthen the notion of weak unpredictability of function ensembles so that it is equivalent to pseudorandomness of functions*: In continuation of Exercise 33, suppose that we strengthen the requirement by allowing the input $x$ to be chosen from any polynomial-time-constructible ensemble. Namely, here we say that a function ensemble $F = \{F_n\}_{n\in\mathbb{N}}$ is *weakly2 unpredictable* if for every probabilistic polynomial-time oracle machine $M$ that *does not* query the oracle on its own input, for every polynomial-time-constructible ensemble $\{X_n\}_{n\in\mathbb{N}}$, where $X_n$ ranges over $\{0,1\}^n$, for every polynomial $p(\cdot)$, and for all sufficiently large $n$'s,

$$\Pr\left[M^{F_n}(X_n) = F_n(X_n)\right] < \frac{1}{2} + \frac{1}{p(n)}$$

Again, show that this definition is a necessary but insufficient condition for pseudorandom function ensembles.

**Guideline:** Modify the function ensemble so that each $f$ in the range of $F_n$ satisfies $f(f(a^1)\,f(a^2)\cdots f(a^n)) = 0$, where $a^1, \ldots, a^n \in \{0,1\}^n$ are some easy-to-compute strings (e.g., $a^i = 0^{i-1}10^{n-i}$).

**Exercise 35:** Let $t : \mathbb{N} \to \mathbb{N}$ be such that on input $n$, one can compute $t(n)$ in poly($n$) time. Let $\{F_n\}_{n\in\mathbb{N}}$ and $\{H_n\}_{n\in\mathbb{N}}$ be two function ensembles that are indistinguishable by any probabilistic polynomial-time oracle machine. Prove that the permutation ensembles $\{\mathrm{DES}^{t(n)}_{F_n}\}_{n\in\mathbb{N}}$ and $\{\mathrm{DES}^{t(n)}_{H_n}\}_{n\in\mathbb{N}}$ (defined as in Section 3.7.2) are indistinguishable by any probabilistic polynomial-time oracle machine. Furthermore, this holds even when the oracle machine is given access both to the permutation and to its inverse (as in Definition 3.7.5).

**Guideline:** Use a hybrid argument to bridge between the $t(n)$ independent copies of $F_n$ and the $t(n)$ independent copies of $H_n$. The $i$th hybrid is $\mathrm{DES}_{F_n^{(t(n))},\ldots,F_n^{(i+1)},H_n^{(i)},\ldots,H_n^{(1)}}$. Note that oracle access to the permutation $\mathrm{DES}_{F_n^{(t(n))},\ldots,F_n^{(i+2)},g,H_n^{(i)},\ldots,H_n^{(1)}}$ (as well as to its inverse) can be emulated by using oracle access to the function $g$.

**Exercise 36:** Let $F_n$ and $\mathrm{DES}^t_{F_n}$ be as in Construction 3.7.6. Prove that regardless of the choice of the ensemble $F = \{F_n\}_{n\in\mathbb{N}}$, the ensemble $\mathrm{DES}^2_{F_n}$ is *not* pseudorandom.

**Guideline:** Start by showing that the ensemble $\mathrm{DES}^1_{F_n}$ is *not* pseudorandom (a single query suffices here). Use two related queries in order to distinguish $\mathrm{DES}^2_{F_n}$ from a random permutation.

**Exercise 37 (Suggested by Luca Trevisan):** Assuming the existence of pseudorandom function ensembles, prove that there exists a pseudorandom permutation ensemble that is *not* strongly pseudorandom.

**Guideline:** First construct a pseudorandom permutation ensemble with seed length smaller than or equal to the logarithm of domain size. Next modify it so that the seed is mapped to a fixed point (e.g., the all-zero string) and so that the modified ensemble remains one of permutations.

**Exercise 38:** In similarity to Exercise 36, prove that the ensemble $\mathrm{DES}^3_{F_n}$ is *not strongly* pseudorandom.

**Guideline:** This requires more thought and probably more than a couple of queries. You should definitely use queries to both oracles.

CHAPTER 4

# Zero-Knowledge Proof Systems

In this chapter we discuss zero-knowledge (ZK) proof systems. Loosely speaking, such proof systems have the remarkable property of being convincing *and* yielding nothing (beyond the validity of the assertion). In other words, receiving a zero-knowledge proof that an assertion holds is equivalent to being told by a trusted party that the assertion holds (see illustration in Figure 4.1). The main result presented in this chapter is a method for constructing zero-knowledge proof systems for every language in $\mathcal{NP}$. This method can be implemented using any bit-commitment scheme, which in turn can be implemented using any pseudorandom generator. The importance of this method stems from its generality, which is the key to its many applications. Specifically, almost all statements one may wish to prove in practice can be encoded as claims concerning membership in languages in $\mathcal{NP}$. In addition, we discuss more advanced aspects of the concept of zero-knowledge and their effects on the applicability of this concept.

**Organization.** The basic material is presented in Sections 4.1 through 4.4. In particular, we start with motivation (Section 4.1), next we define and exemplify the notions of interactive proofs (Section 4.2) and of zero-knowledge (Section 4.3), and finally

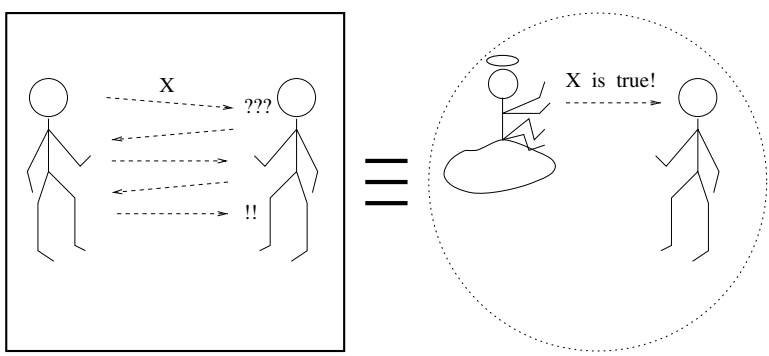


**Figure 4.1:** Zero-knowledge proofs: an illustration.

| | |
|---|---|
| Section 4.5: | *Negative results* |
| Section 4.6: | *Witness indistinguishability* and *witness hiding* |
| Section 4.7: | *Proofs of knowledge* |
| Section 4.8: | *Computationally sound proofs* (*arguments*) |
| Section 4.9: | *Constant-round* zero-knowledge systems |
| Section 4.10: | *Non-interactive zero-knowledge* proofs |
| Section 4.11: | *Multi-prover* zero-knowledge proofs |

**Figure 4.2:** The advanced sections of this chapter.

we present a zero-knowledge proof system for every language in $\mathcal{NP}$ (Section 4.4). Sections dedicated to advanced topics follow (see Figure 4.2). Unless stated differently (in the following list and in Figure 4.3), each of these advanced sections can be read independently of the others.

- In Section 4.5 we present some *negative results* regarding zero-knowledge proofs. These results demonstrate the "optimality" of the results in Section 4.4 and motivate the variants presented in Sections 4.6 and 4.8.
- In Section 4.6 we present a major relaxation of zero-knowledge and prove that it is closed under parallel composition (which is not the case, in general, for zero-knowledge). Here we refer to a notion called *witness indistinguishability*, which is related to *witness hiding* (also defined and discussed).
- In Section 4.7 we define and discuss (zero-knowledge) *proofs of knowledge*.
- In Section 4.8 we discuss a relaxation of interactive proofs, termed *computationally sound proofs* (or *arguments*).
- In Section 4.9 we present two constructions of *constant-round* zero-knowledge systems. The first is an interactive proof system, whereas the second is an argument system. Section 4.8.2 (discussing perfectly hiding commitment schemes) is a prerequisite for the first construction, whereas Sections 4.8, 4.7, and 4.6 constitute a prerequisite for the second.

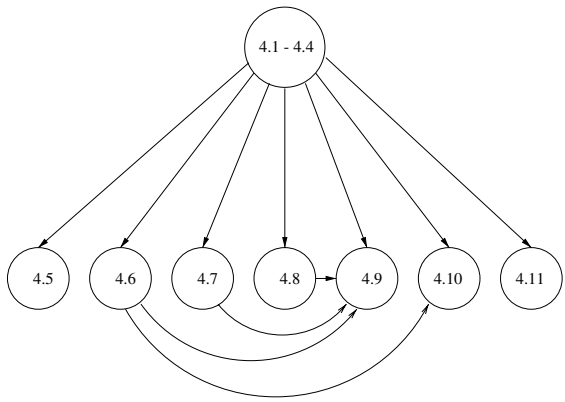


**Figure 4.3:** The dependence structure of this chapter.

- In Section 4.10 we discuss *non-interactive zero-knowledge* proofs. The notion of witness indistinguishability (defined in Section 4.6) is a prerequisite for the results presented in Section 4.10.3.1.
- In Section 4.11 we discuss *multi-prover* proof systems.

We conclude, as usual, with a miscellaneous section (Section 4.12).

**Teaching Tip.** The interactive proof system for Graph Non-Isomorphism (presented in Section 4.2) and the zero-knowledge proof of Graph Isomorphism (presented in Section 4.3) are merely illustrative examples. Thus, one should avoid analyzing those examples in detail.

## 4.1. Zero-Knowledge Proofs: Motivation

An archetypical cryptographic problem consists of providing mutually distrustful parties with a means of disclosing (predetermined) "pieces of information." It refers to settings in which parties posses secrets, and they wish to reveal parts of these secrets. The secrets are fully or partially determined by some publicly known information, and so it makes sense to talk about revealing the correct value of the secret. The question is how to allow verification of newly revealed parts of the secret without disclosing other parts of the secret. To clarify the issue, let us consider a specific example.

> Suppose that all users in a system keep backups of their entire file systems, encrypted using their secret keys, in publicly accessible storage media. Suppose that at some point, one user, called `Alice`, wishes to reveal to another user, called `Bob`, the cleartext of some record in one of her files (which appears in her backup). A trivial "solution" is for `Alice` simply to send the (cleartext) record to `Bob`. The problem with this "solution" is that `Bob` has no way of verifying that `Alice` has really sent him the true record (as appearing encrypted in her public backup), rather than just sending him an arbitrary record. `Alice` could prove that she sent the correct record simply by revealing to `Bob` her secret key. However, doing so would reveal to `Bob` the contents of all her files, which is certainly something that `Alice` does not want. The question is whether or not `Alice` can convince `Bob` that she has indeed revealed the correct record without yielding any additional "knowledge."
>
> An analogous problem can be phrased formally as follows. Let $f$ be a one-way *permutation* and $b$ a hard-core predicate with respect to $f$. Suppose that one party, $A$, has a string $x$, whereas another party, denoted $B$, has only $f(x)$. Furthermore, suppose that $A$ wishes to reveal $b(x)$ to party $B$, without yielding any further information. The trivial "solution" is to let $A$ send $b(x)$ to $B$, but, as explained earlier, $B$ will have no way of verifying that $A$ has really sent the correct bit (and not its complement). Party $A$ could indeed prove that it has sent the correct bit (i.e., $b(x)$) by sending $x$ as well, but revealing $x$ to $B$ would be much more than what $A$ originally had in mind. Again, the question is whether or not $A$ can convince $B$ that it has indeed revealed the correct bit (i.e., $b(x)$), without yielding any additional "knowledge."

In general, the question is whether or not *it is possible to prove a statement without yielding anything beyond its validity*. Such proofs, whenever they exist, are called *zero-knowledge*, and they play a central role in the construction of "cryptographic" protocols.

Loosely speaking, *zero-knowledge proofs are proofs that yield nothing* (i.e., "no knowledge") *beyond the validity of the assertion*. In the rest of this introductory section, we discuss the notion of a "proof" and a possible meaning of the phrase "yield nothing (i.e., no knowledge) beyond something."

### 4.1.1. The Notion of a Proof

*A proof is whatever convinces me.*
Shimon Even, answering a student's question
in his graph-algorithms class (1978)

We discuss the notion of a proof with the intention of uncovering some of its underlying aspects.

#### 4.1.1.1. A Static Object versus an Interactive Process

Traditionally in mathematics, a "proof" is a *fixed* sequence consisting of statements that either are self-evident or are derived from previous statements via self-evident rules. Actually, it is more accurate to replace the phrase "self-evident" with the phrase "commonly agreed." In fact, in the formal study of proofs (i.e., logic), the commonly agreed statements are called *axioms*, whereas the commonly agreed rules are referred to as *derivation rules*. We wish to stress two properties of mathematical proofs:

**1.** Proofs are viewed as fixed objects.

**2.** Proofs are considered at least as fundamental as their consequences (i.e., the theorems).

However, in other areas of human activity, the notion of a "proof" has a much wider interpretation. In particular, a proof is not a fixed object, but rather a process by which the validity of an assertion is established. For example, withstanding cross-examination in court can yield what can be considered a proof in law, and failure to provide an adequate answer to a rival's claim is considered a proof in philosophical, political, and sometimes even technical discussions. In addition, in many real-life situations, proofs are considered secondary (in importance) to their consequences.

To summarize, in "canonical" mathematics, proofs have a static nature (e.g., they are "written"), whereas in real-life situations proofs have a dynamic nature (i.e., they are established via an interaction). A dynamic interpretation of the notion of a proof is more appropriate to our setting, in which proofs are used as tools (i.e., sub-protocols) inside "cryptographic" protocols. Furthermore, a dynamic interpretation (at least in a weak sense) is essential to the non-triviality of the notion of a zero-knowledge proof.

#### 4.1.1.2. Prover and Verifier

The notion of a prover is implicit in all discussions of proofs, be it in mathematics or in real-life situations: The prover is the (sometimes hidden or transcendental) entity

providing the proof. In contrast, the notion of a verifier tends to be more explicit in such discussions, which typically emphasize the *verification process*, or in other words the role of the verifier. Both in mathematics and in real-life situations, proofs are defined in terms of the verification procedure. The verification procedure is considered to be relatively simple, and the burden is placed on the party/person supplying the proof (i.e., the prover).

The asymmetry between the complexity of the verification task and the complexity of the theorem-proving task is captured by the complexity class $\mathcal{NP}$, which can be viewed as a class of proof systems. Each language $L \in \mathcal{NP}$ has an efficient verification procedure for proofs of statements of the form "$x \in L$." Recall that each $L \in \mathcal{NP}$ is characterized by a polynomial-time-recognizable relation $R_L$ such that

$$L = \{x : \exists y \text{ s.t. } (x, y) \in R_L\}$$

and $(x, y) \in R_L$ only if $|y| \leq \text{poly}(|x|)$. Hence, the verification procedure for membership claims of the form "$x \in L$" consists of applying the (polynomial-time) algorithm for recognizing $R_L$ to the claim (encoded by) $x$ and a prospective proof, denoted $y$. Any $y$ satisfying $(x, y) \in R_L$ is considered a *proof* of membership of $x \in L$. Thus, correct statements (i.e., $x \in L$) and only these have proofs in this proof system. Note that the verification procedure is "easy" (i.e., polynomial-time), whereas coming up with proofs may be "difficult" (if indeed $\mathcal{NP}$ is not contained in $\mathcal{BPP}$).

It is worthwhile to note the "distrustful attitude" toward the prover that underlies any proof system. If the verifier trusts the prover, then no proof is needed. Hence, whenever discussing a proof system, one considers a setting in which the verifier is not trusting the prover and furthermore is skeptical of anything the prover says.

### 4.1.1.3. Completeness and Soundness

Two fundamental properties of a proof system (i.e., a verification procedure) are its *soundness* (or *validity*) and *completeness*. The soundness property asserts that the verification procedure cannot be "tricked" into accepting false statements. In other words, *soundness* captures the verifier's ability to protect itself from being convinced of false statements (no matter what the prover does in order to fool the verifier). On the other hand, *completeness* captures the ability of some prover to convince the verifier of true statements (belonging to some predetermined set of true statements). Note that both properties are essential to the very notion of a proof system.

We remark here that not every set of true statements has a "reasonable" proof system in which each of those statements can be proved (while no false statement can be "proved"). This fundamental fact is given precise meaning in results such as *Gödel's Incompleteness Theorem* and *Turing's* theorem regarding the *undecidability of the Halting Problem*. We stress that in this chapter we confine ourselves to the class of sets (of valid statements) that do have "efficient proof systems." In fact, Section 4.2 is devoted to discussing and formulating the concept of "efficient proof systems." Jumping ahead, we hint that the efficiency of a proof system will be associated with the efficiency of its verification procedure.

### 4.1.2. Gaining Knowledge

Recall that we have motivated zero-knowledge proofs as proofs by which the verifier gains "no knowledge" (beyond the validity of the assertion). The reader may rightfully wonder what knowledge is and what a gain in knowledge is. When discussing zero-knowledge proofs, we avoid the first question (which is quite complex) and treat the second question directly. Namely, *without* presenting a definition of knowledge, we present a generic case in which it is certainly justified to say that no knowledge is gained. Fortunately, this approach seems to suffice as far as cryptography is concerned.

To motivate the definition of zero-knowledge, consider a conversation between two parties, `Alice` and `Bob`. Assume first that this conversation is unidirectional; specifically, `Alice` only talks, and `Bob` only listens. Clearly, we can say that `Alice` gains no knowledge from the conversation. On the other hand, `Bob` may or may not gain knowledge from the conversation (depending on what `Alice` says). For example, if all that `Alice` says is "$1+1=2$," then clearly `Bob` gains no knowledge from the conversation, because he already knows that fact. If, on the other hand, `Alice` reveals to `Bob` a proof that $\mathcal{P} \neq \mathcal{NP}$, then he certainly gains knowledge from the conversation.

To give a better flavor of the definition, we now consider a conversation between `Alice` and `Bob` in which `Bob` asks `Alice` questions about a large graph (that is known to both of them). Consider first the case in which `Bob` asks `Alice` whether or not the graph[1] is Eulerian. Clearly, `Bob` gains no knowledge from `Alice`'s answer, because he could easily have determined the answer by himself (by running a linear-time decision procedure[2]). On the other hand, if `Bob` asks `Alice` whether or not the graph is Hamiltonian, and `Alice` (somehow) answers this question, then we cannot say that `Bob` has gained no knowledge (because we do not know of an efficient procedure by which `Bob` could have determined the answer by himself, and assuming $\mathcal{P} \neq \mathcal{NP}$, no such efficient procedure exists). Hence, we say that `Bob` *has gained knowledge* from the interaction if his *computational ability*, concerning the publicly known graph, *has increased* (i.e., if after the interaction he can easily compute something that he could not have efficiently computed before the interaction). On the other hand, if whatever `Bob` can efficiently compute about the graph *after interacting* with `Alice` he can also efficiently compute *by himself* (from the graph), then we say that `Bob` *has gained no knowledge* from the interaction. That is, `Bob` gains knowledge only if he receives the result of a computation that is infeasible for him. The question of how `Alice` could conduct this infeasible computation (e.g., answer `Bob`'s question of whether or not the graph is Hamiltonian) has been ignored thus far. Jumping ahead, we remark that `Alice` may be a mere abstraction or may be in possession of additional hints that enable her to efficiently conduct computations that are otherwise infeasible (and in particular are infeasible for `Bob`, who does not have these hints).

[1] See Footnote 13.

[2] For example, by relying on Euler's theorem, which asserts that a graph is Eulerian if and only if it is connected and all its vertices have even degrees.

### Knowledge versus Information

We wish to stress that *knowledge* (as discussed here) is very different from *information* (in the sense of information theory). Two major aspects of the difference are as follows:

1. *Knowledge* is related to computational difficulty, whereas *information* is not. In the foregoing examples, there is a difference between the knowledge revealed in the case in which `Alice` answers questions of the form "Is the graph Eulerian?" and the case in which she answers questions of the form "Is the graph Hamiltonian?" From an information-theory point of view there is no difference between the two cases (i.e., in each case the answer is determined by the question, and so `Bob` gets no information).
2. *Knowledge* relates mainly to publicly known objects, whereas *information* relates mainly to objects on which only partial information is publicly known. Consider the case in which `Alice` answers each question by flipping an unbiased coin and telling `Bob` the outcome. From an information-theoretic point of view, `Bob` gets from `Alice` information concerning an event. However, we say that `Bob` gains no knowledge from `Alice`, because he could toss coins by himself.

## 4.2. Interactive Proof Systems

In this section we introduce the notion of an interactive proof system and present a non-trivial example of such a system (specifically to claims of the form "the following two graphs are *not* isomorphic"). The presentation is directed toward the introduction of zero-knowledge interactive proofs. Interactive proof systems are interesting for their own sake and have important complexity-theoretic applications.[3]

### 4.2.1. Definition

The definition of an interactive proof system refers explicitly to the two computational tasks related to a proof system: "producing" a proof and verifying the validity of a proof. These tasks are performed by two different parties, called the *prover* and the *verifier*, which interact with one another. In some cases, the interaction may be very simple and, in particular, unidirectional (i.e., the prover sends a text, called the proof, to the verifier). In general, the interaction may be more complex and may take the form of the verifier interrogating the prover. We start by defining such an interrogation process.

#### 4.2.1.1. Interaction

Interaction between two parties is defined in the natural manner. The only point worth noting is that the interaction is parameterized by a common input (given to both parties). In the context of interactive proof systems, the common input represents the statement to be proved. We first define the notion of an interactive machine and next the notion

[3] See the suggestions for further reading at the end of the chapter.

of interaction between two such machines. The reader may skip to Section 4.2.1.2, which introduces some important conventions (regarding interactive machines), with little loss (if any).

**Definition 4.2.1 (An Interactive Machine):**

- *An* **interactive Turing machine** (ITM) *is a (deterministic) multi-tape Turing machine. The tapes are a read-only* input tape*, a read-only* random tape*, a read-and-write* work tape*, a write-only* output tape*, a pair of* communication tapes*, and a read-and-write* switch tape *consisting of a single cell. One communication tape is read-only, and the other is write-only.*
- *Each ITM is associated a single bit* $\sigma \in \{0,1\}$*, called its* **identity**. *An ITM is said to be* active*, in a configuration, if the content of its switch tape equals the machine's identity. Otherwise the machine is said to be* idle*. While being idle, the state of the machine, the locations of its heads on the various tapes, and the contents of the writable tapes of the ITM are not modified.*
- *The content of the input tape is called* **input***, the content of the random tape is called* random input*, and the content of the output tape at termination is called* **output**. *The content written on the write-only communication tape during a (time) period in which the machine is active is called the* **message sent** *at that period. Likewise, the content read from the read-only communication tape during an active period is called the* **message received** *(at that period).*

  *(Without loss of generality, the machine movements on both communication tapes are in only one direction, e.g., from left to right.)*

This definition, taken by itself, seems quite non-intuitive. In particular, one may say that once being idle, the machine will never become active again. One may also wonder as to what is the point of distinguishing the read-only communication tape from the input tape (and respectively distinguishing the write-only communication tape from the output tape). The point is that we are never going to consider a single interactive machine, but rather a pair of machines combined together such that some of their tapes coincide. Intuitively, the messages sent by one interactive machine are received by a second machine that shares its communication tapes (so that the read-only communication tape of one machine coincides with the write-only tape of the other machine). The active machine may become idle by changing the content of the shared switch tape, and when it does so, the other machine (having opposite identity) will become active. The computation of such a pair of machines consists of the machines alternately sending messages to one another, based on their initial (common) input, their (distinct) random inputs, and the messages each machine has received thus far.

**Definition 4.2.2 (Joint Computation of Two ITMs):**

- *Two interactive machines are said to be* **linked** *if they have opposite identities, their input tapes coincide, their switch tapes coincide, and the read-only communication tape of one machine coincides with the write-only communication tape of the other machine, and vice versa. We stress that the other tapes of both machines (i.e., the random tape, the work tape, and the output tape) are distinct.*

- *The* **joint computation** *of a linked pair of ITMs, on a common input $x$, is a sequence of pairs representing the local configurations of both machines. That is, each pair consists of two strings, each representing the local configuration of one of the machines. In each such pair of local configurations, one machine (not necessarily the same one) is active, while the other machine is idle. The first pair in the sequence consists of initial configurations corresponding to the common input $x$, with the content of the switch tape set to zero.*
- *If one machine halts while the switch tape still holds its identity, then we say that both machines have halted. The* **outputs** *of both machines are determined at that time.*

At this point, the reader may object to this definition, saying that the individual machines are deprived of individual local inputs (whereas they are given individual and unshared random tapes). This restriction is removed in Section 4.2.4, and in fact allowing individual local inputs (in addition to the common shared input) is quite important (at least as far as practical purposes are concerned). Yet, for a first presentation of interactive proofs, as well as for demonstrating the power of this concept, we prefer the foregoing simpler definition. On the other hand, the convention of individual random tapes is essential to the power of interactive proofs (see Exercise 4).

### 4.2.1.2. Conventions Regarding Interactive Machines

Typically, we consider executions in which the content of the random tape of each machine is uniformly and independently chosen (among all infinite bit sequences). The convention of having an infinite sequence of internal coin tosses should not bother the reader, because during a finite computation only a finite prefix is read (and matters). The content of each of these random tapes can be viewed as internal coin tosses of the corresponding machine (as in the definition of ordinary probabilistic machines presented in Chapter 1). Hence, interactive machines are in fact probabilistic.

**Notation.** Let $A$ and $B$ be a linked pair of ITMs, and suppose that all possible interactions of $A$ and $B$ on each common input terminate in a finite number of steps. We denote by $\langle A, B\rangle(x)$ the random variable representing the (local) output of $B$ when interacting with machine $A$ on common input $x$, when the random input to each machine is uniformly and independently chosen. (Indeed, this definition is asymmetric, since it considers only $B$'s output.)

Another important convention is to consider the time-complexity of an interactive machine as a function of only its input's length.

**Definition 4.2.3 (The Complexity of an Interactive Machine):** *We say that an interactive machine $A$ has* **time-complexity** $t : \mathbb{N} \to \mathbb{N}$ *if for every interactive machine $B$ and every string $x$, it holds that when interacting with machine $B$, on common input $x$, machine $A$ always (i.e., regardless of the content of its random tape and $B$'s random tape) halts within $t(|x|)$ steps. In particular, we say that $A$ is* **polynomial-time** *if there exists a polynomial $p$ such that $A$ has time-complexity $p$.*

We stress that the time-complexity, so defined, is independent of the content of the messages that machine $A$ receives. In other words, it is an upper bound that holds for all possible incoming messages (as well as all internal coin tosses). In particular, an interactive machine with time-complexity $t(\cdot)$ may read, on input $x$, only a prefix of total length $t(|x|)$ of the messages sent to it.

### 4.2.1.3. Proof Systems

In general, proof systems are defined in terms of the verification procedure (which can be viewed as one entity, called the verifier). A "proof" for a specific claim is always considered as coming from the outside (which can be viewed as another entity, called the prover). The verification procedure itself does not generate "proofs," but merely verifies their validity. Interactive proof systems are intended to capture whatever can be efficiently verified via interaction with the outside. In general, the interaction with the outside can be very complex and may consist of many message exchanges, as long as the total time spent by the verifier is polynomial (in the common input).

Our choice to consider probabilistic polynomial-time verifiers is justified by the association of efficient procedures with probabilistic polynomial-time algorithms. Furthermore, the verifier's verdict of whether to accept or reject the claim is probabilistic, and a bounded error probability is allowed. (Jumping ahead, we mention that the error can be decreased to be negligible by repeating the verification procedure sufficiently many times.)

Loosely speaking, we require that the prover be able to convince the verifier of the validity of true statements, while nobody can fool the verifier into believing false statements. Both conditions are given a probabilistic interpretation: It is required that the verifier accept valid statements with "high" probability, whereas the probability that it will accept a false statement is "low" (regardless of the machine with which the verifier interacts). In the following definition, the verifier's output is interpreted as its decision on whether to accept or reject the common input. Output 1 is interpreted as "accept," whereas output 0 is interpreted as "reject."

**Definition 4.2.4 (Interactive Proof System):** *A pair of interactive machines $(P, V)$ is called an* **interactive proof system for a language** *$L$ if machine $V$ is polynomial-time and the following two conditions hold:*

- Completeness: *For every $x \in L$,*

$$\Pr[\langle P, V\rangle(x) = 1] \geq \frac{2}{3}$$

- Soundness: *For every $x \notin L$ and every interactive machine $B$,*

$$\Pr[\langle B, V\rangle(x) = 1] \leq \frac{1}{3}$$

Some remarks are in order. We first stress that the soundness condition refers to all potential "provers," whereas the completeness condition refers only to the prescribed prover $P$. Second, the verifier is required to be a (probabilistic) polynomial-time machine, but

no resource bounds are placed on the computing power of the prover (in either completeness or soundness conditions). Third, as in the case of $\mathcal{BPP}$, the error probability in the foregoing definition can be made exponentially small by repeating the interaction (polynomially) many times.

Every language in $\mathcal{NP}$ has an interactive proof system. Specifically, let $L \in \mathcal{NP}$, and let $R_L$ be a witness relation associated with the language $L$ (i.e., $R_L$ is recognizable in polynomial time, and $L$ equals the set $\{x : \exists y \text{ s.t. } |y| = \text{poly}(|x|) \wedge (x, y) \in R_L\}$). Then an interactive proof for the language $L$ consists of a prover that on input $x \in L$ sends a witness $y$ (as before), and a verifier that upon receiving $y$ (on common input $x$) outputs 1 if $|y| = \text{poly}(|x|)$ and $(x, y) \in R_L$ (and outputs 0 otherwise). Clearly, when interacting with the prescribed prover, this verifier will always accept inputs in the language. On the other hand, no matter what a cheating "prover" does, this verifier will never accept inputs not in the language. We point out that in this specific proof system, both parties are deterministic (i.e., make no use of their random tapes). It is easy to see that only languages in $\mathcal{NP}$ have interactive proof systems in which both parties are deterministic (see Exercise 2).

In other words, $\mathcal{NP}$ can be viewed as a class of interactive proof systems in which the interaction is unidirectional (i.e., from the prover to the verifier) and the verifier is deterministic (and never errs). In general interactive proofs, *both* restrictions are waived: The interaction is bidirectional, and the verifier is probabilistic (and may err, with some small probability). Both bidirectional interaction and randomization seem essential to the power of interactive proof systems (see Exercise 2).

**Definition 4.2.5 (The Class $\mathcal{IP}$):** *The class $\mathcal{IP}$ consists of all languages having interactive proof systems.*

By the foregoing discussion, $\mathcal{NP} \subseteq \mathcal{IP}$. Because languages in $\mathcal{BPP}$ can be viewed as each having a verifier that decides on membership without any interaction, it follows that $\mathcal{BPP} \cup \mathcal{NP} \subseteq \mathcal{IP}$. We remind the reader that it is not known whether or not $\mathcal{BPP} \subseteq \mathcal{NP}$.

We next show that the definition of the class $\mathcal{IP}$ remains invariant if we replace the (constant) bounds in the completeness and soundness conditions with two functions $\mathsf{c}, \mathsf{s} : \mathbb{N} \to [0, 1]$ satisfying $\mathsf{c}(n) < 1 - 2^{-\text{poly}(n)}$, $\mathsf{s}(n) > 2^{-\text{poly}(n)}$, and $\mathsf{c}(n) > \mathsf{s}(n) + \frac{1}{\text{poly}(n)}$. Namely, we consider the following generalization of Definition 4.2.4.

**Definition 4.2.6 (Generalized Interactive Proof):** *Let $\mathsf{c}, \mathsf{s} : \mathbb{N} \to \mathbb{R}$ be functions satisfying $\mathsf{c}(n) > \mathsf{s}(n) + \frac{1}{p(n)}$ for some polynomial $p(\cdot)$. An interactive pair $(P, V)$ is called a (generalized) interactive proof system for the language $L$, with* **completeness bound** $\mathsf{c}(\cdot)$ *and* **soundness bound** $\mathsf{s}(\cdot)$*, if*

- (modified) completeness: *for every $x \in L$,*

$$\Pr[\langle P, V\rangle(x) = 1] \geq \mathsf{c}(|x|)$$

- (modified) soundness: *for every $x \notin L$ and every interactive machine $B$,*

$$\Pr[\langle B, V\rangle(x) = 1] \leq \mathsf{s}(|x|)$$

*The function* $\mathsf{g}(\cdot)$ *defined as* $\mathsf{g}(n) \stackrel{\text{def}}{=} \mathsf{c}(n) - \mathsf{s}(n)$ *is called the* acceptance gap *of* $(P, V)$*, and the function* $\mathsf{e}(\cdot)$*, defined as* $\mathsf{e}(n) \stackrel{\text{def}}{=} \max\{1 - \mathsf{c}(n), \mathsf{s}(n)\}$*, is called the* error probability *of* $(P, V)$*. In particular,* $\mathsf{s}$ *is the* soundness error *of* $(P, V)$*, and* $1 - \mathsf{c}$ *is its* completeness error.

We stress that $\mathsf{c}$ is a lower bound, whereas $\mathsf{s}$ is an upper bound.

**Proposition 4.2.7:** *The following three conditions are equivalent:*

1. $L \in \mathcal{IP}$*. Namely, there exists an interactive proof system, with completeness bound* $\frac{2}{3}$ *and soundness bound* $\frac{1}{3}$*, for the language* $L$*.*
2. $L$ has very strong interactive proof systems: *For every polynomial* $p(\cdot)$*, there exists an interactive proof system for the language* $L$*, with error probability bounded above by* $2^{-p(\cdot)}$*.*
3. $L$ has a very weak interactive proof: *There exists a polynomial* $p(\cdot)$ *and a generalized interactive proof system for the language* $L$*, with acceptance gap bounded below by* $1/p(\cdot)$*. Furthermore, completeness and soundness bounds for this system, namely, the values* $\mathsf{c}(n)$ *and* $\mathsf{s}(n)$*, can be computed in time polynomial in* $n$*.*

Clearly, either of the first two items implies the third one (including the requirement for efficiently computable bounds). The ability to efficiently compute completeness and soundness bounds is used in proving the opposite (non-trivial) direction. The proof is left as an exercise (i.e., Exercise 1).

### 4.2.2. An Example (Graph Non-Isomorphism in $\mathcal{IP}$)

All examples of interactive proof systems presented thus far have been degenerate (e.g., the interaction, if any, has been unidirectional). We now present an example of a non-degenerate interactive proof system. Furthermore, we present an interactive proof system for a language *not known to be in* $\mathcal{BPP} \cup \mathcal{NP}$. Specifically, the language is the set of *pairs of non-isomorphic graphs*, denoted $GNI$. The idea underlying this proof system is presented through the following story:

> Petra von Kant claims that Goldstar[4] beer in large bottles tastes different than Goldstar beer in small bottles. Virgil does not believe her. To prove her claim, Petra and Virgil repeat the following process a number of times sufficient to convince Virgil beyond reasonable doubt.
>
> Virgil selects at random either a large bottle or a small one and pours some beer into a tasting glass, without Petra seeing which bottle he uses. Virgil then hands Petra the glass and asks her to tell which of the bottles he has used.
>
> If Petra never errs in her answers, then Virgil is convinced of the validity of her claim. (In fact, he should be convinced even if she answers correctly with probability substantially larger than 50%, because if the beer tastes the same

[4] Goldstar is an Israeli beer, available in 330-ml and 500-ml bottles. Actually, the story traces back to Athena's claim regarding jars of nectar, which was contested by Zeus himself. Unfortunately, Ovid does not tell the outcome of their interaction.

regardless of the bottle, then there would be no way for Petra to guess correctly with probability higher than 50% which bottle was used.)

We now get back to the formal exposition. Let us first define the language in focus: Two graphs,[5] $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, are called *isomorphic* if there exists a 1-1 and onto mapping, $\pi$, from the vertex set $V_1$ to the vertex set $V_2$ such that $(u, v) \in E_1$ if and only if $(\pi(v), \pi(u)) \in E_2$. The mapping $\pi$, if it exists, is called an *isomorphism* between the graphs. The set of pairs of non-isomorphic graphs is denoted by $GNI$.

**Construction 4.2.8 (An Interactive Proof System for Graph Non-Isomorphism):**

- Common input: *A pair of two graphs, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Suppose, without loss of generality, that $V_1 = \{1, 2, \ldots, |V_1|\}$, and similarly for $V_2$.*
- Verifier's first step (V1): *The verifier selects at random one of the two input graphs and sends to the prover a random isomorphic copy of this graph. Namely, the verifier selects uniformly $\sigma \in \{1, 2\}$ and a random permutation $\pi$ from the set of permutations over the vertex set $V_\sigma$. The verifier constructs a graph with vertex set $V_\sigma$ and edge set*

$$F \stackrel{\text{def}}{=} \{(\pi(u), \pi(v)) : (u, v) \in E_\sigma\}$$

  *and sends $(V_\sigma, F)$ to the prover.*
- Motivating remark: *If the input graphs are non-isomorphic, as the prover claims, then the prover should be able to distinguish (not necessarily by an efficient procedure) isomorphic copies of one graph from isomorphic copies of the other graph. However, if the input graphs are isomorphic, then a random isomorphic copy of one graph will be distributed identically to a random isomorphic copy of the other graph.*
- Prover's first step (P1): *Upon receiving a graph $G' = (V', E')$ from the verifier, the prover finds $\tau \in \{1, 2\}$ such that the graph $G'$ is isomorphic to the input graph $G_\tau$. (If both $\tau = 1$ and $r = 2$ satisfy the condition, then $\tau$ is selected arbitrarily. In case no $\tau \in \{1, 2\}$ satisfies the condition, $\tau$ is set to 0.) The prover sends $\tau$ to the verifier.*
- Verifier's second step (V2): *If the message $\tau$ received from the prover equals $\sigma$ (chosen in Step V1), then the verifier outputs 1 (i.e., accepts the common input). Otherwise the verifier outputs 0 (i.e., rejects the common input).*

This verifier program is easily implemented in probabilistic polynomial time. We do not know of a probabilistic polynomial-time implementation of the prover's program, but this is not required. We shall now show that the foregoing pair of interactive machines constitutes an interactive proof system (in the general sense) for the language $GNI$ (Graph Non-Isomorphism).

**Proposition 4.2.9:** *The language $GNI$ is in the class $\mathcal{IP}$. Furthermore, the programs specified in Construction 4.2.8 constitute a generalized interactive proof system for $GNI$, with completeness bound 1 and soundness bound $\frac{1}{2}$. Namely:*

[5] See footnote 13.

1. *If $G_1$ and $G_2$ are not isomorphic (i.e., $(G_1, G_2) \in GNI$), then the verifier always accepts (when interacting with the prover).*
2. *If $G_1$ and $G_2$ are isomorphic (i.e., $(G_1, G_2) \notin GNI$), then no matter with what machine the verifier interacts, it rejects the input with probability at least $\frac{1}{2}$.*

***Proof:*** Clearly, if $G_1$ and $G_2$ are not isomorphic, then no graph can be isomorphic to both $G_1$ and $G_2$. It follows that there exists a unique $\tau$ such that the graph $G'$ (received by the prover in Step P1) is isomorphic to the input graph $G_\tau$. Hence, $\tau$ found by the prover in Step P1 always equals $\sigma$ chosen in Step V1. Part 1 follows.

On the other hand, if $G_1$ and $G_2$ are isomorphic, then the graph $G'$ is isomorphic to both input graphs. Furthermore, we shall show that in this case the graph $G'$ yields no information about $\sigma$, and consequently no machine can (on input $G_1$, $G_2$ and $G'$) set $\tau$ such that it will equal $\sigma$ with probability greater than $\frac{1}{2}$. Details follow.

Let $\pi$ be a permutation on the vertex set of a graph $G = (V, E)$. We denote by $\pi(G)$ the graph with vertex set $V$ and edge set $\{(\pi(u), \pi(v)) : (u, v) \in E\}$. Let $\xi$ be a random variable uniformly distributed over $\{1, 2\}$, and let $\Pi$ be a random variable uniformly distributed over the set of permutations on $V$. We stress that these two random variables are independent. We are interested in the distribution of the random variable $\Pi(G_\xi)$. We are going to show that although $\Pi(G_\xi)$ is determined by the random variables $\Pi$ and $\xi$, the random variables $\xi$ and $\Pi(G_\xi)$ are statistically independent. In fact, we show the following:

**Claim 4.2.9.1:** If the graphs $G_1$ and $G_2$ are isomorphic, then for every graph $G'$ that is isomorphic to $G_1$ (and $G_2$), it holds that

$$\mathsf{Pr}[\xi = 1 \mid \Pi(G_\xi) = G'] = \mathsf{Pr}[\xi = 2 \mid \Pi(G_\xi) = G'] = \frac{1}{2}$$

**Proof:** We first claim that the sets $S_1 \stackrel{\text{def}}{=} \{\pi : \pi(G_1) = G'\}$ and $S_2 \stackrel{\text{def}}{=} \{\pi : \pi(G_2) = G'\}$ are of the same cardinality. This follows from the observation that there is a 1-1 and onto correspondence between the set $S_1$ and the set $S_2$ (the correspondence is given by the isomorphism between the graphs $G_1$ and $G_2$). Hence,

$$\begin{aligned}\mathsf{Pr}[\Pi(G_\xi) = G' \mid \xi = 1] &= \mathsf{Pr}[\Pi(G_1) = G'] \\ &= \mathsf{Pr}[\Pi \in S_1] \\ &= \mathsf{Pr}[\Pi \in S_2] \\ &= \mathsf{Pr}[\Pi(G_\xi) = G' | \xi = 2]\end{aligned}$$

Using Bayes' rule, the claim follows. □

Intuitively, Claim 4.2.9.1 says that for every pair $(G_1, G_2)$ of isomorphic graphs, the random variable $\Pi(G_\xi)$ yields no information on $\xi$, and so no prover can fool the verifier into accepting with probability greater than $\frac{1}{2}$. Specifically, we let $R$ be an arbitrary randomized process (representing a possible cheating-prover strategy that depends on $(G_1, G_2)$) that given the verifier's message in Step V1

tries to guess the value of $\xi$. Then, $R(\Pi(G_\xi)) = \xi$ represents the event in which the verifier accepts, and we have

$$\mathsf{Pr}[R(\Pi(G_\xi)) = \xi] = \sum_{G'} \mathsf{Pr}[\Pi(G_\xi) = G'] \cdot \mathsf{Pr}[R(G') = \xi \mid \Pi(G_\xi) = G']$$

Using Claim 4.2.9.1 for the third equality, we have (for any $G'$ in the support of $\Pi(G_\xi)$):

$$\begin{aligned}
\mathsf{Pr}[R(G') = \xi \mid \Pi(G_\xi) = G'] &= \sum_{v} \mathsf{Pr}[R(G') = v \,\&\, \xi = v \mid \Pi(G_\xi) = G'] \\
&= \sum_{v} \mathsf{Pr}[R(G') = v] \cdot \mathsf{Pr}[\xi = v \mid \Pi(G_\xi) = G'] \\
&= \sum_{v \in \{1,2\}} \mathsf{Pr}[R(G') = v] \cdot \mathsf{Pr}[\xi = v] \\
&= \frac{\mathsf{Pr}[R(G') \in \{1,2\}]}{2} \\
&\leq \frac{1}{2}
\end{aligned}$$

with equality in case $R$ always outputs an element in the set $\{1, 2\}$. Part 2 of the proposition follows. ■

**Remarks Concerning Construction 4.2.8.** In the proof system of Construction 4.2.8, the verifier *always* accepts inputs *in* the language (i.e., the completeness error equals zero). The fact that $GNI \in \mathcal{IP}$, whereas it is not known whether or not $GNI \in \mathcal{NP}$, is an indication of the power of interaction and randomness in the context of theorem-proving. Finally, we note that it is essential that the prover not know the outcome of the verifier's internal coin tosses. For a wider perspective on these issues, see the following advanced subsection.

### 4.2.3.* The Structure of the Class $\mathcal{IP}$

In continuation to the foregoing remarks, we briefly discuss several aspects regarding the "proving power" of interactive proof systems.

1. *The completeness and soundness bounds*: All interactive proof systems presented in this book happen to have *perfect completeness*; that is, the verifier *always* accepts inputs IN the language (i.e., the completeness error equals zero). In fact, one can *transform* any interactive proof system into an interactive proof system (for the same language) in which the verifier always accepts inputs in the language.

   On the other hand, as shown in Exercise 5, only languages in $\mathcal{NP}$ have interactive proof systems in which the verifier *always rejects* inputs NOT IN the language (i.e., having soundness error equal to zero).

2. *The privacy of the verifier's coins*: *Arthur-Merlin proofs* (a.k.a. *public-coin proof systems*) are a special case of interactive proofs, where the verifier must send the outcome of any coin it tosses (and thus need not send any other information). As stated earlier, the

proof system of Construction 4.2.8 is *not* of the public-coin type. Yet one can *transform* any interactive proof system into a *public-coin* interactive proof system (for the same language), while preserving perfect completeness.

**3.** *Which languages have interactive proof systems*? (We have ignored this natural question until now.) It turns out that every language in $\mathcal{PSPACE}$ has an interactive proof system. In fact,

$$\mathcal{IP} \text{ equals } \mathcal{PSPACE}$$

We comment that $\mathcal{PSPACE}$ is believed to be much larger than $\mathcal{NP}$; in particular, co$\mathcal{NP} \subseteq \mathcal{PSPACE}$, whereas it is commonly believed that co$\mathcal{NP} \neq \mathcal{NP}$. Also, because $\mathcal{PSPACE}$ is closed under complementation, so is $\mathcal{IP}$.

**4.** *Constant-round interactive proofs*: Construction 4.2.8 constitutes a constant-round protocol (i.e., a constant number of messages are sent). In contrast, in the generic interactive proof system for $\mathcal{PSPACE}$, the number of communication rounds is polynomially related to the input length. We comment that co$\mathcal{NP}$ is believed NOT to have *constant-round* interactive proofs.

We mention that any language having a *constant-round* interactive proof system also has a public-coin interactive proof system in which only two messages are sent: The latter consists of a random challenge from the verifier that is answered by the prover. In general, for any function $r : \mathbb{N} \to \mathbb{N}$, any $2r$-round proof system can be transformed into an $r$-round proof system (for the same language).

### 4.2.4. Augmentation of the Model

For purposes that will become more clear in Sections 4.3 and 4.4, we augment the basic definition of an interactive proof system by allowing each of the parties to have a private input (in addition to the common input). Loosely speaking, these inputs are used to capture additional information available to each of the parties. Specifically, when using interactive proof systems as sub-protocols inside larger protocols, the private inputs are associated with the local configurations of the machines before entering the sub-protocol. In particular, the private input of the prover may contain information that enables an efficient implementation of the prover's task.

**Definition 4.2.10 (Interactive Proof Systems, Revisited):**

***1.*** *An* interactive machine *is defined as in Definition 4.2.1, except that the machine has an additional read-only tape called* **the auxiliary-input tape**. *The content of this tape is called* **auxiliary input**.

***2.*** *The complexity of such an interactive machine is still measured as a function of the (common) input length. Namely, the interactive machine $A$ has* time-complexity $t : \mathbb{N} \to \mathbb{N}$ *if for every interactive machine $B$ and every string $x$, it holds that when interacting with machine $B$, on common input $x$, machine $A$ always (i.e., regardless of the content of its random tape and its auxiliary-input tape, as well as the content of $B$'s tapes) halts within $t(|x|)$ steps.*

3. *We denote by $\langle A(y), B(z)\rangle(x)$ the random variable representing the (local) output of $B$ when interacting with machine $A$ on common input $x$, when the random input to each machine is uniformly and independently chosen, and $A$ (resp., $B$) has auxiliary input $y$ (resp., $z$).*
4. *A pair of interactive machines $(P, V)$ is called an* interactive proof system for a language $L$ *if machine $V$ is polynomial-time and the following two conditions hold:*
   - Completeness: *For every $x \in L$, there exists a string $y$ such that for every $z \in \{0,1\}^*$,*
   $$\Pr[\langle P(y), V(z)\rangle(x) = 1] \geq \frac{2}{3}$$
   - Soundness: *For every $x \notin L$, every interactive machine $B$, and every $y, z \in \{0,1\}^*$,*
   $$\Pr[\langle B(y), V(z)\rangle(x) = 1] \leq \frac{1}{3}$$

We stress that when saying that an interactive machine is polynomial-time, we mean that its running time is polynomial in the length of the common input. Consequently, it is not guaranteed that such a machine has enough time to read its entire auxiliary input.

**Teaching Tip.** The augmented model of interactive proofs is first used in this book in Section 4.3.3, where the notion of zero-knowledge is extended to account for a priori information that the verifier may have. One may thus prefer to present Definition 4.2.10 after presenting the basic definitions of zero-knowledge, that is, postpone Definition 4.2.10 to Section 4.3.3. (However, conceptually speaking, Definition 4.2.10 does belong to the current section.)

## 4.3. Zero-Knowledge Proofs: Definitions

In this section we introduce the notion of a zero-knowledge interactive proof system and present a non-trivial example of such a system (specifically, to claims of the form "the following two graphs are isomorphic").

### 4.3.1. Perfect and Computational Zero-Knowledge

Loosely speaking, we say that an interactive proof system $(P, V)$ for a language $L$ is zero-knowledge if whatever can be efficiently computed after interacting with $P$ on input $x \in L$ can also be efficiently computed from $x$ (without any interaction). We stress that this holds with respect to any efficient way of interacting with $P$, not necessarily the way defined by the verifier program $V$. Actually, zero-knowledge is a property of the prescribed prover $P$. It captures $P$'s robustness against attempts to gain knowledge by interacting with it. A straightforward way of capturing the informal discussion follows.

Let $(P, V)$ be an interactive proof system for some language $L$. We say that $(P, V)$, or actually $P$, is *perfect zero-knowledge* if for every probabilistic polynomial-time interactive machine $V^*$ there exists an (*ordinary*) probabilistic polynomial-time algorithm $M^*$ such that for every $x \in L$ the following two random variables are identically distributed:

- $\langle P, V^* \rangle(x)$ (i.e., the output of the interactive machine $V^*$ after interacting with the interactive machine $P$ on common input $x$)
- $M^*(x)$ (i.e., the output of machine $M^*$ on input $x$)

Machine $M^*$ is called a *simulator* for the interaction of $V^*$ with $P$.

We stress that we require that *for every* $V^*$ interacting with $P$, not merely for $V$, there exists a ("perfect") simulator $M^*$. This simulator, although not having access to the interactive machine $P$, is able to simulate the interaction of $V^*$ with $P$. The fact that such simulators exist means that $V^*$ does not gain any knowledge from $P$ (since the same output could be generated without any access to $P$).

### The Simulation Paradigm

The foregoing discussion follows a general definitional paradigm that is also used in other chapters of this book (specifically, in Volume 2). The *simulation paradigm* postulates that whatever a party can do by itself cannot be considered a gain from interaction with the outside. The validity of this paradigm is evident, provided we bear in mind that by "doing" we mean "efficiently doing" something (and more so if the complexity of "doing it alone" is tightly related to the complexity of "doing it after interaction with the outside").[6] Admittedly, failure to provide a simulation of an interaction with the outside does NOT necessarily mean that this interaction results in some "real gain" (in some intuitive sense). Yet what matters is that any "real gain" can NOT occur whenever we are able to present a simulation. In summary, the approach underlying the simulation paradigm may be overly cautious, but it is certainly valid. (Furthermore, to say the least, it seems much harder to provide a robust definition of "real gain.")

**Trivial Cases.** Note that every language in $\mathcal{BPP}$ has a perfect zero-knowledge proof system in which the prover does nothing (and the verifier checks by itself whether to accept or reject the common input). To demonstrate the zero-knowledge property of this "dummy prover," one can present for every verifier $V^*$ a simulator $M^*$ that is essentially identical to $V^*$ (except that the communication tapes of $V^*$, which are never used, are considered as ordinary work tapes of $M^*$).

#### 4.3.1.1. Perfect Zero-Knowledge

Unfortunately, the preceding formulation of (perfect) zero-knowledge is slightly too strict (at least as far as we know).[7] We relax the formulation by allowing the simulator to fail, with bounded probability, to produce an interaction.

[6] See the discussion of knowledge tightness in Section 4.4.4.2.

[7] That is, we do not know of any non-trivial case in which that requirement is satisfied. In contrast, non-trivial cases satisfying the relaxed definition given next are known, and we actually present one (i.e., a perfect zero-knowledge proof for Graph Isomorphism).

**Definition 4.3.1 (Perfect Zero-Knowledge):** *Let $(P, V)$ be an interactive proof system for some language $L$. We say that $(P, V)$ is* **perfect zero-knowledge** *if for every probabilistic polynomial-time interactive machine $V^*$ there exists a probabilistic polynomial-time algorithm $M^*$ such that for every $x \in L$ the following two conditions hold:*

1. *With probability at most $\frac{1}{2}$, on input $x$, machine $M^*$ outputs a special symbol denoted $\perp$ (i.e., $\mathsf{Pr}[M^*(x) = \perp] \leq \frac{1}{2}$).*
2. *Let $m^*(x)$ be a random variable describing the distribution of $M^*(x)$ conditioned on $M^*(x) \neq \perp$ (i.e., $\mathsf{Pr}[m^*(x) = \alpha] = \mathsf{Pr}[M^*(x) = \alpha \mid M^*(x) \neq \perp]$ for every $\alpha \in \{0,1\}^*$). Then the following random variables are identically distributed:*
   - *$\langle P, V^* \rangle(x)$ (i.e., the output of the interactive machine $V^*$ after interacting with the interactive machine $P$ on common input $x$)*
   - *$m^*(x)$ (i.e., the output of machine $M^*$ on input $x$, conditioned on not being $\perp$)*

*Machine $M^*$ is called a* **perfect simulator** *for the interaction of $V^*$ with $P$.*

Condition 1 can be replaced by a stronger condition requiring that $M^*$ output the special symbol (i.e., $\perp$) only with negligible probability. For example, one can require that (on input $x$) machine $M^*$ will output $\perp$ with probability bounded above by $2^{-p(|x|)}$, for any polynomial $p(\cdot)$; see Exercise 6. Consequently, the statistical difference between the random variables $\langle P, V^* \rangle(x)$ and $M^*(x)$ can be made negligible (in $|x|$); see Exercise 8. Hence, whatever the verifier efficiently computes after interacting with the prover can be efficiently computed (with only an extremely small error) by the simulator (and hence by the verifier himself).

### 4.3.1.2. Computational Zero-Knowledge

Following the spirit of Chapter 3, we observe that for practical purposes there is no need to be able to "perfectly simulate" the output of $V^*$ after it interacts with $P$. Instead, it suffices to generate a probability distribution that is computationally indistinguishable from the output of $V^*$ after it interacts with $P$. The relaxation is consistent with our original requirement that "whatever can be efficiently computed after interacting with $P$ on input $x \in L$ can also be efficiently computed from $x$ (without any interaction)," the reason being that we consider computationally indistinguishable ensembles as being the same. Before presenting the relaxed definition of general zero-knowledge, we recall the definition of computationally indistinguishable ensembles (see Item 2 in Definition 3.2.2). Here we consider ensembles indexed by strings from a language $L$. We say that the ensembles $\{R_x\}_{x \in L}$ and $\{S_x\}_{x \in L}$ are **computationally indistinguishable** if for every probabilistic polynomial-time algorithm $D$, for every polynomial $p(\cdot)$, and for all sufficiently long $x \in L$, it holds that

$$|\mathsf{Pr}[D(x, R_x) = 1] - \mathsf{Pr}[D(x, S_x) = 1]| < \frac{1}{p(|x|)}$$

**Definition 4.3.2 (Computational Zero-Knowledge):** *Let $(P, V)$ be an interactive proof system for some language $L$. We say that $(P, V)$ is* **computational zero-knowledge** *(or just* **zero-knowledge***) if for every probabilistic polynomial-time interactive machine $V^*$ there exists a probabilistic polynomial-time algorithm $M^*$ such that the following two ensembles are computationally indistinguishable:*

- $\{\langle P, V^*\rangle(x)\}_{x\in L}$ *(i.e., the output of the interactive machine $V^*$ after it interacts with the interactive machine $P$ on common input $x$)*
- $\{M^*(x)\}_{x\in L}$ *(i.e., the output of machine $M^*$ on input $x$)*

*Machine $M^*$ is called a* simulator *for the interaction of $V^*$ with $P$.*

The reader can easily verify (see Exercise 9) that allowing the simulator to output the special symbol $\perp$ (with probability bounded above by, say, $\frac{1}{2}$) and considering the conditional output distribution (as done in Definition 4.3.1) does not add to the power of Definition 4.3.2.

**The Scope of Zero-Knowledge.** We stress that both definitions of zero-knowledge apply to interactive proof systems in the general sense (i.e., having any noticeable gap between the acceptance probabilities for inputs inside and outside the language). In fact, the definitions of zero-knowledge apply to any pair of interactive machines (actually to each interactive machine): Namely, we can say that the interactive machine $A$ is *zero-knowledge on $L$* if whatever can be efficiently computed after the interaction with $A$ on common input $x \in L$ can also be efficiently computed from $x$ itself.

### 4.3.1.3. An Alternative Formulation of Zero-Knowledge

An alternative formulation of zero-knowledge considers the verifier's view of the interaction with the prover, rather than only the output of the verifier after such an interaction. By the "verifier's view of the interaction" we mean the entire sequence of the local configurations of the verifier during an interaction (execution) with the prover. Clearly, it suffices to consider only the content of the random tape of the verifier and the sequence of messages that the verifier has received from the prover during the execution (since the entire sequence of local configurations and the final output are determined by those objects).

**Definition 4.3.3 (Zero-Knowledge, Alternative Formulation):** *Let $(P, V)$, $L$, and $V^*$ be as in Definition 4.3.2. We denote by* $\mathrm{view}^P_{V^*}(x)$ *a random variable describing the content of the random tape of $V^*$ and the messages $V^*$ receives from $P$ during a joint computation on common input $x$. We say that $(P, V)$ is* **zero-knowledge** *if for every probabilistic polynomial-time interactive machine $V^*$ there exists a probabilistic polynomial-time algorithm $M^*$ such that the ensembles $\{\mathrm{view}^P_{V^*}(x)\}_{x\in L}$ and $\{M^*(x)\}_{x\in L}$ are computationally indistinguishable.*

A few remarks are in order. First, note that Definition 4.3.3 is obtained from Definition 4.3.2 by replacing $\langle P, V^*\rangle(x)$ with $\mathrm{view}^P_{V^*}(x)$. The simulator $M^*$ used in

Definition 4.3.3 is related to but not equal to the simulator used in Definition 4.3.2 (yet this fact is not reflected in the text of those definitions). Clearly, $\langle P, V^* \rangle(x)$ can be computed in (deterministic) polynomial time from $\text{view}^P_{V^*}(x)$ for each $V^*$. Although that is not always true for the opposite direction, Definition 4.3.3 is equivalent to Definition 4.3.2 (by virtue of the universal quantification on the $V^*$'s; see Exercise 10). The latter fact justifies the use of Definition 4.3.3, which is more convenient to work with, although it seems less natural than Definition 4.3.2. An analogous alternative formulation of perfect zero-knowledge can be obtained from Definition 4.3.1 and is clearly equivalent to it.

### 4.3.1.4.* Almost-Perfect (Statistical) Zero-Knowledge

A less drastic (than computational zero-knowledge) relaxation of the notion of perfect zero-knowledge is the following:

**Definition 4.3.4 (Almost-Perfect (Statistical) Zero-Knowledge):** *Let $(P, V)$ be an interactive proof system for some language $L$. We say that $(P, V)$ is* **almost-perfect zero-knowledge** (or **statistical zero-knowledge**) *if for every probabilistic polynomial-time interactive machine $V^*$ there exists a probabilistic polynomial-time algorithm $M^*$ such that the following two ensembles are statistically close as functions of $|x|$:*

- $\{\langle P, V^* \rangle(x)\}_{x \in L}$ *(i.e., the output of the interactive machine $V^*$ after it interacts with the interactive machine $P$ on common input $x$)*
- $\{M^*(x)\}_{x \in L}$ *(i.e., the output of machine $M^*$ on input $x$).*

*That is, the statistical difference between $\langle P, V^* \rangle(x)$ and $M^*(x)$ is negligible in terms of $|x|$.*

As in the case of computational zero-knowledge, allowing the simulator to output the symbol $\perp$ (with probability bounded above by, say, $\frac{1}{2}$) and considering the conditional output distribution (as done in Definition 4.3.1) does not add to the power of Definition 4.3.4; see Exercise 8. It is also easy to show that perfect zero-knowledge implies almost-perfect zero-knowledge, which in turn implies computational zero-knowledge.

The three definitions (i.e., perfect, almost-perfect, and computational zero-knowledge) correspond to a natural three-stage hierarchy of interpretations of the notion of "close" pairs of probability ensembles. (In all three cases, the pairs of ensembles being postulated as being close are $\{\langle P, V^* \rangle(x)\}_{x \in L}$ and $\{M^*(x)\}_{x \in L}$.)

1. The most stringent interpretation of closeness is the requirement that the two ensembles be identically distributed. This is the requirement in the case of perfect zero-knowledge.
2. A slightly more relaxed interpretation of closeness is that the two ensembles be statistically indistinguishable (or statistically close). This is the requirement in the case of almost-perfect (or statistical) zero-knowledge.
3. A much more relaxed interpretation of closeness, which suffices for all practical purposes, is that the two ensembles be computationally indistinguishable. This is the requirement in the case of computational zero-knowledge.

### 4.3.1.5.* Complexity Classes Based on Zero-Knowledge

The various definitions of zero-knowledge give rise to natural complexity classes:

**Definition 4.3.5 (Class of Languages Having Zero-Knowledge Proofs):** *We denote by* $\mathcal{ZK}$ (also $\mathcal{CZK}$) *the class of languages having* (computational) *zero-knowledge interactive proof systems. Likewise,* $\mathcal{PZK}$ (*resp.,* $\mathcal{SZK}$) *denotes the class of languages having* perfect (*resp.,* statistical) *zero-knowledge interactive proof systems.*

Clearly,

$$\mathcal{BPP} \subseteq \mathcal{PZK} \subseteq \mathcal{SZK} \subseteq \mathcal{CZK} \subseteq \mathcal{IP}$$

Assuming the existence of (non-uniformly) one-way functions, the last inclusion is an equality (i.e., $\mathcal{CZK} = \mathcal{IP}$); see Proposition 4.4.5 and Theorems 3.5.12 and 4.4.12. On the other hand, we believe that the first and third inclusions are strict (as equalities in either case contradict widely believed complexity assumptions). Thus, our belief is that

$$\mathcal{BPP} \subset \mathcal{PZK} \subseteq \mathcal{SZK} \subset \mathcal{CZK} = \mathcal{IP}$$

The relationship of $\mathcal{PZK}$ to $\mathcal{SZK}$ remains an open problem (with no evidence either way).

### 4.3.1.6.* Expected Polynomial-Time Simulators

The formulation of perfect zero-knowledge presented in Definition 4.3.1 is different from the definition used in some early publications in the literature. The original definition requires that the simulator *always* output a legal transcript (which has to be distributed identically to the real interaction), yet it allows the simulator to run in *expected* polynomial time rather than in strictly polynomial time. We stress that the expectation is taken over the coin tosses of the simulator (whereas the input to the simulator is fixed). This yields the following:

**Definition 4.3.6 (Perfect Zero-Knowledge, Liberal Formulation):** *We say that* $(P, V)$ *is* **perfect zero-knowledge in the liberal sense** *if for every* probabilistic *polynomial-time interactive machine* $V^*$ *there exists an* expected *polynomial-time algorithm* $M^*$ *such that for every* $x \in L$ *the random variables* $\langle P, V^* \rangle(x)$ *and* $M^*(x)$ *are identically distributed.*

We stress that by *probabilistic polynomial time* we mean a strict bound on the running time in all possible executions, whereas by *expected polynomial time* we allow non-polynomial-time executions but require that the running time be "polynomial on the average." Clearly, Definition 4.3.1 implies Definition 4.3.6 (see Exercise 7). Interestingly, there exist interactive proofs that are perfect zero-knowledge with respect to the liberal definition but are not known to be perfect zero-knowledge with respect to Definition 4.3.1. We point out that the naive way of transforming an expected

probabilistic polynomial-time algorithm to one that runs in strict polynomial time is not suitable for the current context.[8]

We prefer to adopt Definition 4.3.1, rather than Definition 4.3.6, because we want to avoid the notion of expected polynomial time. The main reason for our desire to avoid the latter notion is that the correspondence between average polynomial time and efficient computations is more controversial than the widely accepted association of strict polynomial time with efficient computations. Furthermore, the notion of expected polynomial time is more subtle than one realizes at first glance:

> The naive interpretation of expected polynomial time is having an *average* running time that is *bounded by a polynomial* in the input length. This definition of expected polynomial time is unsatisfactory because it is *not closed under reductions* and is (too) *machine-dependent*. Both aggravating phenomena follow from the fact that a function can have an average (say over $\{0,1\}^n$) that is bounded by a polynomial (in $n$) and yet squaring the function will yield a function that is not bounded by a polynomial (in $n$). For example, the function $f(x) \stackrel{\text{def}}{=} 2^{|x|}$ if $x \in \{0\}^*$, and $f(x) \stackrel{\text{def}}{=} |x|^2$ otherwise, satisfies $\mathsf{E}[f(U_n)] < n^2 + 1$, but $\mathsf{E}[f(U_n)^2] > 2^n$.
>
> Hence, a better interpretation of expected polynomial time is having a running time that is *bounded by a polynomial in a function that has an average linear growth rate*. That is, using the naive definition of linear on the average, we say that $f$ is *polynomial on the average* if there exist a polynomial $p$ and a linear-on-the-average function $\ell$ such that $f(x) \leq p(\ell(x))$ for all sufficiently long $x$'s. Note that if $f$ is polynomial on the average, then so is $f^2$.

An analogous discussion applies to computational zero-knowledge. More specifically, Definition 4.3.2 requires that the simulator work in polynomial time, whereas a more liberal notion would allow it to work in *expected* polynomial time.

We comment that for the sake of elegance it is customary to modify the definitions that allow *expected* polynomial-time simulators by requiring that such simulators also exist for the interaction of *expected* polynomial-time verifiers with the prover.

### 4.3.1.7.* Honest-Verifier Zero-Knowledge

We briefly discuss a weak notion of zero-knowledge. The notion, called *honest-verifier zero-knowledge*, requires simulatability of the view of only the *prescribed* (or *honest*) verifier, rather than simulatability of the view of *any possible* (probabilistic polynomial-time) verifier. Although this notion does not suffice for typical cryptographic applications, it is interesting for at least a couple of reasons: First, this weak notion of zero-knowledge is highly non-trivial and fascinating by itself. Second, public-coin

[8] The naive transformation truncates runs of the algorithm (in our case, the simulator) that take more than $t$ times the expected number of steps. (Such a truncated run is said to produce some fixed output.) The statistical difference between the output distribution of the original algorithm and the output distribution of the modified algorithm is at most $1/t$. The problem is that $t$ must be bounded by a fixed polynomial in the running time, and so the statistical difference is not negligible. To see that the analysis of this naive transformation is tight, consider its effect on the following algorithm: On input $1^n$, the algorithm first selects uniformly $r \in \{0,1\}^n$, next takes $2^i$ idle steps, where $i$ is the length of the longest all-zero prefix of $r$, and finally runs $S(1^n)$, where $S$ is an arbitrary (strict) probabilistic polynomial-time algorithm.

protocols that are zero-knowledge with respect to the honest verifier can be transformed into similar protocols that are zero-knowledge in general. We stress that in the current context (of the single prescribed verifier) the formulations of *output simulatability* (as in Definition 4.3.2) and *view simulatability* (as in Definition 4.3.3) are NOT equivalent, and it is important to use the latter.[9]

**Definition 4.3.7 (Zero-Knowledge with Respect to an Honest Verifier):** *Let* $(P, V)$*, L, and* $\text{view}_V^P(x)$ *be as in Definition 4.3.3. We say that* $(P, V)$ *is* **honest-verifier zero-knowledge** *if there exists a probabilistic polynomial-time algorithm* $M$ *such that the ensembles* $\{\text{view}_V^P(x)\}_{x\in L}$ *and* $\{M(x)\}_{x\in L}$ *are computationally indistinguishable.*

The preceding definition refers to computational zero-knowledge and is a restriction of Definition 4.3.3. Versions for perfect and statistical zero-knowledge are defined analogously.

### 4.3.2. An Example (Graph Isomorphism in $\mathcal{PZK}$)

As mentioned earlier, every language in $\mathcal{BPP}$ has a trivial (i.e., degenerate) zero-knowledge proof system. We now present an example of a non-degenerate zero-knowledge proof system. Furthermore, we present a zero-knowledge proof system for a language not known to be in $\mathcal{BPP}$. Specifically, the language is the set of *pairs of isomorphic graphs*, denoted $GI$ (see definition in Section 4.2). Again, the idea underlying this proof system is presented through a story:

> In this story, Petra von Kant claims that there is a footpath between the north gate and the south gate of her labyrinth (i.e., a path going inside the labyrinth). Virgil does not believe her. Petra is willing to prove her claim to Virgil, but does not want to provide him any additional knowledge (and, in particular, not to assist him to find an inside path from the north gate to the south gate). To prove her claim, Petra and Virgil repeat the following process a number of times sufficient to convince Virgil beyond reasonable doubt.
>
> Petra miraculously transports Virgil to a random place in her labyrinth. Then Virgil asks to be shown the way to either the north gate or the south gate. His choice is supposed to be random, but he may try to cheat. Petra then chooses a (sufficiently long) random walk from their current location to the desired destination and guides Virgil along that walk.
>
> Clearly, if the labyrinth has a path as claimed (and Petra knows her way in the labyrinth), then Virgil will be convinced of the validity of her claim. If, on the other hand, the labyrinth has no such path, then at each iteration, with probability at least 50%, Virgil will detect that Petra is lying. Finally, Virgil will gain no knowledge from the guided tour, the reason being that he can simulate a guided

[9] Note that for any interactive proof of perfect completeness, the output of the honest verifier is trivially simulatable (by an algorithm that always outputs 1). In contrast, many of the negative results presented in Section 4.5 also apply to zero-knowledge with respect to an honest verifier, as defined next. For example, only languages in $\mathcal{BPP}$ have *unidirectional* proof systems that are zero-knowledge with respect to an honest verifier.

tour by himself, as follows: First, he selects north or south (as he does in the real guided tour) and goes to the suitable gate (from outside the labyrinth). Next, he takes a random walk from the gate to inside the labyrinth while unrolling a spool of thread behind him, and finally he traces the thread back to the gate. (A sufficiently long random walk whose length equals the length of the tour guided by Petra will guarantee that Virgil will visit a random place in the labyrinth, and the way back will look like a random walk from the location at the end of his thread to the chosen gate.)

We now get back to the formal exposition.

**Construction 4.3.8 (A Perfect Zero-Knowledge Proof for Graph Isomorphism):**

- Common input: *A pair of two graphs,* $G_1 = (V_1, E_1)$ *and* $G_2 = (V_2, E_2)$*. Let* $\phi$ *be an isomorphism between the input graphs; namely,* $\phi$ *is a* 1-1 *and onto mapping of the vertex set* $V_1$ *to the vertex set* $V_2$ *such that* $(u, v) \in E_1$ *if and only if* $(\phi(v), \phi(u)) \in E_2$*.*
- Prover's first step (P1): *The prover selects a random isomorphic copy of* $G_2$ *and sends it to the verifier. Namely, the prover selects at random, with uniform probability distribution, a permutation* $\pi$ *from the set of permutations over the vertex set* $V_2$ *and constructs a graph with vertex set* $V_2$ *and edge set*

$$F \stackrel{\text{def}}{=} \{(\pi(u), \pi(v)) : (u, v) \in E_2\}$$

  *The prover sends* $(V_2, F)$ *to the verifier.*
- Motivating remark: *If the input graphs are isomorphic, as the prover claims, then the graph sent in Step P1 is isomorphic to both input graphs. However, if the input graphs are not isomorphic, then no graph can be isomorphic to both of them.*
- Verifier's first step (V1): *Upon receiving a graph* $G' = (V', E')$ *from the prover, the verifier asks the prover to show an isomorphism between* $G'$ *and one of the input graphs, chosen at random by the verifier. Namely, the verifier uniformly selects* $\sigma \in \{1, 2\}$ *and sends it to the prover (who is supposed to answer with an isomorphism between* $G_\sigma$ *and* $G'$*).*
- Prover's second step (P2): *If the message* $\sigma$ *received from the verifier equals* 2*, then the prover sends* $\pi$ *to the verifier. Otherwise (i.e.,* $\sigma \neq 2$*), the prover sends* $\pi \circ \phi$ *(i.e., the composition of* $\pi$ *on* $\phi$*, defined as* $\pi \circ \phi(v) \stackrel{\text{def}}{=} \pi(\phi(v))$*) to the verifier. (Remark: The prover treats any* $\sigma \neq 2$ *as* $\sigma = 1$*.)*
- Verifier's second step (V2): *If the message, denoted* $\psi$*, received from the prover is an isomorphism between* $G_\sigma$ *and* $G'$*, then the verifier outputs* 1*; otherwise it outputs* 0*.*

*Let us denote the prover's program by* $P_{GI}$*.*

The verifier program just presented is easily implemented in probabilistic polynomial time. In case the prover is given an isomorphism between the input graphs as auxiliary input, the prover's program can also be implemented in probabilistic polynomial time. We now show that this pair of interactive machines constitutes a *zero-knowledge* interactive proof system (in the general sense) for the language $GI$ (Graph Isomorphism).

**Proposition 4.3.9:** *The language $GI$ has a perfect zero-knowledge interactive proof system. Furthermore, the programs specified in Construction 4.3.8 satisfy the following:*

1. *If $G_1$ and $G_2$ are isomorphic (i.e., $(G_1, G_2) \in GI$), then the verifier always accepts (when interacting with $P_{GI}$).*
2. *If $G_1$ and $G_2$ are not isomorphic (i.e., $(G_1, G_2) \notin GI$), then no matter with which machine the verifier interacts, it will reject the input with probability at least $\frac{1}{2}$.*
3. *The prover (i.e., $P_{GI}$) is perfect zero-knowledge. Namely, for every probabilistic polynomial-time interactive machine $V^*$, there exists a probabilistic polynomial-time algorithm $M^*$ outputting $\perp$ with probability at most $\frac{1}{2}$, so that for every $x \stackrel{\text{def}}{=} (G_1, G_2) \in GI$, the following two random variables are identically distributed:*
   - $\text{view}_{V^*}^{P_{GI}}(x)$ *(i.e., the view of $V^*$ after interacting with $P_{GI}$, on common input $x$)*
   - $m^*(x)$ *(i.e., the output of machine $M^*$, on input $x$, conditioned on not being $\perp$).*

A zero-knowledge interactive proof system for $GI$ with error probability $2^{-k}$ (only in the soundness condition) can be derived by executing the foregoing protocol, *sequentially*, $k$ times. We stress that in each repetition of the protocol, both the (prescribed) prover and verifier must use "fresh" coin tosses that are independent of the coin tosses used in prior repetitions of the protocol. For further discussion, see Section 4.3.4. We remark that $k$ *parallel* executions will also decrease the error in the soundness condition to $2^{-k}$, but the resulting interactive proof is not known to be zero-knowledge in the case in which $k$ grows faster than logarithmic in the input length. In fact, we believe that such an interactive proof is *not* zero-knowledge. For further discussion, see Section 4.5.

We stress that it is not known whether or not $GI \in \mathcal{BPP}$. Hence, Proposition 4.3.9 asserts the existence of a perfect zero-knowledge proof for a language not known to be in $\mathcal{BPP}$.

***Proof:*** We first show that these programs indeed constitute a (general) interactive proof system for $GI$. Clearly, if the input graphs $G_1$ and $G_2$ are isomorphic, then the graph $G'$ constructed in Step P1 will be isomorphic to both of them. Hence, if each party follows its prescribed program, then the verifier will always accept (i.e., output 1). Part 1 follows. On the other hand, if $G_1$ and $G_2$ are not isomorphic, then no graph can be isomorphic to both $G_1$ and $G_2$. It follows that no matter how the (possibly cheating) prover constructs $G'$, there exists $\sigma \in \{1, 2\}$ such that $G'$ and $G_\sigma$ are *not* isomorphic. Hence, if the verifier follows its program, then it will reject (i.e., output 0) with probability at least $\frac{1}{2}$. Part 2 follows.

It remains to show that $P_{GI}$ is indeed perfect zero-knowledge on $GI$. This is indeed the difficult part of the entire proof. It is easy to simulate the output of the verifier specified in Construction 4.3.8 (since its output is identically 1 for inputs in the language $GI$). Also, it is not hard to simulate the output of a verifier that follows the program specified in Construction 4.3.8, except that at termination it will output the entire transcript of its interaction with $P_{GI}$ (see Exercise 12). The difficult part is to simulate the output of an efficient verifier that deviates arbitrarily from the specified program.

We shall use here the alternative formulation of (perfect) zero-knowledge and show how to simulate $V^*$'s view of the interaction with $P_{GI}$ for every probabilistic polynomial-time interactive machine $V^*$. As mentioned earlier, it is not hard to simulate the verifier's view of the interaction with $P_{GI}$ when the verifier follows the specified program. However, we need to simulate the view of the verifier in the general case (in which it uses an arbitrary polynomial-time interactive program). Following is an overview of our simulation (i.e., of our construction of a simulator $M^*$ for each $V^*$).

The simulator $M^*$ incorporates the code of the interactive program $V^*$. On input $(G_1, G_2)$, the simulator $M^*$ first selects at random one of the input graphs (i.e., either $G_1$ or $G_2$) and generates a random isomorphic copy, denoted $G''$, of this input graph. In doing so, the simulator behaves differently from $P_{GI}$, but the graph generated (i.e., $G''$) is distributed identically to the message sent in Step P1 of the interactive proof. Say that the simulator has generated $G''$ by randomly permuting $G_1$. Now, if $V^*$ asks to see the isomorphism between $G_1$ and $G''$, then the simulator can indeed answer correctly, and in doing so it completes a simulation of the verifier's view of the interaction with $P_{GI}$. However, if $V^*$ asks to see the isomorphism between $G_2$ and $G''$, then the simulator (which, unlike $P_{GI}$, does not "know" $\phi$) has no way to answer correctly, and we let it halt with output $\perp$. We stress that the simulator "has no way of knowing" whether $V^*$ will ask to see an isomorphism to $G_1$ or to $G_2$. The point is that the simulator can try one of the possibilities at random, and if it is lucky (which happens with probability exactly $\frac{1}{2}$), then it can output a distribution that will be identical to the view of $V^*$ when interacting with $P_{GI}$ (on common input $(G_1, G_2)$). A key fact (see Claim 4.3.9.1, following) is that the distribution of $G''$ is stochastically independent of the simulator's choice of which of the two input graphs to use, and so $V^*$ cannot affect the probability that the simulator will be lucky. A detailed description of the simulator follows.

**Simulator $M^*$.** On input $x \stackrel{\text{def}}{=} (G_1, G_2)$, simulator $M^*$ proceeds as follows:

1. *Setting the random tape of $V^*$*: Let $q(\cdot)$ denote a polynomial bounding the running time of $V^*$. The simulator $M^*$ starts by uniformly selecting a string $r \in \{0,1\}^{q(|x|)}$ to be used as the content of the random tape of $V^*$. (Alternatively, one could produce coins for $V^*$ "on the fly," that is, during Step 3, which follows.)
2. *Simulating the prover's first step* (P1): The simulator $M^*$ selects at random, with uniform probability distribution, a "bit" $\tau \in \{1,2\}$ and a permutation $\psi$ from the set of permutations over the vertex set $V_\tau$. It then constructs a graph with vertex set $V_\tau$ and edge set
$$F \stackrel{\text{def}}{=} \{(\psi(u), \psi(v)) : (u,v) \in E_\tau\},$$
and sets $G'' \stackrel{\text{def}}{=} (V_\tau, F)$.
3. *Simulating the verifier's first step* (V1): The simulator $M^*$ initiates an execution of $V^*$ by placing $x$ on $V^*$'s common-input tape, placing $r$ (selected in Step 1) on $V^*$'s random tape, and placing $G''$ (constructed in Step 2) on $V^*$'s incoming-message tape. After executing a polynomial number of steps of $V^*$, the simulator can read

the outgoing message of $V^*$, denoted $\sigma$. To simplify the rest of the description, we *normalize* $\sigma$ by setting $\sigma = 1$ if $\sigma \neq 2$ (and leave $\sigma$ unchanged if $\sigma = 2$).

4. *Simulating the prover's second step* (P2): If $\sigma = \tau$, then the simulator halts with output $(x, r, G'', \psi)$.
5. *Failure of the simulation:* Otherwise (i.e., $\sigma \neq \tau$), the simulator halts with output $\perp$.

Using the hypothesis that $V^*$ is polynomial-time, it follows that so is the simulator $M^*$. It is left to show that $M^*$ outputs $\perp$ with probability at most $\frac{1}{2}$ and that, conditioned on not outputting $\perp$, the simulator's output is distributed as the verifier's view in a "real interaction with $P_{GI}$." The following claim is the key to the proof of both claims.

**Claim 4.3.9.1:** Suppose that the graphs $G_1$ and $G_2$ are isomorphic. Let $\xi$ be a random variable uniformly distributed in $\{1, 2\}$, and let $\Pi$ be a random variable uniformly distributed over the set of permutations over $V_\xi$. Then for every graph $G''$ that is isomorphic to $G_1$ (and $G_2$), it holds that

$$\mathsf{Pr}[\xi = 1 \mid \Pi(G_\xi) = G''] = \mathsf{Pr}[\xi = 2 \mid \Pi(G_\xi) = G''] = \frac{1}{2}$$

where, as in Claim 4.2.9.1, $\pi(G)$ denotes the graph obtained from the graph $G$ by relabeling its nodes using the permutation $\pi$.

Claim 4.3.9.1 is identical to Claim 4.2.9.1 (used to demonstrate that Construction 4.2.8 constitutes an interactive proof for $GNI$).[10] As in the rest of the proof of Proposition 4.2.9, it follows that any random process with output in $\{1, 2\}$, given $\Pi(G_\xi)$, outputs $\xi$ with probability exactly $\frac{1}{2}$. Hence, given $G''$ (constructed by the simulator in Step 2), the verifier's program yields (normalized) $\sigma$, so that $\sigma \neq \tau$ with probability exactly $\frac{1}{2}$. We conclude that the simulator outputs $\perp$ with probability $\frac{1}{2}$. It remains to prove that, conditioned on not outputting $\perp$, the simulator's output is identical to "$V^*$'s view of real interactions." Namely:

**Claim 4.3.9.2:** Let $x = (G_1, G_2) \in GI$. Then for every string $r$, graph $H$, and permutation $\psi$, it holds that

$$\mathsf{Pr}\left[\mathrm{view}_{V^*}^{P_{GI}}(x) = (x, r, H, \psi)\right] = \mathsf{Pr}[M^*(x) = (x, r, H, \psi) \mid M^*(x) \neq \perp]$$

**Proof:** Let $m^*(x)$ describe $M^*(x)$ conditioned on its not being $\perp$. We first observe that both $m^*(x)$ and $\mathrm{view}_{V^*}^{P_{GI}}(x)$ are distributed over quadruples of the form $(x, r, \cdot, \cdot)$, with uniformly distributed $r \in \{0, 1\}^{q(|x|)}$. Let $\nu(x, r)$ be a random variable describing the last two elements of $\mathrm{view}_{V^*}^{P_{GI}}(x)$ conditioned on the second element equaling $r$. Similarly, let $\mu(x, r)$ describe the last two elements of $m^*(x)$ (conditioned on the second element equaling $r$). We need to show that $\nu(x, r)$

[10] In Construction 4.2.8, the graph $\Pi(G_\xi)$ was presented to the prover, and Claim 4.2.9.1 was used to establish the soundness of the proof system (i.e., analyze what happens in case $(G_1, G_2) \notin GNI$, which means $(G_1, G_2) \in GI$). Here the graph $\Pi(G_\xi)$ is presented to the verifier, and the claim is used to establish the zero-knowledge property (and so also refers to $(G_1, G_2) \in GI$).

and $\mu(x, r)$ are identically distributed for every $x$ and $r$. Observe that once $r$ is fixed, the message sent by $V^*$, on common input $x$, random tape $r$, and incoming message $H$, is uniquely defined. Let us denote this message by $v^*(x, r, H)$. We show that both $v(x, r)$ and $\mu(x, r)$ are uniformly distributed over the set

$$C_{x,r} \stackrel{\text{def}}{=} \left\{(H, \psi) : H = \psi\left(G_{v^*(x,r,H)}\right)\right\}$$

where (again) $\psi(G)$ denotes the graph obtained from $G$ by relabeling the vertices using the permutation $\psi$ (i.e., if $G = (V, E)$, then $\psi(G) = (V, F)$, so that $(u, v) \in E$ iff $(\psi(u), \psi(v)) \in F$). The proof of this statement is rather tedious and is unrelated to the subjects of this book (and hence can be skipped with no damage).

The proof is slightly non-trivial because it relates (at least implicitly) to the automorphism group of the graph $G_2$ (i.e., the set of permutations $\pi$ for which $\pi(G_2)$ is identical with $G_2$, not just isomorphic to $G_2$). For simplicity, consider first the special case in which the automorphism group of $G_2$ consists of merely the identity permutation (i.e., $G_2 = \pi(G_2)$ if and only if $\pi$ is the identity permutation). In this case, $(H, \psi) \in C_{x,r}$ if and only if $H$ is isomorphic to (both $G_1$ and) $G_2$ and $\psi$ is the (unique) isomorphism between $H$ and $G_{v^*(x,r,H)}$. Hence, $C_{x,r}$ contains exactly $|V_2|!$ pairs, each containing a different graph $H$ as the first element. In the general case, $(H, \psi) \in C_{x,r}$ if and only if $H$ is isomorphic to (both $G_1$ and) $G_2$ and $\psi$ is an isomorphism between $H$ and $G_{v^*(x,r,H)}$. We stress that $v^*(x, r, H)$ is the same in all pairs containing $H$. Let $\text{aut}(G_2)$ denote the size of the automorphism group of $G_2$. Then each $H$ (isomorphic to $G_2$) appears in exactly $\text{aut}(G_2)$ pairs of $C_{x,r}$, and each such pair contains a different isomorphism between $H$ and $G_{v^*(x,r,H)}$. The number of different $H$'s that are isomorphic to $G_2$ is $|V_2|!/\text{aut}(G_2)$, and so $|C_{x,r}| = |V_2|!$ also in the general case.

We first consider the random variable $\mu(x, r)$ (describing the suffix of $m^*(x)$). Recall that $\mu(x, r)$ is defined by the following two-step random process. In the *first* step, one selects uniformly a pair $(\tau, \psi)$, over the set of pairs ($\{1, 2\} \times$ permutation), and sets $H = \psi(G_\tau)$. In the *second* step, one outputs (i.e., sets $\mu(x, r)$ to) $(\psi(G_\tau), \psi)$ if $v^*(x, r, H) = \tau$ (and ignores the $(\tau, \psi)$ pair otherwise). Hence, each graph $H$ (isomorphic to $G_2$) is generated, at the first step, by exactly $\text{aut}(G_2)$ different $(1, \cdot)$-pairs (i.e., the pairs $(1, \psi)$ satisfying $H = \psi(G_1)$) and by exactly $\text{aut}(G_2)$ different $(2, \cdot)$-pairs (i.e., the pairs $(2, \psi)$ satisfying $H = \psi(G_2)$). All these $2 \cdot \text{aut}(G_2)$ pairs yield the same graph $H$ and hence lead to the same value of $v^*(x, r, H)$. It follows that out of the $2 \cdot \text{aut}(G_2)$ pairs of the form $(\tau, \psi)$ that yield the graph $H = \psi(G_\tau)$, only the $\text{aut}(G_2)$ pairs satisfying $\tau = v^*(x, r, H)$ lead to an output. Hence, for each $H$ (that is isomorphic to $G_2$), the probability that $\mu(x, r) = (H, \cdot)$ equals $\text{aut}(G_2)/(|V_2|!)$. Furthermore, for each $H$ (that is isomorphic to $G_2$),

$$\Pr[\mu(x, r) = (H, \psi)] = \begin{cases} \frac{1}{|V_2|!} & \text{if} \quad H = \psi\left(G_{v^*(x,r,H)}\right) \\ 0 & \text{otherwise} \end{cases}$$

Hence $\mu(x, r)$ is uniformly distributed over $C_{x,r}$.

We now consider the random variable $v(x, r)$ (describing the suffix of the verifier's view in a "real interaction" with the prover). Recall that $v(x, r)$ is defined by selecting uniformly a permutation $\pi$ (over the set $V_2$) and setting $v(x, r) = (\pi(G_2), \pi)$ if $v^*(x, r, \pi(G_2)) = 2$, and $v(x, r) = (\pi(G_2), \pi \circ \phi)$ otherwise, where $\phi$ is the isomorphism between $G_1$ and $G_2$. Clearly, for each $H$ (that is isomorphic to $G_2$), the probability that $v(x, r) = (H, \cdot)$ equals $\text{aut}(G_2)/(|V_2|!)$. Furthermore, for each $H$ (that

is isomorphic to $G_2$),

$$\mathsf{Pr}[\nu(x,r) = (H,\psi)] = \begin{cases} \frac{1}{|V_2|!} & \text{if} \quad \psi = \pi \circ \phi^{2-v^*(x,r,H)} \\ 0 & \text{otherwise} \end{cases}$$

Observing that $H = \psi(G_{v^*(x,r,H)})$ if and only if $\psi = \pi \circ \phi^{2-v^*(x,r,H)}$, we conclude that $\mu(x,r)$ and $\nu(x,r)$ are identically distributed.

The claim follows. □

This completes the proof of Part 3 of the proposition. ■

### 4.3.3. Zero-Knowledge with Respect to Auxiliary Inputs

The definitions of zero-knowledge presented earlier fall short of what is required in practical applications, and consequently a minor modification should be used. We recall that these definitions guarantee that whatever can be efficiently computed after interaction with the prover on any *common input* can be efficiently computed *from the input itself*. However, in typical applications (e.g., when an interactive proof is used as a sub-protocol inside a larger protocol) the verifier interacting with the prover on common input $x$ may have some additional a priori information, encoded by a string $z$, that may assist it in its attempts to "extract knowledge" from the prover. This danger may become even more acute in the likely case in which $z$ is related to $x$. (For example, consider the protocol of Construction 4.3.8 and the case where the verifier has a priori information concerning an isomorphism between the input graphs.) What is typically required is that whatever can be efficiently computed from $x$ and $z$ after interaction with the prover on any common input $x$ can be efficiently computed from $x$ and $z$ (without any interaction with the prover). This requirement is formulated next using the augmented notion of interactive proofs presented in Definition 4.2.10.

**Definition 4.3.10 (Zero-Knowledge, Revisited):** *Let $(P,V)$ be an interactive proof for a language $L$ (as in Definition 4.2.10). Denote by $P_L(x)$ the set of strings $y$ satisfying the completeness condition with respect to $x \in L$ (i.e., $\mathsf{Pr}[\langle P(y), V(z)\rangle(x) = 1] \geq \frac{2}{3}$ for every $z \in \{0,1\}^*$). We say that $(P,V)$ is* **zero-knowledge with respect to auxiliary input** (or is **auxiliary-input zero-knowledge**) *if for every probabilistic polynomial-time interactive machine $V^*$ there exists a probabilistic algorithm $M^*$, running in time polynomial in the length of its first input, such that the following two ensembles are computationally indistinguishable (when the distinguishing gap is considered as a function of $|x|$):*

- $\{\langle P(y_x), V^*(z)\rangle(x)\}_{x\in L, z\in\{0,1\}^*}$ *for arbitrary $y_x \in P_L(x)$*
- $\{M^*(x,z)\}_{x\in L, z\in\{0,1\}^*}$

*Namely, for every probabilistic algorithm $D$ with running time polynomial in the length of the first input, for every polynomial $p(\cdot)$, and for all sufficiently long $x \in L$, all $y \in P_L(x)$, and $z \in \{0,1\}^*$, it holds that*

$$|\mathsf{Pr}[D(x,z,\langle P(y), V^*(z)\rangle(x)) = 1] - \mathsf{Pr}[D(x,z,M^*(x,z)) = 1]| < \frac{1}{p(|x|)}$$

In this definition, $y$ represents a priori information to the prover, whereas $z$ represents a priori information to the verifier. Both $y$ and $z$ may depend on the common input $x$; for example, if $y$ facilitates the proving task, then $y$ must depend on $x$ (e.g., in case $y$ is an $\mathcal{NP}$-witness for $x \in L \in \mathcal{NP}$). We stress that the local inputs (i.e., $y$ and $z$) may not be known, even in part, to the other party. We also stress that the auxiliary input $z$ (but not $y$) is also given to the distinguishing algorithm (which can be thought of as an extension of the verifier).

Recall that by Definition 4.2.10, saying that the interactive machine $V^*$ is probabilistic polynomial-time means that its running time is bounded by a polynomial in the length of the common input. Hence, the verifier program, the simulator, and the distinguishing algorithm all run in time polynomial in the length of $x$ (and not in time polynomial in the total length of all their inputs). This convention is essential in many respects (unless one explicitly bounds the length of the auxiliary input by a polynomial in the length of $x$; see Exercise 11). For example, having allowed the distinguishing algorithm to run in time proportional to the length of the auxiliary input would have collapsed computational zero-knowledge to perfect zero-knowledge (e.g., by considering verifiers that run in time polynomial in the common input, yet have huge auxiliary inputs of length exponential in the common input).

Definition 4.3.10 refers to computational zero-knowledge. A formulation of perfect zero-knowledge with respect to auxiliary input is straightforward. We remark that the perfect zero-knowledge proof for Graph Isomorphism, presented in Construction 4.3.8, is in fact perfect zero-knowledge with respect to auxiliary input. This fact follows easily by a minor augmentation to the simulator constructed in the proof of Proposition 4.3.9 (i.e., when invoking the verifier, the simulator should provide it the auxiliary input that is given to the simulator). In general, a demonstration of zero-knowledge can be extended to yield zero-knowledge with respect to auxiliary input whenever the simulator used in the original demonstration works by invoking the verifier's program as a black box (see Definition 4.5.10 in Section 4.5.4). All simulators presented in this book have this property.

### Advanced Comment: Implicit Non-Uniformity in Definition 4.3.10

The non-uniform nature of Definition 4.3.10 is captured by the fact that the distinguisher gets an auxiliary input. It is true that this auxiliary input is also given to both the verifier program and the simulator; however, if the auxiliary input is sufficiently long, then only the distinguisher can make use of its suffix (since the distinguisher may be determined after the polynomial-time bound of the simulator is fixed). It follows that the simulator guaranteed in Definition 4.3.10 produces output that is indistinguishable from the real interactions also by non-uniform polynomial-size circuits (see Definition 3.2.7). Namely, for every (even non-uniform) polynomial-size circuit family $\{C_n\}_{n\in\mathbb{N}}$, every polynomial $p(\cdot)$, all sufficiently large $n$'s, all $x \in L \cap \{0,1\}^n$, all $y \in P_L(x)$, and $z \in \{0,1\}^*$,

$$|\mathsf{Pr}[C_n(x, z, \langle P(y), V^*(z)\rangle(x)) = 1] - \mathsf{Pr}[C_n(x, z, M^*(x, z)) = 1]| < \frac{1}{p(|x|)}$$

Following is a sketch of the proof of this claim. We assume, to the contrary, that there exists a polynomial-size circuit family $\{C_n\}_{n\in\mathbb{N}}$ such that for infinitely many $n$'s there exist triples $(x, y, z)$ for which $C_n$ has a non-negligible distinguishing gap. We derive a contradiction by incorporating the description of $C_n$ together with the auxiliary input $z$ into a longer auxiliary input, denoted $z'$. This is done in such a way that both $V^*$ and $M^*$ have insufficient time to reach the description of $C_n$. For example, let $q(\cdot)$ be a polynomial bounding the running times of both $V^*$ and $M^*$. Assume, without loss of generality, that $|z| \leq q(n)$ (or else the rest of $z$, which is unreadable by both $V^*$ and $M^*$, can be ignored). Then we let $z'$ be the string that results by padding $z$ with blanks to a total length of $q(n)$ and appending the description of the circuit $C_n$ at its end (i.e., $z$ is a prefix of $z'$). Clearly, $M^*(x, z') = M^*(x, z)$ and $\langle P(y), V^*(z')\rangle(x) = \langle P(y), V^*(z)\rangle(x)$. On the other hand, by using a universal circuit-evaluating algorithm, we get a probabilistic polynomial-time algorithm $D$ such that $D(x, z', \alpha) = C_n(x, z, \alpha)$, and contradiction (to the hypothesis that $M^*$ produces output that is probabilistic polynomial-time-indistinguishable from the output of $(P, V^*)$) follows.

We mention that Definition 4.3.2 itself has some non-uniform flavor, since it requires indistinguishability for all but finitely many $x$'s. In contrast, a fully uniform analogue of the definition would require only that it be infeasible to find $x$'s on which the simulation would fail (with respect to some probabilistic polynomial-time distinguisher). That is, a fully uniform definition of zero-knowledge requires only that it be infeasible to find $x$'s on which a verifier can gain knowledge (and not that such instances do not exist at all). See further discussion in Section 4.4.2.4.

### Advanced Comment: Why Not Go for a Fully Non-Uniform Formulation?

An oversimplified version of Definition 4.3.10 allows the verifier to be modeled by a (non-uniform) family of (polynomial-size) circuits, and allows the same for the simulator. The non-uniform circuits are supposed to account for auxiliary inputs, and so these are typically omitted from such an oversimplified version. For example, one may require the following:

> *For every polynomial-size circuit family* $\{V_n\}_{n\in\mathbb{N}}$ (representing a possible verifier strategy machine) *there exists a polynomial-size circuit family* $\{M_n\}_{n\in\mathbb{N}}$ (representing a simulator) *such that the ensembles* $\{\langle P, V_{|x|}\rangle(x)\}_{x\in L}$ *and* $\{M_{|x|}(x)\}_{x\in L}$ *are indistinguishable by polynomial-size circuits*.

However, the impression that non-uniform circuits account for auxiliary inputs is wrong, and in general we find such oversimplified versions unsatisfactory. First, these versions do not guarantee an "effective" transformation of verifiers to simulators. Indeed, such a transformation is not required in Definition 4.3.10 either, but there the objects (i.e., machines) are of fixed size, whereas here we deal with infinite objects (i.e., circuit families). Thus, the level of "security" offered by the oversimplified definition is unsatisfactory. Second, the oversimplified version does not guarantee a relation between the size of the non-uniform part of the verifier and the corresponding part of the simulator, whereas in Definition 4.3.10 the only non-uniform part is the auxiliary input, which remains unchanged. Both issues arise when trying to prove a sequential-composition theorem for a non-constant number of iterations of zero-knowledge proof systems. Finally, we note

that the oversimplified version does *not* imply the basic version (i.e., Definition 4.3.2); consider, for example, a prover that on common input $x$ sends some hard-to-compute poly($|x|$)-bit-long string that depends only on $|x|$ (e.g., the prime-factorization of all integers in the interval $[2^{|x|}+1, \ldots, 2^{|x|}+|x|^3]$).

### 4.3.4. Sequential Composition of Zero-Knowledge Proofs

An intuitive requirement that a definition of zero-knowledge proofs must satisfy is that zero-knowledge proofs should be closed under sequential composition. Namely, if we execute one zero-knowledge proof after another, then the composed execution must be zero-knowledge. The same should remain valid even if we execute polynomially many proofs one after the other. Indeed, as will be shown shortly, the revised definition of zero-knowledge (i.e., Definition 4.3.10) satisfies this requirement. Interestingly, zero-knowledge proofs as defined in Definition 4.3.2 are not closed under sequential composition, and this fact is indeed another indication of the necessity of augmenting this definition (as done in Definition 4.3.10).

In addition to its conceptual importance, the sequential-composition lemma is an important tool in the design of zero-knowledge proof systems. Typically, such a proof system consists of many repetitions of an atomic zero-knowledge proof. Loosely speaking, the atomic proof provides *some* (but not much) statistical evidence for the validity of the claim. By repeating the atomic proof sufficiently many times, the confidence in the validity of the claim is increased. More precisely, the atomic proof offers a gap between the acceptance probabilities for strings in the language and strings outside the language. For example, in Construction 4.3.8, pairs of isomorphic graphs (i.e., inputs in $GI$) are accepted with probability 1, whereas pairs of non-isomorphic graphs (i.e., inputs not in $GI$) are accepted with probability at most $\frac{1}{2}$. By repeating the atomic proof, the gap between the two probabilities is further increased. For example, repeating the proof of Construction 4.3.8 $k$ times will yield a new interactive proof in which inputs in $GI$ are still accepted with probability 1, whereas inputs not in $GI$ are accepted with probability at most $\frac{1}{2^k}$. The sequential-composition lemma guarantees that if the atomic-proof system is zero-knowledge, then so is the proof system resulting from repeating the atomic proof polynomially many times.

Before we state the sequential-composition lemma, we remind the reader that the zero-knowledge property of an interactive proof is actually a property of the prover. Also, the prover is required to be zero-knowledge only on inputs in the language. Finally, we stress that when talking about zero-knowledge with respect to auxiliary input, we refer to all possible auxiliary inputs for the verifier.

**Lemma 4.3.11 (Sequential-Composition Lemma):** *Let $P$ be an interactive machine (i.e., a prover) that is* zero-knowledge with respect to auxiliary input *on some language $L$. Suppose that the last message sent by $P$, on input $x$, bears a special end-of-proof symbol. Let $Q(\cdot)$ be a polynomial, and let $P_Q$ be an interactive machine that, on common input $x$, proceeds in $Q(|x|)$ phases, each of them consisting of running $P$ on common input $x$. (We stress that in case $P$ is probabilistic, the interactive machine $P_Q$ uses independent coin tosses for*

*each of the $Q(|x|)$ phases.) Then $P_Q$ is zero-knowledge (with respect to auxiliary input) on $L$. Furthermore, if $P$ is perfect zero-knowledge (with respect to auxiliary input), then so is $P_Q$.*

The convention concerning the end-of-proof symbol is introduced for technical purposes (and is redundant in all known proof systems, and furthermore whenever the number of messages sent during the execution is easily computed from the common input). Clearly, every machine $P$ can be easily modified so that its last message will bear an appropriate symbol (as assumed earlier), and doing so will preserve the zero-knowledge properties of $P$ (as well as the completeness and soundness conditions).

The lemma ignores other aspects of repeating an interactive proof several times, specifically, the effect on the gap between the acceptance probabilities for inputs inside and outside of the language. The latter aspect of repeating an interactive proof system is discussed in Section 4.2.1.3 (see also Exercise 1).

***Proof:*** Let $V^*$ be an *arbitrary* probabilistic polynomial-time interactive machine interacting with the composed prover $P_Q$. Our task is to construct a (polynomial-time) simulator $M^*$ that will simulate the real interactions of $V^*$ with $P_Q$. Following is a very high level description of the simulation. The key idea is to simulate the real interaction on common input $x$ in $Q(|x|)$ phases corresponding to the phases of the operation of $P_Q$. Each phase of the operation of $P_Q$ is simulated using the simulator guaranteed for the atomic prover $P$. The information accumulated by the verifier in each phase is passed to the next phase using the auxiliary input.

(In the following exposition, we ignore the auxiliary input to the prover. This merely simplifies our notation. That is, instead of writing $P(y)$ and $P_Q(y)$, where $y$ is the prover's auxiliary input, we write $P$ and $P_Q$.)

The first step in carrying out this plan is to partition the execution of an arbitrary interactive machine $V^*$ into phases. The partition may not exist in the code of the program $V^*$, and yet it can be imposed on the executions of this program. This is done using the phase structure *of the prescribed prover* $P_Q$, which in turn is induced by the end-of-proof symbols. Hence, we claim that no matter how $V^*$ operates, the interaction of $V^*$ with $P_Q$ on common input $x$ can be captured by $Q(|x|)$ successive interactions of a related machine, denoted $V^{**}$, with $P$. Namely:

**Claim 4.3.11.1:** There exists a probabilistic polynomial-time $V^{**}$ such that for every common input $x$ and auxiliary input $z$, it holds that

$$\langle P_Q, V^*(z)\rangle(x) = Z^{(Q(|x|))}$$

where $Z^{(0)} \stackrel{\text{def}}{=} z$ and

$$Z^{(i+1)} \stackrel{\text{def}}{=} \left\langle P, V^{**}\left(Z^{(i)}\right)\right\rangle(x) \quad \text{for } i = 0, \ldots, Q(|x|) - 1$$

Namely, $Z^{(Q(|x|))}$ is a random variable describing the output of $V^{**}$ after $Q(|x|)$ successive interactions with $P$, on common input $x$, where the auxiliary input of $V^{**}$ in the $i+1$ interaction equals the output of $V^{**}$ after the $i$th interaction (i.e., $Z^{(i)}$).

**Proof:** Intuitively, $V^{**}$ captures the functionality of $V^*$ during each single phase. By the *functionality of $V^*$ during a phase* we mean the way $V^*$ transforms the content of its work tapes at the beginning of the phase to their content at the end of the phase, as well as the way $V^*$ determines the messages it sends during this phase. Indeed, this transformation depends on the messages received during the current phase. We stress that we can effect this transformation without "reverse-engineering" (the code of) $V^*$, but rather by emulating its execution while monitoring all its tapes. Details follow.

In order to facilitate this process, we first modify $V^*$ so that all its "essential" activities refer only to its work tapes. Machine $V^*$ can be slightly modified so that it starts its execution by reading the common input, the random input, and the auxiliary input into special regions in its work tape and never accesses the aforementioned read-only tapes again. Likewise, $V^*$ is modified so that it starts each active period[11] (see Definition 4.2.2) by reading the current incoming message from the communication tape to a special region in the work tape (and never accesses the incoming-message tape again during this period). Actually, this description should be modified so that $V^*$ copies only a polynomially long (in the common input) prefix of each of these tapes, the polynomial being the one bounding the running time of (the original) $V^*$.

> (Formally speaking, given an arbitrary $V^*$, we construct a machine $W^*$ that emulates $V^*$ in a way that satisfies the foregoing conditions; that is, $W^*$ will satisfy these conditions even if $V^*$ does not. Machine $W^*$ will have several extra work tapes that will be designated as the common-input, random-input, auxiliary-input, and incoming-communication tapes of $V^*$. Machine $W^*$ will start by copying its own common input, random input, and auxiliary input to the corresponding designated tapes. Likewise, $W^*$ will start each active period by copying the current incoming message from its own communication tape to the corresponding designated tape (i.e., the incoming-communication tape of $V^*$). After completing these copying activities, $W^*$ just emulates the execution of $V^*$. Clearly, $W^*$ satisfies the requirements postulated. Thus, formally speaking, whenever we later refer to $V^*$, we mean $W^*$.)

Consider an interaction of $V^*(z)$ with $P_Q$, on common input $x$. By the foregoing modification, the interaction consists of $Q(|x|)$ phases, so that, except in the first phase, machine $V^*$ never accesses its common-input, random-input, and auxiliary-input tapes. (In the first phase, machine $V^*$ starts by copying the content of these tapes into its work tapes and never accesses the former tapes again.) Likewise, when executing the current phase, machine $V^*$ does not try to read messages of *previous* phases from its incoming-communication tape (yet it may read these "old" messages from storage in its work tapes). Considering the content of the *work tapes of $V^*$* at the end of *each* of the $Q(|x|)$ phases (of interaction with $P_Q$) naturally leads us to the construction of $V^{**}$.

We are now finally ready present the construction of $V^{**}$: On common input $x$ and auxiliary input $z'$, machine $V^{**}$ starts by copying $z'$ into the work tape of

[11] Recall that an *active period* during an execution of an interactive machine $M$ consists of the steps $M$ takes from the time the last message is received up to the time at which $M$ completes sending its response message.

$V^*$. Next, machine $V^{**}$ emulates a *single phase* of the interaction of $V^*$ with $P_Q$ (on input $x$), starting with the foregoing contents of the work tape of $V^*$ (instead of starting with an empty work tape). The emulated machine $V^*$ regards the communication tapes of machine $V^{**}$ as its own communication tapes. When $V^*$ completes the interaction in the current phase, machine $V^{**}$ terminates by outputting the current contents of the work tape of $V^*$. Thus, when $z'$ equals a possible content of the work tape of $V^*$ after $i \geq 1$ phases, the emulated $V^*$ behaves as in the $i+1$ phase, and the output of $V^{**}$ is distributed as the content of the work tape of $V^*$ after $i+1$ phases. Actually, the foregoing description should be slightly modified to deal with the first phase in the interaction with $P_Q$ (i.e., the case $i = 0$ ignored earlier). Specifically, $V^{**}$ copies $z'$ into the work tape of $V^*$ only if $z'$ encodes the content of the work tape of $V^*$ (we assume, without loss of generality, that the content of the work tape of $V^*$ is encoded differently from the encoding of an auxiliary input for $V^*$). In case $z'$ encodes an auxiliary input to $V^*$, machine $V^{**}$ invokes $V^*$ on an empty work tape, and $V^*$ regards the readable tapes of $V^{**}$ (i.e., common-input tape, random-input tape, and auxiliary-input tape) as its own. Observe that $Z^{(1)} \stackrel{\text{def}}{=} \langle P, V^{**}(z)\rangle(x)$ describes the content of the work tape of $V^*$ after the first phase (in the interaction with $P_Q$ on common input $x$ and auxiliary input $z$). Likewise, for every $i = 2, \ldots, Q(|x|)$, the random variable $Z^{(i)} \stackrel{\text{def}}{=} \langle P, V^{**}(Z^{(i-1)})\rangle(x)$ describes the content of the work tape of $V^*$ after $i$ phases. The claim follows. □

Because $V^{**}$ is a polynomial-time interactive machine (with auxiliary input) interacting with $P$, it follows by the lemma's hypothesis that there exists a probabilistic machine that simulates these interactions in time polynomial in the length of the first input. Let $M^{**}$ denote this simulator.[12] Then for every probabilistic polynomial-time (in $x$) algorithm $D$, every polynomial $p(\cdot)$, all sufficiently long $x \in L$, and all $z \in \{0,1\}^*$, we have

$$|\mathsf{Pr}[D(x, z, \langle P, V^{**}(z)\rangle(x)) = 1] - \mathsf{Pr}[D(x, z, M^{**}(x, z)) = 1]| < \frac{1}{p(|x|)} \tag{4.1}$$

We are now ready to present the construction of a simulator $M^*$ that simulates the "real" output of $V^*$ after interaction with $P_Q$. We can assume, without loss of generality, that the output of $V^*$ equals the content of its work tapes at the end of the interaction (since the output of $V^*$ is probabilistic polynomial-time-computable from the content of its work tapes at that time). Machine $M^*$ uses the simulator $M^{**}$ (as a black box).

**The simulator $M^*$:** On input $(x, z)$, machine $M^*$ sets $z^{(0)} = z$ and proceeds in $Q(|x|)$ phases. In the $i$th phase, machine $M^*$ computes $z^{(i)}$ by running machine

[12] Recall that in the case of perfect zero-knowledge (see Definition 4.3.1) machine $M^{**}$ may halt with no real output (but rather with output $\perp$). However, by sufficiently many repetitions, we can make the probability of this event exponentially vanishing. In the rest of the exposition, we assume for simplicity that $M^{**}$ always halts with output.

$M^{**}$ on input $(x, z^{(i-1)})$. After $Q(|x|)$ phases are completed, machine $M^*$ stops outputting $z^{(Q(|x|))}$.

Clearly, machine $M^*$, as constructed here, runs in time polynomial in its first input. It is left to show that machine $M^*$ indeed produces output that is computationally indistinguishable from the output of $V^*$ (after interacting with $P_Q$). Namely:

**Claim 4.3.11.2:** For every probabilistic algorithm $D$ with running time polynomial in its first input, every polynomial $p(\cdot)$, all sufficiently long $x \in L$, and all $z \in \{0,1\}^*$, we have

$$|\mathsf{Pr}[D(x, z, \langle P_Q, V^*(z)\rangle(x)) = 1] - \mathsf{Pr}[D(x, z, M^*(x, z)) = 1]| < \frac{1}{p(|x|)}$$

Furthermore, if $P$ is perfect zero-knowledge, then $\langle P_Q, V^*(z)\rangle(x)$ and $M^*(x, z)$ are identically distributed.

**Proof sketch:** We use a hybrid argument (see Chapter 3). In particular, we define the following $Q(|x|)+1$ hybrids. The $i$th hybrid, $0 \le i \le Q(|x|)$, corresponds to the following random process. We first let $V^{**}$ interact with $P$ for $i$ phases, starting with common input $x$ and auxiliary input $z$, and denote by $Z^{(i)}$ the output of $V^{**}$ after the $i$th phase. We next repeatedly iterate $M^{**}$ for the remaining $Q(|x|) - i$ phases. In both cases, we use the output of the previous phase as auxiliary input to the new phase. Formally, the hybrid $H^{(i)}$ is defined as follows:

$$H^{(i)}(x, z) \stackrel{\text{def}}{=} M^{**}_{Q(|x|)-i}\left(x, Z^{(i)}\right)$$

where the $Z^{(j)}$'s are as defined in Claim 4.3.11.1, and where

$$M^{**}_0(x, z') \stackrel{\text{def}}{=} z' \quad \text{and} \quad M^{**}_k(x, z') \stackrel{\text{def}}{=} M^{**}_{k-1}(x, M^{**}(x, z')) \quad \text{for } k = 1, \ldots, Q(|x|) - i$$

By Claim 4.3.11.1, the $Q(|x|)$ hybrid (i.e., $H^{(Q(|x|))}(x, z) = Z^{(Q(|x|))}$) equals $\langle P_Q, V^*(z)\rangle(x)$. On the other hand, recalling the construction of $M^*$, we see that the zero hybrid (i.e., $H^{(0)}(x, z) = M^{**}_{Q(|x|)}(x, z)$) equals $M^*(x, z)$. Hence, all that is required to complete the proof is to show that all pairs of two adjacent hybrids are computationally indistinguishable (as this will imply that the extreme hybrids, $H^{(Q(|x|))}$ and $H^{(0)}$, are also indistinguishable). To this end, we rewrite the $i$ and $i-1$ hybrids as follows:

$$\begin{aligned} H^{(i)}(x, z) &= M^{**}_{Q(|x|)-i}\left(x, Z^{(i)}\right) \\ &= M^{**}_{Q(|x|)-i}\left(x, \left\langle P, V^{**}\left(Z^{(i-1)}\right)\right\rangle(x)\right) \\ H^{(i-1)}(x, z) &= M^{**}_{Q(|x|)-(i-1)}\left(x, Z^{(i-1)}\right) \\ &= M^{**}_{Q(|x|)-i}\left(x, M^{**}\left(x, Z^{(i-1)}\right)\right) \end{aligned}$$

where $Z^{(i-1)}$ is as defined in Claim 4.3.11.1.

Using an averaging argument, it follows that if an algorithm $D$ distinguishes the hybrids $H^{(i)}(x, z)$ and $H^{(i-1)}(x, z)$, then there exists a $z'$ (in the support of $Z^{(i-1)}$) such that algorithm $D$ distinguishes the random variables $M^{**}_{Q(|x|)-i}(x, \langle P, V^{**}(z')\rangle(x))$ and $M^{**}_{Q(|x|)-i}(x, M^{**}(x, z'))$ at least as well. (In all cases, $D$ is also given $x$ and $z$.) Using algorithms $M^{**}$ and $D$, we get a new algorithm $D'$, with running time polynomially related to the former algorithms, that distinguishes the random variables $(x, z, i, z', \langle P, V^{**}(z')\rangle(x))$ and $(x, z, i, z', M^{**}(x, z'))$ at least as well. Specifically, on input $(x, (z, i, z'), \alpha)$ (where $\alpha$ is taken either from $\langle P, V^{**}(z')\rangle(x)$ or from $M^{**}(x, z')$), algorithm $D'$ invokes $D$ on input $(x, z, M^{**}_{Q(|x|)-i}(x, \alpha))$ and outputs whatever $D$ does. Clearly,

$$\begin{aligned}&|\mathsf{Pr}[D'(x, (z, i, z'), \langle P, V^{**}(z')\rangle(x)) = 1] - \mathsf{Pr}[D'(x, (z, i, z'), M^{**}(x, z')) = 1]| \\ &\quad \geq\ |\mathsf{Pr}[D(x, z, H^{(i)}(x, z)) = 1] - \mathsf{Pr}[D(x, z, H^{(i-1)}(x, z)) = 1]|\end{aligned}$$

Note that $D'$ uses additional input $(x, z, i, z')$, whereas it distinguishes $\langle P, V^{**}(z')\rangle(x)$ from $M^{**}(x, z')$. This does not fit the definition of a distinguisher for (auxiliary-input) zero-knowledge, as the latter is to be given only $(x, z')$ and the string to be distinguished. In other words, we have actually constructed a non-uniform $D' = D'_{i,z}$ that, depending on $i$ and $z$, distinguishes $\langle P, V^{**}(z')\rangle(x)$ from $M^{**}(x, z')$. Still, in the case of perfect zero-knowledge, letting $D$ be an arbitrary function (rather than an efficient algorithm), this suffices for contradicting the hypothesis that $M^{**}$ perfectly simulates $(P, V^{**})$. For the case of computational zero-knowledge, we use the fact that the definition of auxiliary-input zero-knowledge implies robustness against non-uniform (polynomial-size) distinguishers, and we note that $D'_{i,z}$ falls into this category (provided that $D$ also does). Thus, in both cases, contradiction (to the hypothesis that $M^{**}$ simulates $(P, V^{**})$) follows. □

**Further details concerning the proof of Claim 4.3.11.2:** At this stage (assuming the reader has gone through Chapter 3), the reader should be able to transform the foregoing proof sketch into a detailed proof. The main thing that is missing is the detail concerning the way in which an algorithm contradicting the hypothesis that $M^{**}$ is a simulator for $(P, V^{**})$ is derived from an algorithm contradicting the statement of Claim 4.3.11.2. These details are presented next.

We assume, to the contradiction, that there exists a probabilistic polynomial-time algorithm $D$ and a polynomial $p(\cdot)$ such that for infinitely many $x \in L$, there exists $z \in \{0, 1\}^*$ such that

$$|\mathsf{Pr}[D(x, z, \langle P_Q, V^*(z)\rangle(x)) = 1] - \mathsf{Pr}[D(x, z, M^*(x, z)) = 1]| > \frac{1}{p(|x|)}$$

It follows that for every such $x$ and $z$, there exists an $i \in \{1, \ldots, Q(|x|)\}$ such that

$$\left|\mathsf{Pr}\left[D\left(x, z, H^{(i)}(x, z)\right) = 1\right] - \mathsf{Pr}\left[D\left(x, z, H^{(i-1)}(x, z)\right) = 1\right]\right| > \frac{1}{Q(|x|) \cdot p(|x|)}$$

where the hybrid $H^{(j)}$'s are as defined earlier. Denote $\varepsilon(n) \stackrel{\text{def}}{=} 1/(Q(n) \cdot p(n))$. Combining, as before, the definitions of the $i$ and $i-1$ hybrids with an averaging argument, it follows that for each such $x$, $z$, and $i$, there exists a $z'$ such that

$$\Big|\mathsf{Pr}\big[D\big(x, z, M^{**}_{Q(|x|)-i}(x, \langle P, V^{**}(z')\rangle(x))\big) = 1\big] - \mathsf{Pr}\big[D\big(x, z, M^{**}_{Q(|x|)-i}(x, M^{**}(x, z'))\big) = 1\big]\Big| > \varepsilon(|x|)$$

This almost leads to the desired contradiction. Namely, the random variables $(x, z', \langle P, V^{**}(z')\rangle(x))$ and $(x, z', M^{**}(x, z'))$ can be distinguished using the algorithms $D$ and $M^{**}$, *provided we "know" $i$ and $z$*. But how do we get to "know" $i$ and $z$? The problem is resolved using the fact, pointed out earlier, that the output of $M^{**}$ should be indistinguishable from the interactions of $V^{**}$ with $P$ even with respect to non-uniform polynomial-size circuits. Thus, in order to derive a contradiction, it suffices to construct a non-uniform distinguisher that incorporates $i$ and $z$ in its description. Alternatively, we can incorporate $i$ and $z$ in a new auxiliary input, denoted $z''$, so that $z'$ is a prefix of $z''$, but $z''$ looks the same as $z'$ to both $V^*$ and $M^*$. Next we shall follow the latter alternative.

Let $T$ denote a polynomial upper bound on the time-complexity of both $V^*$ and $M^*$. Note that for every $z'$ determined for a pair $(x, z)$, as before, it must hold that $|z'| \leq T(|x|)$ (since $z'$ is a possible record of a partial computation of $M^*(x, z)$). Let $z'' = (z' \flat^{T(|x|)-|z'|}, i, z)$, where $i$ and $z$ are as before (and $\flat$ denotes the blank symbol of the work tape). We construct a probabilistic polynomial-time algorithm $D'$ that distinguishes $(x, z'', \langle P, V^{**}(z'')\rangle(x))$ and $(x, z'', M^{**}(x, z''))$ for the aforementioned $(x, z, i, z')$-tuples. On input $(x, z'', \alpha)$ (where $\alpha$ supposedly is in either $\langle P, V^{**}(z'')\rangle(x) = \langle P, V^{**}(z')\rangle(x)$ or $M^{**}(x, z'') = M^{**}(x, z')$), algorithm $D'$ first extracts $i$ and $z$ from $z''$. Next, it uses $M^{**}$ to compute $\beta = M^{**}_{Q(|x|)-i}(x, \alpha)$. Finally, $D'$ halts with output $D(x, z, \beta)$. Using the fact that $V^{**}$ and $M^{**}$ cannot distinguish the auxiliary inputs $z'$ and $z''$, we have

$$\begin{aligned}
&|\mathsf{Pr}[D'(x, z'', \langle P, V^{**}(z'')\rangle(x)) = 1] - \mathsf{Pr}[D'(x, z'', M^{**}(x, z'')) = 1]| \\
&\quad = |\mathsf{Pr}[D'(x, z'', \langle P, V^{**}(z')\rangle(x)) = 1] - \mathsf{Pr}[D'(x, z'', M^{**}(x, z')) = 1]| \\
&\quad = \Big|\mathsf{Pr}\big[D\big(x, z, M_{Q(|x|)-i}(x, \langle P, V^{**}(z')\rangle(x))\big) = 1\big] \\
&\qquad - \mathsf{Pr}\big[D\big(x, z, M_{Q(|x|)-i}(x, M^{**}(x, z'))\big) = 1\big]\Big| \\
&\quad > \varepsilon(|x|)
\end{aligned}$$

Contradiction (to the hypothesis that $M^{**}$ is a simulator for $(P, V^{**})$) follows. □

The lemma follows. ■

## And What About Parallel Composition?

Unfortunately, we cannot prove that zero-knowledge (even with respect to auxiliary input) is preserved under parallel composition. Furthermore, there exist (auxiliary-input) zero-knowledge proofs that when played twice in parallel do yield knowledge (to a "cheating verifier"). For further details, see Section 4.5.4.

The fact that zero-knowledge is not preserved under parallel composition of protocols is indeed bad news. One might even say that this fact is a conceptually annoying

phenomenon. We disagree with that assessment. Our feeling is that the behavior of protocols and "games" under parallel composition is, in general (i.e., not only in the context of zero-knowledge), a much more complex issue than their behavior under sequential composition: in fact, in several other cases (e.g., computationally sound proofs, proofs of knowledge, and multi-prover proof systems; see Sections 4.8, 4.7, and 4.11, respectively), parallel composition lags behind sequential composition. Furthermore, the only advantage of parallel composition over sequential composition is in efficiency. Hence, we do not consider the non-closure under parallel composition to be a fundamental weakness of the formulation of zero-knowledge. Yet the "non-closure" of zero-knowledge motivates the search for alternative (related) notions that are preserved under parallel composition. (Such notions may be either weaker or stronger than the formulation of zero-knowledge.) For further details, the reader is referred to Sections 4.9 and 4.6.

## 4.4. Zero-Knowledge Proofs for $\mathcal{NP}$

This section presents the main thrust of this chapter, namely, a method for constructing zero-knowledge proofs for *every* language in $\mathcal{NP}$. The importance of this method stems from its generality, which is the key to its many applications. Specifically, almost all statements one might wish to prove in practice can be encoded as claims concerning membership in languages in $\mathcal{NP}$. In particular, the construction of zero-knowledge proofs for such statements provides a tool for "forcing" parties to properly execute any given protocol.

The method for constructing zero-knowledge proofs for $\mathcal{NP}$ languages makes essential use of the concept of *bit commitment*. Hence, we start with a presentation of the latter concept. (A reader who wishes to have more of the flavor of this application of commitment schemes before studying them is encouraged to read Section 4.4.2.1 first.)

### 4.4.1. Commitment Schemes

Commitment schemes are basic ingredients in many cryptographic protocols. They are used to enable a party to commit itself to a value while keeping it secret. In a later stage the commitment is "opened," and it is guaranteed that the "opening" can yield only a single value determined in the committing phase. Commitment schemes are the digital analogues of non-transparent sealed envelopes. By putting a note in such an envelope, a party commits itself to the content of the note while keeping the content secret.

#### 4.4.1.1. Definition

Loosely speaking, a commitment scheme is an efficient *two-phase* two-party protocol through which one party, called the *sender*, can commit itself to a *value* such that the following two conflicting requirements are satisfied.

1. *Secrecy* (or *hiding*): At the end of the first phase, the other party, called the *receiver*, does not gain any knowledge of the sender's value. This requirement has to be satisfied even if the receiver tries to cheat.
2. *Unambiguity* (or *binding*): Given the transcript of the interaction in the first phase, there exists at most one value that the receiver can later (i.e., in the second phase) accept as a legal "opening" of the commitment. This requirement has to be satisfied even if the sender tries to cheat.

In addition, one should require that the protocol be *viable*, in the sense that if both parties follow it, then at the end of the second phase the receiver gets the value committed to by the sender. The first phase is called the *commit phase*, and the second phase is called the *reveal phase*. We are requiring that the commit phase yield no knowledge (at least no knowledge of the sender's value) to the receiver, whereas the commit phase does "bind" the sender to a unique value (in the sense that in the reveal phase the receiver can accept only this value). We stress that the protocol is efficient in the sense that the predetermined programs of both parties can be implemented in probabilistic polynomial time. Without loss of generality, the reveal phase may consist of merely letting the sender send, to the receiver, the original value and the sequence of random coin tosses that it has used during the commit phase. The receiver will accept the value if and only if the supplied information matches its transcript of the interaction in the commit phase. The latter convention leads to the following definition (which refers explicitly only to the commit phase).

**Definition 4.4.1 (Bit-Commitment Scheme):** *A* **bit-commitment scheme** *is a pair of probabilistic polynomial-time interactive machines, denoted* $(S, R)$ (*for* sender *and* receiver)*, satisfying the following:*

- Input specification: *The* common input *is an integer n presented in unary (serving as the security parameter).*
  *The* private input *to the sender is a bit, denoted v.*
- Secrecy (or hiding): *The receiver (even when deviating arbitrarily from the protocol) cannot distinguish a commitment to* 0 *from a commitment to* 1*. Namely, for every probabilistic polynomial-time machine $R^*$ interacting with S, the probability ensembles describing the output of $R^*$ in the two cases, namely $\{\langle S(0), R^*\rangle(1^n)\}_{n\in\mathbb{N}}$ and $\{\langle S(1), R^*\rangle(1^n)\}_{n\in\mathbb{N}}$, are computationally indistinguishable.*
- Unambiguity (or binding): *Preliminaries to the requirement*:

  ***1.*** A receiver's view of an interaction *with the sender, denoted* $(r, \overline{m})$*, consists of the random coins used by the receiver* $(r)$ *and the sequence of messages received from the sender* $(\overline{m})$.

  ***2.*** *Let* $\sigma \in \{0, 1\}$*. We say that a receiver's view (of such interaction),* $(r, \overline{m})$*, is a* **possible $\sigma$-commitment** *if there exists a string s such that* $\overline{m}$ *describes the messages received by R when R uses local coins r and interacts with machine S that uses local coins s and has input* $(\sigma, 1^n)$.

  *(Using the notation of Definition 4.3.3, we say that* $(r, \overline{m})$ *is a* possible $\sigma$-commitment *if* $(r, \overline{m}) = \text{view}^{S(\sigma,1^n,s)}_{R(1^n,r)}$*.)*

*3. We say that the receiver's view $(r, \overline{m})$ is* **ambiguous** *if it is both a possible* 0*-commitment and a possible* 1*-commitment.*

*The* unambiguity requirement *asserts that for all but a negligible fraction of the coin tosses of the receiver there exists no sequence of messages (from the sender) that together with these coin tosses forms an ambiguous receiver view. Namely, for all but a negligible fraction of the $r \in \{0,1\}^{\mathrm{poly}(n)}$ there is no $\overline{m}$ such that $(r, \overline{m})$ is ambiguous.*

The secrecy requirement is a computational one. On the other hand, the *unambiguity requirement* has an information-theoretic flavor (i.e., it does not refer to computational powers) and is sometimes referred to as *perfect* (or *absolute*). Thus, a commitment scheme as in Definition 4.4.1 is sometimes referred to as *computationally hiding* and *perfectly binding*. A dual definition, requiring information-theoretic secrecy and computational infeasibility of creating ambiguities, is presented in Section 4.8.2. (The latter is referred to as *perfectly hiding* and *computationally binding*.)

**Canonical Reveal Phase.** The secrecy requirement refers explicitly to the situation at the end of the commit phase. On the other hand, we stress that the unambiguity requirement implicitly assumes that the reveal phase takes the following form:

1. The sender sends to the receiver its initial private input $v$ and the random coins $s$ it has used in the commit phase.
2. The receiver verifies that $v$ and $s$ (together with the coins $(r)$ used by $R$ in the commit phase) indeed yield the messages that $R$ has received in the commit phase. Verification is done in polynomial time (by running the programs $S$ and $R$).

Note that the viability requirement (i.e., asserting that if both parties follow the protocol, then at the end of the reveal phase the receiver gets $v$) is implicitly satisfied by this convention.

### 4.4.1.2. Construction Based on Any One-Way Permutation

*Some* public-key encryption scheme can be used as a commitment scheme. This can be done by having the sender generate a pair of keys and use the public key together with the encryption of a value as its commitment to the value. In order to satisfy the unambiguity requirement, the underlying public-key scheme needs to satisfy additional requirements (i.e., the set of legitimate public keys should be efficiently recognizable, and an encryption relative to legitimate public keys should have a unique decryption). In any case, public-key encryption schemes have additional properties not required of commitment schemes, and their existence seems to require stronger intractability assumptions. (Thus, we consider the aforementioned approach to be "conceptually wrong.") An alternative construction, presented next, uses any *one-way permutation*. Specifically, we use a one-way permutation, denoted $f$, and a hard-core predicate for it, denoted $b$ (see Section 2.5). In fact, we can use any 1-1 one-way function.

**Construction 4.4.2 (Simple Bit Commitment):** *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a function, and let $b : \{0,1\}^* \to \{0,1\}$ be a predicate.*

1. Commit phase: *To commit to value $v \in \{0,1\}$ (using security parameter $n$), the sender uniformly selects $s \in \{0,1\}^n$ and sends the pair $(f(s), b(s) \oplus v)$ to the receiver.*
2. (Canonical) reveal phase: *In the reveal phase, the sender reveals the bit $v$ and the string $s$ used in the commit phase. The receiver accepts the value $v$ if $f(s) = \alpha$ and $b(s) \oplus v = \sigma$, where $(\alpha, \sigma)$ is the receiver's view of the commit phase.*

**Proposition 4.4.3:** *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a 1-1 one-way function, and let $b : \{0,1\}^* \to \{0,1\}$ be a hard-core predicate of $f$. Then the protocol presented in Construction 4.4.2 constitutes a bit-commitment scheme.*

***Proof:*** The secrecy requirement follows directly from the fact that $b$ is a hard-core of $f$. The unambiguity requirement follows from the 1-1 property of $f$. In fact, there exists *no* ambiguous receiver view. Namely, for each possible receiver view $(\alpha, \sigma)$, there is a unique $s \in \{0,1\}^{|\alpha|}$ such that $f(s) = \alpha$, and hence a unique $v \in \{0,1\}$ such that $b(s) \oplus v = \sigma$. ■

### 4.4.1.3. Construction Based on Any One-Way Function

We now present a construction of a bit-commitment scheme that is based on the weakest assumption possible: the existence of one-way functions. Proving that the assumption is indeed minimal is left as an exercise (i.e., Exercise 13). On the other hand, by the results in Chapter 3 (specifically, Theorems 3.3.3 and 3.5.12), the existence of one-way functions implies the existence of pseudorandom generators expanding $n$-bit strings into $3n$-bit strings. We shall use such a pseudorandom generator in the construction presented next.

We start by motivating the construction. Let $G$ be a pseudorandom generator satisfying $|G(s)| = 3 \cdot |s|$. Assume that $G$ has the property that the sets $\{G(s) : s \in \{0,1\}^n\}$ and $\{G(s) \oplus 1^{3n} : s \in \{0,1\}^n\}$ are disjoint, where $\alpha \oplus \beta$ denotes the bit-by-bit XOR of the strings $\alpha$ and $\beta$. Then the sender can commit itself to the bit $v$ by uniformly selecting $s \in \{0,1\}^n$ and sending the message $G(s) \oplus v^{3n}$ ($v^k$ denotes the all-$v$ $k$-bit-long string). Unfortunately, the foregoing assumption cannot be justified in general, and a slightly more complex variant is required. The key observation is that for most strings $r \in \{0,1\}^{3n}$ the sets $\{G(s) : s \in \{0,1\}^n\}$ and $\{G(s) \oplus r : s \in \{0,1\}^n\}$ are disjoint. Such a string $r$ is called *good*. This observation suggests the following protocol: The receiver uniformly selects $r \in \{0,1\}^{3n}$, hoping that it is good, and sends $r$ to the sender. Having received $r$, the sender commits to the bit $v$ by uniformly selecting $s \in \{0,1\}^n$ and sending the message $G(s)$ if $v = 0$, and $G(s) \oplus r$ otherwise.

**Construction 4.4.4 (Bit Commitment under General Assumptions):** *Let $G : \{0,1\}^* \to \{0,1\}^*$ be a function such that $|G(s)| = 3 \cdot |s|$ for all $s \in \{0,1\}^*$.*

1. Commit phase:
   - *To receive a commitment to a bit (using security parameter $n$), the receiver uniformly selects $r \in \{0,1\}^{3n}$ and sends it to the sender.*

- *Upon receiving the message $r$ (from the receiver), the sender commits to value $v \in \{0,1\}$ by uniformly selecting $s \in \{0,1\}^n$ and sending $G(s)$ if $v = 0$, and $G(s) \oplus r$ otherwise.*

**2.** (Canonical) reveal phase: *In the reveal phase, the sender reveals the string $s$ used in the commit phase. The receiver accepts the value 0 if $G(s) = \alpha$ and accepts the value 1 if $G(s) \oplus r = \alpha$, where $(r, \alpha)$ is the receiver's view of the commit phase.*

Such a definition of the (canonical) reveal phase allows the receiver to accept both values, but we shall show that that happens very rarely (if at all).

**Proposition 4.4.5:** *If $G$ is a pseudorandom generator, then the protocol presented in Construction 4.4.4 constitutes a bit-commitment scheme.*

***Proof:*** The secrecy requirement follows the fact that $G$ is a pseudorandom generator. Specifically, let $U_k$ denote the random variable uniformly distributed on strings of length $k$. Then for every $r \in \{0,1\}^{3n}$, the random variables $U_{3n}$ and $U_{3n} \oplus r$ are identically distributed. Hence, if it is feasible to find an $r \in \{0,1\}^{3n}$ such that $G(U_n)$ and $G(U_n) \oplus r$ are computationally distinguishable, then either $U_{3n}$ and $G(U_n)$ are computationally distinguishable or $U_{3n} \oplus r$ and $G(U_n) \oplus r$ are computationally distinguishable. In either case, contradiction to the pseudorandomness of $G$ follows.

We now turn to the unambiguity requirement. Following the motivating discussion, we call $r \in \{0,1\}^{3n}$ *good* if the sets $\{G(s) : s \in \{0,1\}^n\}$ and $\{G(s) \oplus r : s \in \{0,1\}^n\}$ are disjoint. We say that $r \in \{0,1\}^{3n}$ *yields a collision between the seeds $s_1$ and $s_2$* if $G(s_1) = G(s_2) \oplus r$. Clearly, $r$ is good if it does not yield a collision between any pair of seeds. On the other hand, there is at most one string $r$ that yields a collision between a given pair of seeds $(s_1, s_2)$; that is, $r = G(s_1) \oplus G(s_2)$. Because there are at most $\binom{2^n}{2} < 2^{2n}$ possible pairs of seeds, fewer than $2^{2n}$ strings will yield collisions between pairs of seeds, and so all the other $3n$-bit-long strings are good. It follows that with probability at least $1 - 2^{2n-3n}$ the receiver selects a good string, in which case its view $(r, \alpha)$ is unambiguous (since if $r$ is good and $G(s_1) = \alpha$ holds for some $s_1$, then $G(s_2) \neq \alpha \oplus r$ must hold for all $s_2$'s). The unambiguity requirement follows. ■

### 4.4.1.4. Extensions

The definition and the constructions of bit-commitment schemes are easily extended to general commitment schemes, enabling the sender to commit to a string rather than to a single bit. Actually, for the purposes of the rest of this section, we need a commitment scheme by which one can commit to a ternary value. Extending the definition and the constructions to deal with this special case is even more straightforward.

In the rest of this section we shall need commitment schemes with a seemingly stronger secrecy requirement than defined earlier. Specifically, instead of requiring secrecy with respect to all polynomial-time machines, we require secrecy with respect to all (not necessarily uniform) families of polynomial-size circuits. Assuming the

existence of non-uniformly one-way functions (see Definition 2.2.6 in Section 2.2), commitment schemes with non-uniform secrecy can be constructed, using the same construction as in the uniform case. Thus, we have the following:

**Theorem 4.4.6:** *Suppose there exist non-uniformly one-way functions (as in Definition 2.2.6). Then there exists a bit-commitment scheme (as in Definition 4.4.1) for which the secrecy condition also holds with respect to polynomial-size circuits.*

### 4.4.2. Zero-Knowledge Proof of Graph Coloring

Presenting a zero-knowledge proof system for one $\mathcal{NP}$-complete language implies the existence of a zero-knowledge proof system for every language in $\mathcal{NP}$. This intuitively appealing statement does require a proof, which we postpone to a later stage. In the current section we present a zero-knowledge proof system for one $\mathcal{NP}$-complete language, specifically Graph 3-Colorability. This choice is indeed arbitrary.

The language *Graph 3-Coloring*, denoted $G3C$, consists of all simple (finite) graphs (i.e., no parallel edges or self-loops)[13] that can be *vertex-colored* using three colors such that no two adjacent vertices are given the same color. Formally, a graph $G = (V, E)$ is *3-colorable* if there exists a mapping $\phi : V \to \{1, 2, 3\}$ such that $\phi(u) \neq \phi(v)$ for every $(u, v) \in E$.

#### 4.4.2.1. Motivating Discussion

The idea underlying the zero-knowledge proof system for $G3C$ is to break the proof of the claim that a graph is 3-colorable into polynomially many *pieces* arranged in *templates* so that each template by itself will yield no knowledge and yet all the templates put together will guarantee the validity of the main claim. Suppose that the prover generates such pieces of information, places each of them in a separate sealed and non-transparent envelope, and allows the verifier to open and inspect the pieces participating in one of the templates. Then certainly the verifier gains no knowledge in the process, yet its confidence in the validity of the claim (that the graph is 3-colorable) increases. A concrete implementation of this abstract idea follows.

To prove that the graph $G = (V, E)$ is 3-colorable, the prover generates a random 3-coloring of the graph, denoted $\phi$ (actually a random relabeling of a fixed coloring will do). The color of each single vertex constitutes a piece of information concerning the 3-coloring. The set of templates corresponds to the set of edges (i.e., each pair $(\phi(u), \phi(v))$, where $(u, v) \in E$, constitutes a template to the claim that $G$ is 3-colorable). Each single template (being merely a random pair of distinct elements in $\{1, 2, 3\}$) will yield no knowledge. However, if all the templates are OK (i.e., each contains a pair of distinct elements in $\{1, 2, 3\}$), then the graph must be 3-colorable. Consequently, graphs that

[13] A simple finite graph is a pair $(V, E)$, where $V$ is a finite set and $E$ is a set of 2-subsets of $V$; that is, $E \subseteq \{e \subseteq V : |e| = 2\}$. The elements of $V$ are called **vertices**, and the elements of $E$ are called **edges**. Although each edge is an unordered pair of two elements in $V$, we use the ordered-pair notation $(u, v) \in E$ rather than the notation $\{u, v\} \in E$. For $e = (u, v) \in E$, we say that $u$ and $v$ are the endpoints of $e$ and that $u$ is adjacent to $v$.

are not 3-colorable must contain at least one bad template and hence will be rejected with noticeable probability. Following is an abstract description of the resulting zero-knowledge interactive proof system for $G3C$.

- *Common input:* A simple graph $G = (V, E)$.
- *Prover's first step:* Let $\psi$ be a 3-coloring of $G$. The prover selects a random permutation $\pi$ over $\{1, 2, 3\}$ and sets $\phi(v) \stackrel{\text{def}}{=} \pi(\psi(v))$ for each $v \in V$. Hence, the prover forms a random relabeling of the 3-coloring $\psi$. The prover sends the verifier a sequence of $|V|$ locked and non-transparent boxes such that the $v$th box contains the value $\phi(v)$.
- *Verifier's first step:* The verifier uniformly selects an edge $(u, v) \in E$ and sends it to the prover.
- *Motivating remark:* The verifier asks to inspect the colors of vertices $u$ and $v$.
- *Prover's second step:* The prover sends to the verifier the keys to boxes $u$ and $v$.
- *Verifier's second step:* The verifier opens boxes $u$ and $v$ and accepts if and only if they contain two different elements in $\{1, 2, 3\}$.

Clearly, if the input graph is 3-colorable, then the prover can cause the verifier to always accept. On the other hand, if the input graph is not 3-colorable, then any content placed in the boxes must be invalid on at least one edge, and consequently the verifier will reject with probability at least $1/|E|$. Hence, the foregoing protocol exhibits a noticeable gap in the acceptance probabilities between the case of inputs in $G3C$ and the case of inputs not in $G3C$. The zero-knowledge property follows easily in this abstract setting, because one can simulate the real interaction by placing a random pair of different colors in the boxes indicated by the verifier. We stress that this simple argument will not be possible in the digital implementation, because the boxes are not totally unaffected by their contents (but rather are affected, yet in an indistinguishable manner). Finally, we remark that confidence in the validity of the claim (that the input graph is 3-colorable) can be increased by sequentially applying the foregoing proof sufficiently many times. (In fact, if the boxes are perfect, as assumed, then one can also use parallel repetitions; however, the boxes are not perfect in the digital implementation presented next.)

### 4.4.2.2. The Interactive Proof

We now turn to the digital implementation of the abstract protocol. In this implementation the boxes are implemented by a commitment scheme. Namely, for each box, we invoke an *independent* execution of the commitment scheme. This will enable us to execute the reveal phase for only some of the commitments, a property that is crucial to our scheme. For simplicity of exposition, we use the simple commitment scheme presented in Construction 4.4.2 (or, more generally, any *one-way-interaction* commitment scheme). We denote by $C_s(\sigma)$ the commitment of the sender, using coins $s$, to the (ternary) value $\sigma$.

**Construction 4.4.7 (A Zero-Knowledge Proof for Graph 3-Coloring):**

- Common input: *A simple (3-colorable) graph* $G = (V, E)$. *Let* $n \stackrel{\text{def}}{=} |V|$ *and* $V = \{1, \ldots, n\}$.

- Auxiliary input to the prover: *A 3-coloring of $G$, denoted $\psi$.*
- Prover's first step (P1): *The prover selects a random permutation $\pi$ over $\{1, 2, 3\}$ and sets $\phi(v) \stackrel{\text{def}}{=} \pi(\psi(v))$ for each $v \in V$. The prover uses the commitment scheme to commit itself to the color of each of the vertices. Namely, the prover uniformly and independently selects $s_1, \ldots, s_n \in \{0,1\}^n$, computes $c_i = C_{s_i}(\phi(i))$ for each $i \in V$, and sends $c_1, \ldots, c_n$ to the verifier.*
- Verifier's first step (V1): *The verifier uniformly selects an edge $(u, v) \in E$ and sends it to the prover.*
- Prover's second step (P2): *Without loss of generality, we can assume that the message received from the verifier is an edge, denoted $(u, v)$. (Otherwise, the prover sets $(u, v)$ to be some predetermined edge of $G$.) The prover uses the (canonical) reveal phase of the commitment scheme in order to reveal the colors of vertices $u$ and $v$ to the verifier. Namely, the prover sends $(s_u, \phi(u))$ and $(s_v, \phi(v))$ to the verifier.*
- Verifier's second step (V2): *The verifier checks whether or not the values corresponding to commitments $u$ and $v$ were revealed correctly and whether or not these values are different. Namely, upon receiving $(s, \sigma)$ and $(s', \tau)$, the verifier checks whether or not $c_u = C_s(\sigma)$, $c_v = C_{s'}(\tau)$, and $\sigma \neq \tau$ (and both $\sigma$ and $\tau$ are in $\{1, 2, 3\}$). If all conditions hold, then the verifier accepts. Otherwise it rejects.*

*Let us denote this prover's program by $P_{G3C}$.*

We stress that the program of the verifier and that of the prover can be implemented in probabilistic polynomial time. In the case of the prover's program, this property is made possible by use of the *auxiliary input* to the prover. As we shall later see, the foregoing protocol constitutes a weak interactive proof for $G3C$. As usual, the confidence can be increased (i.e., the error probability can be decreased) by sufficiently many successive applications. However, the mere existence of an interactive proof for $G3C$ is obvious (since $G3C \in \mathcal{NP}$). The punch line is that this protocol is zero-knowledge (also with respect to auxiliary input). Using the sequential-composition-lemma (Lemma 4.3.11), it follows that polynomially many sequential applications of this protocol will preserve the zero-knowledge property.

**Proposition 4.4.8:** *Suppose that the commitment scheme used in Construction 4.4.7 satisfies the (non-uniform) secrecy and the unambiguity requirements. Then Construction 4.4.7 constitutes an auxiliary-input zero-knowledge (generalized) interactive proof for $G3C$.*

For further discussion of Construction 4.4.7, see Section 4.4.2.4.

### 4.4.2.3. The Simulator: Proof of Proposition 4.4.8

We first prove that Construction 4.4.7 constitutes a weak interactive proof for $G3C$. Assume first that the input graph is indeed 3-colorable. Then if the prover follows the specified program, the verifier will always accept (i.e., accept with probability 1). On the other hand, if the input graph is not 3-colorable, then no matter what the prover

does, the $n$ commitments sent in Step P1 cannot correspond to a 3-coloring of the graph (since such coloring does not exist). We stress that the unique correspondence of commitments to values is guaranteed by the unambiguity property of the commitment scheme. It follows that there must exist an edge $(u, v) \in E$ such that $c_u$ and $c_v$, sent in Step P1, are not commitments to two *different* elements of $\{1, 2, 3\}$. Hence, no matter how the prover behaves, the verifier will reject with probability at least $1/|E|$. Therefore, there is a noticeable (in the input length) gap between the acceptance probabilities in the case in which the input is in $G3C$ and in the case in which it is not.

We shall now show that $P_{G3C}$, the prover program specified in Construction 4.4.7, is indeed zero-knowledge for $G3C$. The claim is proved without reference to auxiliary-input (to the verifier), but an extension of the argument to auxiliary-input zero-knowledge is straightforward. Again, we use the alternative formulation of zero-knowledge (i.e., Definition 4.3.3) and show how to simulate $V^*$'s view of the interaction with $P_{G3C}$ for every probabilistic polynomial-time interactive machine $V^*$. As in the case of the Graph Isomorphism proof system (i.e., Construction 4.3.8), it is easy to simulate the verifier's view of the interaction with $P_{G3C}$, *provided that* the verifier follows the specified program. However, we need to simulate the view of the verifier in the general case (in which the verifier uses an *arbitrary* polynomial-time interactive program). Following is an overview of our simulation (i.e., of our construction of a simulator $M^*$ for an arbitrary $V^*$).

The simulator $M^*$ incorporates the code of the interactive program $V^*$. On input a graph $G = (V, E)$, the simulator $M^*$ (not having access to a 3-coloring of $G$) first uniformly and independently selects $n$ values $e_1, \ldots, e_n \in \{1, 2, 3\}$ and constructs a commitment to each of them. (These $e_i$'s constitute a "pseudo-coloring" of the graph in which the endpoints of each edge will be colored differently with probability $\frac{2}{3}$.) In doing so, the simulator behaves very differently from $P_{G3C}$, but nevertheless the sequence of commitments thus generated is computationally indistinguishable from the sequence of commitments to a valid 3-coloring sent by $P_{G3C}$ in Step P1. If $V^*$, when given the commitments generated by the simulator, asks to inspect an edge $(u, v)$ such that $e_u \neq e_v$, then the simulator can indeed answer correctly, and in doing so it completes a simulation of the verifier's view of the interaction with $P_{G3C}$. However, if $V^*$ asks to inspect an edge $(u, v)$ such that $e_u = e_v$, then the simulator has no way to answer correctly, and we let it halt with output $\perp$. We stress that we do not assume that the simulator "knows" a priori which edge the verifier $V^*$ will ask to inspect. The validity of the simulator stems from a different source. If the verifier's request were oblivious of the prover's commitment, then with probability $\frac{2}{3}$ the verifier would have asked to inspect an edge that was properly colored. Using the secrecy property of the commitment scheme, it follows that the verifier's request is "almost oblivious" of the values in the commitments. The zero-knowledge claim follows (yet, with some effort). Further details follow. We start with a detailed description of the simulator.

**Simulator $M^*$.** On input a graph $G = (V, E)$, where $n = |V|$, the simulator $M^*$ proceeds as follows:

1. *Setting the random tape of $V^*$*: Let $q(\cdot)$ denote a polynomial bounding the running time of $V^*$. The simulator $M^*$ starts by uniformly selecting a string $r \in \{0,1\}^{q(n)}$ to be used as the content of the local random tape of $V^*$.
2. *Simulating the prover's first step* (P1): The simulator $M^*$ uniformly and independently selects $n$ values $e_1, \ldots, e_n \in \{1,2,3\}$ and $n$ random strings $s_1, \ldots, s_n \in \{0,1\}^n$ to be used for committing to these values. The simulator computes, for each $i \in V$, a commitment $d_i = C_{s_i}(e_i)$.
3. *Simulating the verifier's first step* (V1): The simulator $M^*$ initiates an execution of $V^*$ by placing $G$ on $V^*$'s common-input tape, placing $r$ (selected in Step 1) on $V^*$'s local random tape, and placing the sequence $(d_1, \ldots, d_n)$ (constructed in Step 2) on $V^*$'s incoming-message tape. After executing a polynomial number of steps of $V^*$, the simulator can read the outgoing message of $V^*$, denoted $m$. Again, we assume without loss of generality that $m \in E$ and let $(u,v) = m$. (Actually, $m \notin E$ is treated as in Step P2 of $P_{G3C}$; namely, $(u,v)$ is set to be some predetermined edge of $G$.)
4. *Simulating the prover's second step* (P2): If $e_u \neq e_v$, then the simulator halts with output $(G, r, (d_1, \ldots, d_n), (s_u, e_u, s_v, e_v))$.
5. *Failure of the simulation:* Otherwise (i.e., $e_u = e_v$), the simulator halts with output $\perp$.

Using the hypothesis that $V^*$ is polynomial-time, it follows that so is the simulator $M^*$. It is left to show that $M^*$ outputs $\perp$ with probability at most $\frac{1}{2}$ and that, conditioned on not outputting $\perp$, the simulator's output is computationally indistinguishable from the verifier's view in a "real interaction with $P_{G3C}$." The proposition will follow by running the simulator $n$ times and outputting the first output different from $\perp$. We now turn to proving the two claims.

**Claim 4.4.8.1:** For every sufficiently large graph $G = (V, E)$, the probability that $M^*(G) = \perp$ is bounded above by $\frac{1}{2}$.

(Actually, a stronger claim can be proved: For every polynomial $p$ and all sufficiently large graphs $G = (V, E)$, the probability that $M^*(G) = \perp$ is bounded above by $\frac{1}{3} + \frac{1}{p(|V|)}$.)

**Proof:** Let us denote by $p_{u,v}(G, r, (e_1, \ldots, e_n))$ the probability, taken over all the choices of $s_1, \ldots, s_n \in \{0,1\}^n$, that $V^*$, on input $G$, random coins $r$, and prover message $(C_{s_1}(e_1), \ldots, C_{s_n}(e_n))$, replies with the message $(u,v)$. We assume, for simplicity, that $V^*$ always answers with an edge of $G$ (since otherwise its message is treated as if it were an edge of $G$). We first claim that for every sufficiently large graph $G = (V, E)$, every $r \in \{0,1\}^{q(n)}$, every edge $(u,v) \in E$, and every two sequences $\alpha, \beta \in \{1,2,3\}^n$, it holds that

$$|p_{u,v}(G, r, \alpha) - p_{u,v}(G, r, \beta)| \leq \frac{1}{2|E|} \tag{4.2}$$

Actually, we can prove the following sub-claim.

**Request Obliviousness Sub-Claim:** For every polynomial $p(\cdot)$, every sufficiently large graph $G = (V, E)$, every $r \in \{0,1\}^{q(n)}$, every edge $(u,v) \in E$, and

every two sequences $\alpha, \beta \in \{1,2,3\}^n$, it holds that

$$|p_{u,v}(G,r,\alpha) - p_{u,v}(G,r,\beta)| \leq \frac{1}{p(n)}$$

The Request Obliviousness Sub-Claim is proved using the non-uniform secrecy of the commitment scheme. The reader should be able to fill out the details of such a proof at this stage. (Nevertheless, a proof of the sub-claim follows.)

**Proof of the Request Obliviousness Sub-Claim:** Assume, on the contrary, that there exists a polynomial $p(\cdot)$ and an infinite sequence of integers such that for each integer $n$ (in the sequence) there exists an $n$-vertex graph $G_n = (V_n, E_n)$, a string $r_n \in \{0,1\}^{q(n)}$, an edge $(u_n, v_n) \in E_n$, and two sequences $\alpha_n, \beta_n \in \{1,2,3\}^n$ such that

$$\left|p_{u_n,v_n}(G_n,r_n,\alpha_n) - p_{u_n,v_n}(G_n,r_n,\beta_n)\right| > \frac{1}{p(n)}$$

We construct a circuit family $\{A_n\}$ by letting $A_n$ incorporate the interactive machine $V^*$, the graph $G_n$, and $r_n, u_n, v_n, \alpha_n, \beta_n$, all being as in the contradiction hypothesis. On input $y$ (supposedly a sequence of commitments to either $\alpha_n$ or $\beta_n$), circuit $A_n$ runs $V^*$ (on input $G_n$, coins $r_n$, and prover's message $y$) and outputs 1 if and only if $V^*$ replies with $(u_n, v_n)$. Clearly, $\{A_n\}$ is a (non-uniform) family of polynomial-size circuits. The key observation is that $A_n$ distinguishes commitments to $\alpha_n$ from commitments to $\beta_n$, since

$$\mathsf{Pr}\left[A_n\left(C_{U_n^{(1)}}(e_1),\ldots,C_{U_n^{(n)}}(e_n)\right)=1\right] = p_{u_n,v_n}(G_n,r_n,(e_1,\ldots,e_n))$$

where the $U_n^{(i)}$'s denote, as usual, independent random variables uniformly distributed over $\{0,1\}^n$. Contradiction to the (non-uniform) secrecy of the commitment scheme follows by a standard hybrid argument (which relates the indistinguishability of sequences of commitments to the indistinguishability of single commitments).

Returning to the proof of Claim 4.4.8.1, we now use this sub-claim to upper-bound the probability that the simulator outputs $\perp$. The intuition is simple: Because the requests of $V^*$ are almost oblivious of the values to which the simulator has committed itself, it is unlikely that $V^*$ will request to inspect an illegally colored edge more often than it would if it had made the request without looking at the commitment. Thus, $V^*$ asks to inspect an illegally colored edge with probability approximately $\frac{1}{3}$, and so $\mathsf{Pr}[M^*(G) = \perp] \approx \frac{1}{3}$. A more rigorous (but straightforward) analysis follows.

Let $M_r^*(G)$ denote the output of machine $M^*$ on input $G$, conditioned on the event that it chooses the string $r$ in Step 1. We remind the reader that $M_r^*(G) = \perp$ only in the case in which the verifier, on input $G$, random tape $r$, and a commitment to some pseudo-coloring $(e_1,\ldots,e_n)$, asks to inspect an edge $(u,v)$ that is illegally colored (i.e., $e_u = e_v$). Let $E_{(e_1,\ldots,e_n)}$ denote the set of edges $(u,v) \in E$ that are illegally colored (i.e., satisfy $e_u = e_v$) with respect to $(e_1,\ldots,e_n)$. Then, fixing an arbitrary $r$ and considering all possible choices of $\overline{e} = (e_1,\ldots,e_n) \in \{1,2,3\}^n$, we have

$$\mathsf{Pr}[M_r^*(G) = \perp] = \sum_{\overline{e}\in\{1,2,3\}^n} \frac{1}{3^n} \cdot \sum_{(u,v)\in E_{\overline{e}}} p_{u,v}(G,r,\overline{e})$$

(Recall that $p_{u,v}(G, r, \overline{e})$ denotes the probability that the verifier will ask to inspect $(u, v)$ when given a sequence of random commitments to the values $\overline{e}$.) Define $B_{u,v}$ to be the set of $n$-tuples $(e_1, \ldots, e_n) \in \{1, 2, 3\}^n$ satisfying $e_u = e_v$. Clearly, $|B_{u,v}| = 3^{n-1}$, and

$$\begin{aligned}\{(\overline{e}, (u, v)) : \overline{e} \in \{1, 2, 3\}^n \,\&\, (u, v) \in E_{\overline{e}}\} &= \{(\overline{e}, (u, v)) : \overline{e} \in \{1, 2, 3\}^n \,\&\, e_u = e_v\} \\ &= \{(\overline{e}, (u, v)) : (u, v) \in E \,\&\, \overline{e} \in B_{u,v}\}\end{aligned}$$

By straightforward calculation we get

$$\begin{aligned}\mathsf{Pr}[M^*_r(G) = \perp] &= \frac{1}{3^n} \cdot \sum_{\overline{e} \in \{1,2,3\}^n} \sum_{(u,v) \in E_{\overline{e}}} p_{u,v}(G, r, \overline{e}) \\ &= \frac{1}{3^n} \cdot \sum_{(u,v) \in E} \sum_{\overline{e} \in B_{u,v}} p_{u,v}(G, r, \overline{e}) \\ &\leq \frac{1}{3^n} \cdot \sum_{(u,v) \in E} |B_{u,v}| \cdot \left( p_{u,v}(G, r, (1, \ldots, 1)) + \frac{1}{2|E|} \right) \\ &= \frac{1}{6} + \frac{1}{3} \cdot \sum_{(u,v) \in E} p_{u,v}(G, r, (1, \ldots, 1)) \\ &= \frac{1}{6} + \frac{1}{3}\end{aligned}$$

where the inequality is due to Eq. (4.2). The claim follows. □

For simplicity, we assume in the sequel that on common input $G \in G3C$ the prover gets the *lexicographically first* 3-coloring of $G$ as auxiliary input. This enables us to omit the auxiliary input to $P_{G3C}$ (which is now implicit in the common input) from the notation. The argument is easily extended to the general case where $P_{G3C}$ gets an arbitrary 3-coloring of $G$ as auxiliary input.

**Claim 4.4.8.2:** The ensemble consisting of the output of $M^*$ on input $G = (V, E) \in G3C$, conditioned on it not being $\perp$, is computationally indistinguishable from the ensemble $\{\text{view}^{P_{G3C}}_{V^*}(G)\}_{G \in G3C}$. Namely, for every probabilistic polynomial-time algorithm $A$, every polynomial $p(\cdot)$, and all sufficiently large graphs $G = (V, E)$,

$$\left| \mathsf{Pr}[A(M^*(G)) = 1 \mid M^*(G) \neq \perp] - \mathsf{Pr}\left[A\left(\text{view}^{P_{G3C}}_{V^*}(G)\right) = 1\right] \right| < \frac{1}{p(|V|)}$$

We stress that these ensembles are very different (i.e., the statistical distance between them is very close to the maximum possible), and yet they are computationally indistinguishable. Actually, we can prove that these ensembles are indistinguishable also by (non-uniform) families of polynomial-size circuits. At first glance it seems that Claim 4.4.8.2 follows easily from the secrecy property of the commitment scheme. Indeed, Claim 4.4.8.2 is proved using the secrecy property of the commitment scheme, but the proof is more complex than one might anticipate at first glance. The difficulty lies in the fact that the foregoing ensembles consist not only of commitments to values but also of openings of *some*

of the values. Furthermore, the choice of which commitments are to be opened depends on the entire sequence of commitments. (We take advantage of the fact that the number of such openings is a constant.)

**Proof:** Let $m^*(G)$ denote the distribution of $M^*(G)$ conditioned on $M^*(G) \neq \perp$. For any algorithm $A$, we denote the distinguishing gap of $A$, regarding the ensembles in the claim, by $\varepsilon_A(G)$; that is,

$$\varepsilon_A(G) \stackrel{\text{def}}{=} \left|\mathsf{Pr}[A(m^*(G)) = 1] - \mathsf{Pr}\left[A\left(\text{view}_{V^*}^{P_{G3C}}(G)\right) = 1\right]\right| \tag{4.3}$$

Our goal is to prove that for every probabilistic polynomial-time algorithm $A$, the value of $\varepsilon_A(G)$ is negligible as a function of the number of vertices in $G$. Recall that for $G = (V, E)$ both $m^*(G)$ and $\text{view}_{V^*}^{P_{G3C}}(G)$ are sequences of the form $(r, (\alpha_1, \ldots, \alpha_{|V|}), (u, v), (s_u, \sigma_u, s_v, \sigma_v))$, where $r \in \{0,1\}^{q(|V|)}$, $(u, v) \in E$, $\sigma_u \neq \sigma_v \in \{1, 2, 3\}$, $\alpha_u = C_{s_u}(\sigma_u)$, and $\alpha_v = C_{s_v}(\sigma_v)$. In both cases, the pair $(u, v)$ is called the *verifier's request*.

Given a graph $G = (V, E)$, we define for each edge $(u, v) \in E$ two random variables describing, respectively, the output of $M^*$ and the view of $V^*$ in a real interaction in the case in which the verifier's request equals $(u, v)$. Specifically:

- $\mu_{u,v}(G)$ describes $M^*(G)$ (equivalently, $m^*(G)$) conditioned on $M^*(G)$ (equivalently, $m^*(G)$) having the verifier's request equal to $(u, v)$.
- $\nu_{u,v}(G)$ describes $\text{view}_{V^*}^{P_{G3C}}(G)$ conditioned on $\text{view}_{V^*}^{P_{G3C}}(G)$ having the verifier's request equal to $(u, v)$.

Let $p_{u,v}(G)$ denote the probability that $m^*(G)$ has the verifier's request equal to $(u, v)$. Similarly, let $q_{u,v}(G)$ denote the probability that $\text{view}_{V^*}^{P_{G3C}}(G)$ has the verifier's request equal to $(u, v)$.

Assume, contrary to the claim, that the ensembles mentioned in the claim are computationally distinguishable. Then one of the following cases must occur.

**Case 1:** There is a non-negligible difference between the probabilistic profile of the request of $V^*$ when interacting with $P_{G3C}$ and that of the verifier's request in the output represented by $m^*(G)$. Formally, there exists a polynomial $p(\cdot)$ and an infinite sequence of integers such that for each integer $n$ (in the sequence) there exists an $n$-vertex graph $G_n = (V_n, E_n)$ and an edge $(u_n, v_n) \in E_n$ such that

$$\left|p_{u_n,v_n}(G_n) - q_{u_n,v_n}(G_n)\right| > \frac{1}{p(n)}$$

Otherwise, for every polynomial $p'$, all but finitely many $G$'s, and all edges $(u, v)$ in such $G = (V, E)$, it holds that

$$|p_{u,v}(G) - q_{u,v}(G)| \leq \frac{1}{p'(|V|)} \tag{4.4}$$

**Case 2:** An algorithm distinguishing the foregoing ensembles also does so conditioned on $V^*$ making a particular request. Furthermore, this request occurs with non-negligible probability that is about the same for both ensembles. Formally, there exists a probabilistic polynomial-time algorithm $A$, a polynomial $p(\cdot)$, and

an infinite sequence of integers such that for each integer $n$ (in the sequence) there exists an $n$-vertex graph $G_n = (V_n, E_n)$ and an edge $(u_n, v_n) \in E_n$ such that the following conditions hold:

- $q_{u_n,v_n}(G_n) > \frac{1}{p(n)}$
- $|p_{u_n,v_n}(G_n) - q_{u_n,v_n}(G_n)| < \frac{1}{3 \cdot p(n)^2}$
- $|\mathsf{Pr}[A(\mu_{u_n,v_n}(G_n)) = 1] - \mathsf{Pr}[A(\nu_{u_n,v_n}(G_n)) = 1]| > \frac{1}{p(n)}$

The fact that if Case 1 does not hold, then Case 2 does hold follows by breaking the probability space according to the edge being revealed. The obvious details follow:

> Consider an algorithm $A$ that distinguishes the simulator's output from the real interaction for infinitely many graphs $G = (V, E)$, where the distinguishing gap is a reciprocal of a polynomial in the size of $G$; i.e., $\varepsilon_A(G) > 1/\text{poly}(|V|)$. Let $\text{req}_{u,v}(\alpha)$ denote the event that *in transcript $\alpha$, the verifier's request equals* $(u, v)$. Then there must be an edge $(u, v)$ in $G$ such that
>
> $$\Big|\mathsf{Pr}[A(m^*(G)) = 1 \,\&\, \text{req}_{u,v}(m^*(G))] - \mathsf{Pr}\Big[A\Big(\text{view}_{V^*}^{P_{G3C}}(G)\Big) = 1 \,\&\, \text{req}_{u,v}\Big(\text{view}_{V^*}^{P_{G3C}}(G)\Big)\Big]\Big| \geq \frac{\varepsilon_A(G)}{|E|}$$
>
> Note that
>
> $$\begin{aligned} p_{u,v}(G) &= \mathsf{Pr}[\text{req}_{u,v}(m^*(G))] \\ q_{u,v}(G) &= \mathsf{Pr}\Big[\text{req}_{u,v}\Big(\text{view}_{V^*}^{P_{G3C}}(G)\Big)\Big] \\ \mathsf{Pr}[A(\mu_{u,v}(G)) = 1] &= \mathsf{Pr}[A(m^*(G)) = 1 \mid \text{req}_{u,v}(m^*(G))] \\ \mathsf{Pr}[A(\nu_{u,v}(G)) = 1] &= \mathsf{Pr}\Big[A\Big(\text{view}_{V^*}^{P_{G3C}}(G)\Big) = 1 \,\Big|\, \text{req}_{u,v}\Big(\text{view}_{V^*}^{P_{G3C}}(G)\Big)\Big] \end{aligned}$$
>
> Thus, omitting $G$ from some of the notations, we have
>
> $$|p_{u,v} \cdot \mathsf{Pr}[A(\mu_{u,v}(G)) = 1] - q_{u,v} \cdot \mathsf{Pr}[A(\nu_{u,v}(G)) = 1]| \geq \frac{\varepsilon_A(G)}{|E|}$$
>
> Setting $p(|V|) \stackrel{\text{def}}{=} \frac{2|E|}{\varepsilon_A(G)}$ (i.e., so that $\frac{\varepsilon_A(G)}{|E|} = \frac{2}{p(|V|)}$) and using Eq. (4.4) (with $p' = 3p^2$), we get $|p_{u,v} - q_{u,v}| < \frac{1}{3p(|V|)^2}$ and
>
> $$|q_{u,v} \cdot \mathsf{Pr}[A(\mu_{u,v}(G)) = 1] - q_{u,v} \cdot \mathsf{Pr}[A(\nu_{u,v}(G)) = 1]| > \frac{1}{p(|V|)}$$
>
> for all but finitely many of these $G$'s. Thus, both $q_{u,v} > 1/p(|V|)$ and
>
> $$|\mathsf{Pr}[A(\mu_{u,v}(G)) = 1] - \mathsf{Pr}[A(\nu_{u,v}(G)) = 1]| > 1/p(|V|)$$
>
> follow.

Case 1 can immediately be discarded because it leads easily to contradiction (to the non-uniform secrecy of the commitment scheme): The idea is to use the Request Obliviousness Sub-Claim appearing in the proof of Claim 4.4.8.1. Details are omitted. We are thus left with Case 2.

We are now going to show that Case 2 also leads to contradiction. To this end we shall construct a circuit family that will distinguish commitments to different sequences of values. Interestingly, neither of these sequences will equal

the sequence of commitments generated either by the prover or by the simulator. Following is an overview of the construction. The $n$th circuit gets a sequence of $3n$ commitments and produces from it a sequence of $n$ commitments (part of which is a subsequence of the input). When the input sequence to the circuit is taken from one distribution, the circuit generates a subsequence corresponding to the sequence of commitments generated by the prover. Likewise, when the input sequence (to the circuit) is taken from the other distribution, the circuit will generate a subsequence corresponding to the sequence of commitments generated by the simulator. We stress that the circuit does so without knowing from which distribution the input is taken. After generating an $n$-long sequence, the circuit feeds it to $V^*$, and depending on $V^*$'s behavior the circuit may feed part of the sequence to algorithm $A$ (mentioned in Case 2). Following is a detailed description of the circuit family.

Let us denote by $\psi_n$ the (lexicographically first) 3-coloring of $G_n = (V_n, E_n)$ used by the prover, where $V_n = \{1, \ldots, n\}$. We construct a circuit family, denoted $\{A_n\}$, by letting $A_n$ incorporate the interactive machine $V^*$, the "distinguishing" algorithm $A$, the graph $G_n$, the 3-coloring $\psi_n$, and the edge $(u_n, v_n)$, all being as guaranteed in Case 2. The input to circuit $A_n$ will be a sequence of commitments to $3n$ values, each in $\{1, 2, 3\}$. The circuit will distinguish commitments to a uniformly chosen $3n$-long sequence from commitments to the fixed sequence $1^n 2^n 3^n$ (i.e., the sequence consisting of $n$ 1-values, followed by $n$ 2-values, followed by $n$ 3-values). Following is a description of the operation of $A_n$. In this description, for $e \in \{1, 2, 3\}$, we denote by $C(e)$ the random variable obtained by uniformly selecting $s \in \{0, 1\}^n$ and outputting $C_s(e)$. We extend this notation to sequences over $\{1, 2, 3\}$ (i.e., $C(e_1, \ldots, e_t) = C(e_1), \ldots, C(e_t)$, where independent randomization is used in each commitment).

**Operation of $A_n$:** On input $y = (y_1, \ldots, y_{3n})$ (where each $y_i$ supposedly is a commitment to an element of $\{1, 2, 3\}$), the circuit $A_n$ proceeds as follows:

- $A_n$ first selects uniformly a permutation $\pi$ over $\{1, 2, 3\}$ and computes $\phi(i) = \pi(\psi_n(i))$ for each $i \in V_n$.

  Note that $(\phi(u_n), \phi(v_n))$ is uniformly distributed among the six possible pairs of distinct elements of $\{1, 2, 3\}$.
- For each $i \in V_n \setminus \{u_n, v_n\}$, the circuit sets $c_i = y_{\phi(i)\cdot n - n + i}$ (i.e., $c_i = y_i$ if $\phi(i) = 1$, $c_i = y_{n+i}$ if $\phi(i) = 2$, and $c_i = y_{2n+i}$ if $\phi(i) = 3$).

  Note that each $y_j$ is used at most once and that $2n + 2$ of the $y_j$'s are not used at all.
- The circuit uniformly selects $s_{u_n}, s_{v_n} \in \{0, 1\}^n$ and sets $c_{u_n} = C_{s_{u_n}}(\phi(u_n))$ and $c_{v_n} = C_{s_{v_n}}(\phi(v_n))$.

  In case $y$ is taken from the distribution $C(1^n 2^n 3^n)$, the sequence $c_1, \ldots, c_n$ just formed is distributed exactly as the sequence of commitments sent by the prover in Step P1. On the other hand, suppose that $y$ is uniformly distributed among all possible commitments to all possible $3n$-long sequences (i.e., $y$ is formed by uniformly selecting $\alpha \in \{1, 2, 3\}^{3n}$ and outputting $C(\alpha)$). Then the sequence $c_1, \ldots, c_n$ just formed is distributed exactly as the sequence of commitments formed by the

simulator in Step 2, conditioned on vertices $u_n$ and $v_n$ being assigned different colors.

- The circuit initiates an execution of $V^*$ by placing $G_n$ on $V^*$'s common-input tape, placing a uniformly selected $r \in \{0,1\}^{q(n)}$ on $V^*$'s local random tape, and placing the sequence $(c_1, \ldots, c_n)$ on $V^*$'s incoming-message tape. The circuit reads the outgoing message of $V^*$, denoted $m$.
- If $m \neq (u_n, v_n)$, then the circuit outputs 0.
- Otherwise (i.e., $m = (u_n, v_n)$), the circuit invokes algorithm $A$ and outputs

$$A\big(G_n, r, (c_1, \ldots, c_n), \big(s_{u_n}, \phi(u_n), s_{v_n}, \phi(v_n)\big)\big)$$

Clearly, the size of $A_n$ is polynomial in $n$. We now evaluate the distinguishing ability of $A_n$. Let us first consider the probability that circuit $A_n$ will output 1 on input a random commitment to the sequence $1^n 2^n 3^n$. The reader can easily verify that the sequence $(c_1, \ldots, c_n)$ constructed by circuit $A_n$ is distributed identically to the sequence sent by the prover in Step P1. Hence, recalling some of the notations introduced earlier, we get

$$\mathsf{Pr}[A_n(C(1^n 2^n 3^n)) = 1] = q_{u_n,v_n}(G_n) \cdot \mathsf{Pr}\big[A\big(\nu_{u_n,v_n}(G_n)\big) = 1\big]$$

On the other hand, we consider the probability that circuit $A_n$ will output 1 on input a random commitment to a uniformly chosen $3n$-long sequence over $\{1,2,3\}$. The reader can easily verify that the sequence $(c_1, \ldots, c_n)$ constructed by circuit $A_n$ is distributed identically to the sequence $(d_1, \ldots, d_n)$ generated by the simulator in Step 2, conditioned on $e_{u_n} \neq e_{v_n}$. (Recall that $d_i = C(e_i)$.) Letting $T_{3n}$ denote a random variable uniformly distributed over $\{1,2,3\}^{3n}$, we get

$$\mathsf{Pr}[A_n(C(T_{3n})) = 1] = p'_{u_n,v_n}(G_n) \cdot \mathsf{Pr}\big[A\big(\mu_{u_n,v_n}(G_n)\big) = 1\big]$$

where $p'_{u_n,v_n}(G_n)$ denotes the probability that in Step 3 of the simulation the verifier will answer with $(u_n, v_n)$, conditioned on $e_{u_n} \neq e_{v_n}$. Using the fact that the proof of Claim 4.4.8.1 actually establishes that $|\mathsf{Pr}[M^*(G_n) \neq \perp] - \frac{2}{3}|$ is negligible in $n$, it follows that $|p'_{u_n,v_n}(G_n) - p_{u_n,v_n}(G_n)| < \frac{1}{3 \cdot p(n)^2}$ for all but at most finitely many $G_n$'s.

**Justification for the last assertion:** Note that $p'_{u_n,v_n}(G_n)$ and $p_{u_n,v_n}(G_n)$ refer to the same event (i.e., $V^*$'s request equals $(u_n, v_n)$), but under a different conditional space (i.e., either $e_{u_n} \neq e_{v_n}$ or $M^*(G_n) \neq \perp$). In fact, it is instructive to consider in both cases the event that *$V^*$'s request equals $(u_n, v_n)$ and $e_{u_n} \neq e_{v_n}$*. Denoting the latter event by $X$, we have[14]

[14] For further justification of the following equations, let $X'$ denote the event that $V^*$'s *request equals* $(u_n, v_n)$, and observe that $X$ is the conjunction of $X'$ and $e_{u_n} \neq e_{v_n}$. Then:

- By definition, $p'_{u_n,v_n}(G_n) = \mathsf{Pr}[X' | e_{u_n} \neq e_{v_n}]$, which equals $\mathsf{Pr}[X' \,\&\, e_{u_n} \neq e_{v_n}]/\mathsf{Pr}[e_{u_n} \neq e_{v_n}] = \mathsf{Pr}[X]/\mathsf{Pr}[e_{u_n} \neq e_{v_n}]$.
- By definition, $p_{u_n,v_n}(G_n) = \mathsf{Pr}[X' | M^*(G_n) \neq \perp]$. Note that the conjunction of $M^*(G_n) \neq \perp$ and $X'$ implies $e_{u_n} \neq e_{v_n}$, and so the former conjunction implies $X$. On the other hand, $X$ implies $M^*(G_n) \neq \perp$. It follows that $\mathsf{Pr}[M^*(G_n) \neq \perp] \cdot \mathsf{Pr}[X' | M^*(G_n) \neq \perp] = \mathsf{Pr}[X' \,\&\, M^*(G_n) \neq \perp] = \mathsf{Pr}[X \,\&\, M^*(G_n) \neq \perp] = \mathsf{Pr}[X]$. We conclude that $p_{u_n,v_n}(G_n) = \mathsf{Pr}[X]/\mathsf{Pr}[M^*(G_n) \neq \perp]$.

- $p'_{u_n,v_n}(G_n) = \mathsf{Pr}[X|e_{u_n} \neq e_{v_n}] = \mathsf{Pr}[X]/\mathsf{Pr}[e_{u_n} \neq e_{v_n}]$
- $p_{u_n,v_n}(G_n) = \mathsf{Pr}[X|M^*(G_n) \neq \perp] = \mathsf{Pr}[X]/\mathsf{Pr}[M^*(G_n) \neq \perp]$

Using $\mathsf{Pr}[e_{u_n} \neq e_{v_n}] = \frac{2}{3} \approx \mathsf{Pr}[M^*(G_n) \neq \perp]$, where $\approx$ denotes equality up to a negligible (in $n$) quantity, it follows that $|p'_{u_n,v_n}(G_n) - p_{u_n,v_n}(G_n)|$ is negligible (in $n$).

Using the conditions of case 2, and omitting $G_n$ from the notation, it follows that

$$\left|p'_{u_n,v_n} - q_{u_n,v_n}\right| \leq \left|p'_{u_n,v_n} - p_{u_n,v_n}\right| + \left|p_{u_n,v_n} - q_{u_n,v_n}\right| < \frac{2}{3 \cdot p(n)^2}$$

Combining the foregoing, we get

$$\begin{aligned}
&|\mathsf{Pr}[A_n(C(1^n2^n3^n)) = 1] - \mathsf{Pr}[A_n(C(T_{3n})) = 1]| \\
&= \left|q_{u_n,v_n} \cdot \mathsf{Pr}\left[A\left(v_{u_n,v_n}\right) = 1\right] - p'_{u_n,v_n} \cdot \mathsf{Pr}\left[A\left(\mu_{u_n,v_n}\right) = 1\right]\right| \\
&\geq q_{u_n,v_n} \cdot \left|\mathsf{Pr}\left[A\left(v_{u_n,v_n}\right) = 1\right] - \mathsf{Pr}\left[A\left(\mu_{u_n,v_n}\right) = 1\right]\right| - \left|p'_{u_n,v_n} - q_{u_n,v_n}\right| \\
&> \frac{1}{p(n)} \cdot \frac{1}{p(n)} - \frac{2}{3 \cdot p(n)^2} = \frac{1}{3 \cdot p(n)^2}
\end{aligned}$$

Hence, the circuit family $\{A_n\}$ distinguishes commitments to $\{1^n2^n3^n\}$ from commitments to $\{T_{3n}\}$. Combining an averaging argument with a hybrid argument, we conclude that there exists a polynomial-size circuit family that distinguishes commitments. This contradicts the non-uniform secrecy of the commitment scheme.

Having reached contradiction in both cases, Claim 4.4.8.2 follows. □

Combining Claims 4.4.8.1 and 4.4.8.2 (and using Exercise 9), the zero-knowledge property of $P_{G3C}$ follows. This completes the proof of the proposition. ■

### 4.4.2.4. Concluding Remarks

Construction 4.4.7 has been presented using a unidirectional commitment scheme. A fundamental property of such schemes is that their secrecy is preserved in case (polynomially) many instances are invoked simultaneously. The proof of Proposition 4.4.8 indeed took advantage on this property. We remark that Construction 4.4.4 also possesses this simultaneous secrecy property (although it is not unidirectional), and hence the proof of Proposition 4.4.8 can be carried out if the commitment scheme used is the one of Construction 4.4.4 (see Exercise 15). We recall that this latter construction constitutes a commitment scheme if and only if such schemes exist at all (since Construction 4.4.4 is based on any one-way function, and the existence of one-way functions is implied by the existence of commitment schemes).

Proposition 4.4.8 assumes the existence of a *non-uniformly* secure commitment scheme. The proof of the proposition makes essential use of the non-uniform security by incorporating instances in which the zero-knowledge property fails into circuits that contradict the security hypothesis. We stress that the sequence of "bad" instances is not necessarily constructible by efficient (uniform) machines. In other words, the zero-knowledge requirement has some non-uniform flavor. A *uniform analogue of zero-knowledge* would require only that it be infeasible to find instances in which a verifier gains knowledge (and not that such instances do not exist at all). Using a *uniformly*

secure commitment scheme, Construction 4.4.7 can be shown to be *uniformly* zero-knowledge.

By itself, Construction 4.4.7 has little practical value, since it offers a very moderate acceptance gap (between inputs from inside and outside of the language). Yet, repeating the protocol, on common input $G = (V, E)$, for $k \cdot |E|$ times (and letting the verifier accept only if all iterations are acceptance) will yield an interactive proof for $G3C$ with error probability bounded by $e^{-k}$, where $e \approx 2.718$ is the natural-logarithm base. Namely, on common input $G \in G3C$, the verifier always accepts, whereas on common input $G \notin G3C$, the verifier accepts with probability bounded above by $e^{-k}$ (no matter what the prover does). We stress that by virtue of the sequential-composition lemma (Lemma 4.3.11), if these iterations are performed sequentially, then the resulting (strong) interactive proof is zero-knowledge as well. Setting $k$ to be any super-logarithmic function of $|G|$ (e.g., $k = |G|$), the error probability of the resulting interactive proof is negligible. We remark that it is unlikely that the interactive proof that results by performing these $k \cdot |E|$ iterations in parallel is zero-knowledge; see Section 4.5.4.

An important property of Construction 4.4.7 is that the prescribed prover (i.e., $P_{G3C}$) can be implemented in probabilistic polynomial time, provided that it is given as auxiliary input a 3-coloring of the common-input graph. As we shall see, this property is essential for application of Construction 4.4.7 to the design of cryptographic protocols.

As mentioned earlier, the choice of $G3C$ as a "bootstrapping" $\mathcal{NP}$-complete language is totally arbitrary. It is quite easy to design analogous zero-knowledge proofs for other popular $\mathcal{NP}$-complete languages using the underlying ideas presented in Section 4.4.2.1 (i.e., the *motivating discussion*).

### 4.4.3. The General Result and Some Applications

The theoretical and practical importance of a zero-knowledge proof for Graph 3-Coloring (e.g., Construction 4.4.7) follows from the fact that it can be applied to prove, in zero-knowledge, any statement having a short proof that can be efficiently verified. More precisely, a zero-knowledge proof system for a specific $\mathcal{NP}$-complete language (e.g., Construction 4.4.7) can be used to present zero-knowledge proof systems for every language in $\mathcal{NP}$.

Before presenting zero-knowledge proof systems for every language in $\mathcal{NP}$, let us recall some conventions and facts concerning $\mathcal{NP}$. We first recall that every language $L \in \mathcal{NP}$ is *characterized* by a binary relation $R$ satisfying the following properties:

- There exists a polynomial $p(\cdot)$ such that for every $(x, y) \in R$, it holds that $|y| \leq p(|x|)$.
- There exists a polynomial-time algorithm for deciding membership in $R$.
- $L = \{x : \exists w \text{ s.t. } (x, w) \in R\}$.

  (Such a $w$ is called a witness for the membership of $x \in L$.)

Actually, each language in $\mathcal{NP}$ can be characterized by infinitely many such relations. Yet for each $L \in \mathcal{NP}$, we fix and consider one characterizing relation, denoted $R_L$. Because $G3C$ is $\mathcal{NP}$-complete, we know that $L$ is polynomial-time-reducible (i.e.,

Karp-reducible) to $G3C$. Namely, there exists a polynomial-time-computable function $f$ such that $x \in L$ if and only if $f(x) \in G3C$. Last, we observe that the standard reduction of $L$ to $G3C$, denoted $f_L$, has the following additional property:

- *There exists a polynomial-time-computable function, denoted $g_L$, such that for every $(x, w) \in R_L$, it holds that $g_L(x, w)$ is a 3-coloring of $f_L(x)$.*

We stress that this additional property is not required by the standard definition of a Karp reduction. Yet it can be easily verified (see Exercise 16) that the standard reduction $f_L$ (i.e., the composition of the generic reduction of $L$ to $SAT$, the standard reductions of $SAT$ to $3SAT$, and the standard reduction of $3SAT$ to $G3C$) does have such a corresponding $g_L$. Using these conventions, we are ready to "reduce" the construction of zero-knowledge proofs for $\mathcal{NP}$ to a zero-knowledge proof system for $G3C$.

**Construction 4.4.9 (A Zero-Knowledge Proof for a Language $L \in \mathcal{NP}$):**

- Common input: *A string $x$ (supposedly in $L$).*
- Auxiliary input to the prover: *A* witness*, $w$, for the membership of $x \in L$ (i.e., a string $w$ such that $(x, w) \in R_L$).*
- Local pre-computation: *Each party computes $G \stackrel{\text{def}}{=} f_L(x)$. The prover computes $\psi \stackrel{\text{def}}{=} g_L(x, w)$.*
- Invoking a zero-knowledge proof for $G3C$: *The parties invoke a zero-knowledge proof on common input $G$. The prover enters this proof with auxiliary input $\psi$.*

Clearly, if the prescribed prover in the $G3C$ proof system can be implemented in probabilistic polynomial time when given an $\mathcal{NP}$-witness (i.e., a 3-coloring) as auxiliary input, then the same holds for the prover in Construction 4.4.9.

**Proposition 4.4.10:** *Suppose that the sub-protocol used in the last step of Construction 4.4.9 is indeed an auxiliary-input zero-knowledge proof for $G3C$. Then Construction 4.4.9 constitutes an auxiliary-input zero-knowledge proof for $L$.*

***Proof:*** The fact that Construction 4.4.9 constitutes an interactive proof for $L$ is immediate from the validity of the reduction (and the fact that it uses an interactive proof for $G3C$). At first glance it seems that the zero-knowledge property of Construction 4.4.9 follows just as easily. There is, however, a minor issue that one should not ignore: The verifier in the zero-knowledge proof for $G3C$ invoked in Construction 4.4.9 possesses not only the common-input graph $G$ but also the original common input $x$ that reduces to $G$. This extra information might have helped this verifier to extract knowledge in the $G3C$ interactive proof if it were not the case that this proof system is also zero-knowledge with respect to the auxiliary input. Details follow.

Suppose we need to simulate the interaction of a machine $V^*$ with the prover of Construction 4.4.9, on common input $x$. Without loss of generality, we can assume that machine $V^*$ invokes an interactive machine $V^{**}$ that interacts with the prover

of the $G3C$ interactive proof, on common input $G = f_L(x)$, and has auxiliary input $x$. Using the hypothesis that the $G3C$ interactive proof is auxiliary-input zero-knowledge, it follows that there exists a simulator $M^{**}$ that on input $(G, x)$ simulates the interaction of $V^{**}$ with the $G3C$ prover (on common input $G$ and the verifier's auxiliary input $x$). Hence the simulator for Construction 4.4.9, denoted $M^*$, operates as follows: On input $x$, the simulator $M^*$ computes $G \stackrel{\text{def}}{=} f_L(x)$ and outputs $M^{**}(G, x)$. The proposition follows. ■

An alternative way of resolving the minor difficulty addressed earlier is to observe that the function $f_L$ (i.e., the one induced by the standard reductions) can be inverted in polynomial time (see Exercise 17). In any case, we immediately get the following:

**Theorem 4.4.11:** *Suppose that there exists a commitment scheme satisfying the (non-uniform) secrecy and unambiguity requirements. Then every language in $\mathcal{NP}$ has an auxiliary-input zero-knowledge proof system. Furthermore, the prescribed prover in this system can be implemented in probabilistic polynomial time provided it gets the corresponding $\mathcal{NP}$-witness as auxiliary input.*

We remind the reader that the condition of the theorem is satisfied if (and only if) there exist (non-uniformly) one-way functions: See Theorem 3.5.12 (asserting that one-way functions imply pseudorandom generators), Proposition 4.4.5 (asserting that pseudorandom generators imply commitment schemes), and Exercise 13 (asserting that commitment schemes imply one-way functions).

### Applications: An Example

A typical application of Theorem 4.4.11 is to enable one party to prove some property of its secret without revealing the secret. For concreteness, consider a party, denoted $S$, that makes a commitment to another party, denoted $R$. Suppose that at a later stage, party $S$ is willing to reveal partial information about the committed value but is not willing to reveal all of it. For example, party $S$ may want to reveal a single bit indicating whether or not the committed value is larger than some value specified by $R$. If party $S$ sends only this bit, party $R$ cannot know if the bit sent is indeed the correct one. Using a zero-knowledge proof allows $S$ to convince $R$ of the correctness of the revealed bit without yielding any additional knowledge. The existence of such a zero-knowledge proof follows from Theorem 4.4.11 and the fact that the statement to be proved is of $\mathcal{NP}$ type (and that $S$ knows the corresponding $\mathcal{NP}$-witness).

A reader who is not fully convinced of the validity of the foregoing claims (i.e., regarding the applicability of Theorem 4.4.11) may want to formalize the story as follows: Let $v$ denote the value to which $S$ commits, let $s$ denote the randomness it uses in the commitment phase, and let $c \stackrel{\text{def}}{=} C_s(v)$ be the resulting commitment (relative to the commitment scheme $C$). Suppose that $S$ wants to prove to $R$ that $c$ is a commitment to a value greater than $u$. So what $S$ wants to prove (in zero-knowledge) is that *there exist $v$ and $s$ such that $c = C_s(v)$ and $v > u$*, where $c$ and $u$ are known to $R$. Indeed, this is an $\mathcal{NP}$-type statement, and $S$ knows the corresponding $\mathcal{NP}$-witness (i.e., $(v, s)$), since it has picked $v$ and $s$ by itself.

Formally, we define a language

$$L \stackrel{\text{def}}{=} \{(c,u) : \exists v, s \text{ s.t. } c = C_s(v) \text{ and } v > u\}$$

Clearly, the language $L$ is in $\mathcal{NP}$, and the $\mathcal{NP}$-witness for $(c,u) \in L$ is a pair $(v,s)$, as shown. Hence, Theorem 4.4.11 can be applied.

Additional examples are presented in Exercise 18. Other applications will appear in Volume 2.

We stress that because it is a general (and in some sense generic) result, the construction underlying Theorem 4.4.11 cannot be expected to provide a practical solution (especially in simple cases). Theorem 4.4.11 should be viewed as a plausibility argument: It asserts that there is a wide class of cryptographic problems (that amount to proving the consistency of a secret-dependent action with respect to some public information) that are solvable in principle. Thus, when faced with such a problem *in practice*, one can infer that a solution does exist. This is merely a first step, to be followed by the search for a practical solution.

### Zero-Knowledge for Any Language in $\mathcal{IP}$

Interestingly, the result of Theorem 4.4.11 can be extended "to the maximum," in the sense that under the same conditions every language having an interactive proof system also has a zero-knowledge interactive proof system. Namely:

**Theorem 4.4.12:** *Suppose that there exists a commitment scheme satisfying the (non-uniform) secrecy and unambiguity requirements. Then every language in $\mathcal{IP}$ has a zero-knowledge proof system.*

We believe that this extension (of Theorem 4.4.11 to Theorem 4.4.12) does not have much practical significance. Theorem 4.4.12 is proved by first converting the interactive proof for $L$ into a public-coin interactive proof with perfect completeness (see Section 4.2.3). In the latter proof system, the verifier is supposed to send random strings (regardless of the prover's previous messages) and decide whether or not to accept by applying some polynomial-time predicate to the full transcript of the communication. Thus, we can modify this proof system by letting the new prover send commitments to the messages sent by the original (public-coin-system) prover, rather than sending these messages in the clear. Once this "encrypted" interaction is completed, the prover proves in zero-knowledge that the original verifier would have accepted the hidden transcript (this is an $\mathcal{NP}$ statement). Thus, Theorem 4.4.12 is proved by applying Theorem 4.4.11.

## 4.4.4. Second-Level Considerations

When presenting zero-knowledge proof systems for every language in $\mathcal{NP}$, we made no attempt to present the most efficient construction possible. Our main concern was to present a proof that is as simple to explain as possible. However, once we know that zero-knowledge proofs for $\mathcal{NP}$ exist, it is natural to ask how efficient they can

be. More importantly, we introduce and discuss a more refined measure of the "actual security" of a zero-knowledge proof, called knowledge tightness.

In order to establish common ground for comparing zero-knowledge proofs, we have to specify a desired measure of error probability for these proofs. An instructive choice, used in the sequel, is to consider the complexity of zero-knowledge proofs with error probability $2^{-k}$, where $k$ is a parameter that may depend on the length of the common input. Another issue to bear in mind when comparing zero-knowledge proofs concerns the assumptions under which they are valid. Throughout this entire subsection we stick to the assumption used thus far (i.e., the existence of one-way functions).

### 4.4.4.1. Standard Efficiency Measures

Natural and standard efficiency measures to be considered are as follows:

- The *communication complexity of the proof.* The most important communication measure is the *round complexity* (i.e., the number of message exchanges). The total number of bits exchanged in the interaction is also an important consideration.
- The *computational complexity of the proof* (specifically, the number of elementary steps taken by each of the parties).

Communication complexity seems more important than computational complexity as long as the trade-off between them is "reasonable."

To demonstrate these measures, we consider the zero-knowledge proof for $G3C$ presented in Construction 4.4.7. Recall that this proof system has a very moderate acceptance gap, specifically $1/|E|$, on common input graph $G = (V, E)$. Thus, Construction 4.4.7 has to be applied sequentially $k \cdot |E|$ times in order to result in a zero-knowledge proof with error probability $e^{-k}$, where $e \approx 2.718$ is the natural-logarithm base. Hence, the round complexity of the resulting zero-knowledge proof is $O(k \cdot |E|)$, the bit complexity is $O(k \cdot |E| \cdot |V|^2)$, and the computational complexity is $O(k \cdot |E| \cdot \text{poly}(|V|))$, where the polynomial $\text{poly}(\cdot)$ depends on the commitment scheme in use.

Much more efficient zero-knowledge proof systems can be custom-made for specific languages in $\mathcal{NP}$. Furthermore, even if one adopts the approach of reducing the construction of zero-knowledge proof systems for $\mathcal{NP}$ languages to the construction of a zero-knowledge proof system for a single $\mathcal{NP}$-complete language, efficiency improvements can be achieved. For example, using Exercise 20, one can present zero-knowledge proofs for the Hamiltonian-cycle problem (again with error $2^{-k}$) having round complexity $O(k)$, bit complexity $O(k \cdot |V|^{2+\varepsilon})$, and computational complexity $O(k \cdot |V|^{2+O(\varepsilon)})$, where $\varepsilon > 0$ is a constant depending on the desired security of the commitment scheme (in Construction 4.4.7 and in Exercise 20 we chose $\varepsilon = 1$). Note that complexities depending on the instance size are affected by reductions among problems, and hence a fair comparison is obtained by considering the complexities for the generic problem (i.e., Bounded Halting).

The round complexity of a protocol is a very important efficiency consideration, and it is desirable to reduce it as much as possible. In particular, it is desirable to have zero-knowledge proofs with constant numbers of rounds and negligible error probability. This goal is pursued in Section 4.9.

### 4.4.4.2. Knowledge Tightness

The foregoing efficiency measures are generic in the sense that they are applicable to any protocol (independent of whether or not it is zero-knowledge). Because security and efficiency often are convertible from one to the other (especially in this context), one should consider refined measures of efficiency only in conjunction with a refined measure of security.

In contrast to the generic (efficiency) measures, we consider a (security) measure specific to zero-knowledge, called *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 the verifier needs 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 (expected) running time of the simulator and the running time of the verifier in the real interaction simulated by the simulator. Note that the simulators presented thus far, as well as all known simulators, operate by repeated random trials, and hence an instructive measure of tightness should consider their expected running times (assuming they never err, i.e., never output the special $\perp$ symbol), rather than the worst case. (Alternatively, one can consider the running time of a simulator that outputs $\perp$ with probability at most $\frac{1}{2}$.)

**Definition 4.4.13 (Knowledge Tightness):** *Let $t : \mathbb{N} \to \mathbb{N}$ be a function. We say that a zero-knowledge proof for language $L$ has* **knowledge tightness** *$t(\cdot)$ if there exists a polynomial $p(\cdot)$ such that for every probabilistic polynomial-time verifier $V^*$ there exists a simulator $M^*$ (as in Definition 4.3.2) such that for all sufficiently long $x \in L$ we have*

$$\frac{\text{Time}_{M^*}(x) - p(|x|)}{\text{Time}_{V^*}(x)} \leq t(|x|)$$

*where $\text{Time}_{M^*}(x)$ denotes the expected running time of $M^*$ on input $x$, and $\text{Time}_{V^*}(x)$ denotes the running time of $V^*$ on common input $x$.*

We assume a model of computation that allows one machine to emulate another machine at the cost of only the running time of the latter machine. The purpose of polynomial $p(\cdot)$ in the foregoing definition is to take care of generic overhead created by the simulation (this is important only in case the verifier $V^*$ is extremely fast). We remark that the definition of zero-knowledge does not guarantee that the knowledge tightness is polynomial. Yet all known zero-knowledge proofs, and, more generally, all zero-knowledge properties demonstrated using a single simulator with black-box access to $V^*$, have polynomial knowledge tightness. In particular, Construction 4.3.8 (like the construction in Exercise 20) has knowledge tightness 2, whereas Construction 4.4.7 has knowledge tightness approximately $\frac{3}{2}$. We believe that knowledge tightness is a very important efficiency consideration and that it is desirable to have it be a constant.

We comment that the notion of knowledge tightness is also instructive in reconciling statements like the following:

1. Executing Construction 4.4.7 $O(\log n)$ times in parallel, where $n$ is the number of vertices in the graph, results in a zero-knowledge proof system.
2. Executing Construction 4.4.7 more than $O(\log n)$ times (say $O((\log n) \cdot (\log \log n))$ times) in parallel is not known to result in a zero-knowledge proof system. (Furthermore, it is unlikely that the resulting proof system can be shown to be zero-knowledge; see Section 4.5.4.2.)

The gap between these conflicting statements seems less dramatic once one realizes that executing Construction 4.4.7 $k(n) = O(\log n)$ times in parallel results in a zero-knowledge proof system of knowledge tightness approximately $(3/2)^{k(n)}$. (See Exercise 19.)

## 4.5.* Negative Results

In this section we review some negative results concerning zero-knowledge. These results indicate that some of the shortcomings of the results and constructions presented in previous sections are unavoidable. Most importantly, Theorem 4.4.11 asserts the existence of (computational) zero-knowledge interactive proof systems for $\mathcal{NP}$, assuming that one-way functions exist. Three questions arise naturally:

1. *Unconditional results*: Can one prove the existence of (computational) zero-knowledge proof systems for $\mathcal{NP}$ without making any assumptions?
2. *Perfect zero-knowledge*: Can one present perfect zero-knowledge proof systems for $\mathcal{NP}$ even under some reasonable assumptions?
3. *The role of randomness and interaction*: For example, can one present error-free zero-knowledge proof systems for $\mathcal{NP}$?

The answers to all these questions seem to be negative.

Another important question concerning zero-knowledge proofs is their preservation under parallel composition. We shall show that, *in general*, zero-knowledge is not preserved under parallel composition (i.e., there exists a pair of zero-knowledge protocols that when executed in parallel will leak knowledge, in a strong sense). Furthermore, we shall consider some natural proof systems, obtained via parallel composition of zero-knowledge proofs (e.g., the one of Construction 4.4.7), and indicate that it is unlikely that the resulting composed proofs can be proved to be zero-knowledge.

**Organization.** We start by reviewing some results regarding the essential roles of both randomness and interaction in Theorem 4.4.11 (i.e., the existence of zero-knowledge proofs for $\mathcal{NP}$). For these results we also present the relatively simple proof ideas (see Section 4.5.1). Next, in Section 4.5.2, we claim that the existence of zero-knowledge proofs for $\mathcal{NP}$ implies some form of average-case one-way hardness, and so the assumption in Theorem 4.4.11 cannot be totally eliminated. In Section 4.5.3 we consider perfect zero-knowledge proof systems, and in Section 4.5.4, the composition of zero-knowledge protocols.

Jumping ahead, we mention that all the results presented in this section, except Theorem 4.5.8 (i.e., the limitation of perfect zero-knowledge proofs), apply also to zero-knowledge arguments as defined and discussed in Section 4.8.

### 4.5.1. On the Importance of Interaction and Randomness

We call a proof system *trivial* if it is a proof system for a language in $\mathcal{BPP}$. Because languages in $\mathcal{BPP}$ can be decided by the verifier without any interaction with the prover, such proof systems are of no use (at least as far as cryptography is concerned).

**On the Triviality of Unidirectional Zero-Knowledge Proofs.** A *unidirectional* proof system is one in which a single message is sent (i.e., from the prover to the verifier). We show that such proof systems, which constitute a special class of interactive proofs that includes $\mathcal{NP}$-type proofs as special cases, are too restricted to allow non-trivial zero-knowledge proofs.

**Theorem 4.5.1:** *Suppose that $L$ has a unidirectional zero-knowledge proof system. Then $L \in \mathcal{BPP}$.*

***Proof Idea:*** Given a simulator $M$ for the view of the honest verifier in this system (as guaranteed by Definition 4.3.3), we construct a decision procedure for $L$. On input $x$, we invoke $M(x)$ and obtain $(w, r)$, where $w$ supposedly is a message sent by the prover and $r \in \{0,1\}^\ell$ supposedly is the random tape of the verifier. We uniformly select $r' \in \{0,1\}^\ell$ and decide as the true verifier would have decided upon receiving the message $w$ and using $r'$ as the content of its random tape. The hypothesis that $M$ is a good simulator is used in the analysis of the case $x \in L$, whereas the soundness of the proof system (and the fact that $r'$ is selected independently of $w$) is used for the case $x \notin L$. ■

**On the Essential Role of the Verifier's Randomness.** We next show that randomization on the verifier's part is necessary for the non-triviality of zero-knowledge proof systems. It follows that a non-zero error probability is essential to the non-triviality of zero-knowledge proof systems, because otherwise the verifier could always set its random tape to be all zeros. (In fact, we can directly prove that a non-zero soundness error is essential to the non-triviality of zero-knowledge proof systems and derive Theorem 4.5.2 as a special case.[15])

**Theorem 4.5.2:** *Suppose that $L$ has a zero-knowledge proof system in which the verifier program is deterministic. Then $L \in \mathcal{BPP}$.*

[15] Again, given a simulator $M$ for the view of the honest verifier in this system, we construct a decision procedure for $L$. On input $x$, we invoke $M(x)$ and accept if and only if the output corresponds to a transcript that the honest verifier would have accepted. The hypothesis that $M$ is a good simulator is used in the analysis of the case $x \in L$, whereas the *perfect* soundness of the proof system is used for the case $x \notin L$. Theorem 4.5.2 follows because deterministic verifiers necessarily have zero soundness error.

***Proof Idea:*** Because the verifier is deterministic, the prover can fully determine each of its future messages. Thus the proof system can be converted into an equivalent one in which the prover simply sends to the verifier the full transcript of an execution in the original proof system. Observe that the completeness, soundness, and zero-knowledge properties of the original proof system are preserved and that the resulting proof system is unidirectional. We conclude by applying Theorem 4.5.1. ■

**On the Essential Role of the Prover's Randomness.** Finally, we show that randomization on the prover's part is also necessary for the non-triviality of zero-knowledge proof systems.

**Theorem 4.5.3:** *Suppose that $L$ has an* auxiliary-input *zero-knowledge proof system in which the prover program is deterministic. Then $L \in \mathcal{BPP}$.*

Note that the hypothesis (i.e., the type of zero-knowledge requirement) is stronger here. (Computationally unbounded deterministic provers may suffice for the non-triviality of the bare definition of zero-knowledge (i.e., Definition 4.3.2).)

***Proof Idea:*** Suppose, without loss of generality, that the verifier is the party sending the first message in this proof system. We consider a cheating verifier that given an auxiliary input $z_1, \ldots, z_t$ sends $z_i$ as its $i$th message. The remaining messages of this verifier are determined arbitrarily. We first observe that because the prover is deterministic, in a real interaction the first $i \leq t$ responses of the prover are determined by $z_1, \ldots, z_i$. Thus, that must be essentially the case in the simulation. We construct a decision procedure for $L$ by emulating the interaction of the prescribed prover with the prescribed verifier on common input equal to the input to the procedure, denoted $x$. Toward this end, we uniformly select and fix a random tape, denoted $r$, for the verifier. The emulation proceeds in iterations corresponding to the prover's messages. To obtain the prover's next message, we first determine the next verifier message (by running the program of the prescribed verifier on input $x$, coins $r$, and incoming messages as recorded thus far). Next, we invoke the simulator on input $(x, (z_1, \ldots, z_i))$, where $z_1, \ldots, z_i$ are the verifier's messages determined thus far, and so we obtain and record the prover's $i$th message. Our final decision is determined by the verifier's decision. ■

### 4.5.2. Limitations of Unconditional Results

Recall that Theorem 4.4.12 asserts the existence of zero-knowledge proofs for all languages in $\mathcal{IP}$, *provided that non-uniformly one-way functions exist*. In this subsection we consider the question of whether or not this sufficient condition is also necessary. The following results seem to provide some (yet, weak) indication in that direction. Specifically, the existence of zero-knowledge proof systems for languages outside of $\mathcal{BPP}$ implies (very weak) forms of one-wayness. In a dual way, the

existence of zero-knowledge proof systems for languages that are hard to approximate (in some average-case sense) implies the existence of one-way functions (but not of non-uniformly one-way functions). In the rest of this subsection we merely provide precise statements of these results.

**Non-Triviality of ZK Implies Weak Forms of One-Wayness.** By the non-triviality of zero-knowledge we mean the existence of zero-knowledge proof systems for languages outside of $\mathcal{BPP}$ (as the latter have trivial zero-knowledge systems in which the prover does nothing). Let us clarify what we mean by "weak forms of one-wayness." Our starting point is the definition of a collection of one-way functions (i.e., Definition 2.4.3). Recall that these are collections of functions, indexed by some $\overline{I} \subseteq \{0,1\}^*$, that are easy to sample and evaluate but *typically* hard to invert. That is, a typical function $f_i$ (for $i \in \overline{I}$) is hard to invert on a typical image. Here we require only that there exist functions in the collection that are hard to invert on a typical image.

**Definition 4.5.4 (Collection of Functions with One-Way Instances):** *A collection of functions $\{f_i : D_i \to \{0,1\}^*\}_{i\in\overline{I}}$ is said to* **have one-way instances** *if there exist three probabilistic polynomial-time algorithms $I$, $D$, and $F$ such that the following two conditions hold:*

1. Easy to sample and compute: *As in Definition 2.4.3.*
2. Some functions are hard to invert: *For every probabilistic polynomial-time algorithm $A'$, every polynomial $p(\cdot)$, and infinitely many $i \in \overline{I}$,*

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

*where $X_i = D(i)$.*

Actually, because the hardness condition does not refer to the distribution induced by $I$, we can omit $I$ from the definition and refer only to the index set $\overline{I}$. Such a collection contains infinitely many functions that are hard to invert, but there may be no efficient way of selecting such a function (and thus the collection is of no real value). Still, we stress that the hardness condition has an average-case flavor; each of these infinitely many functions is hard to invert in a strong probabilistic sense, not merely in the worst case.

**Theorem 4.5.5:** *If there exist zero-knowledge proofs for languages outside of $\mathcal{BPP}$, then there exist collections of functions with one-way instances.*

We remark that the mere assumption that $\mathcal{BPP} \subset \mathcal{IP}$ is not known to imply any form of (average) one-wayness. Even the existence of a language in $\mathcal{NP}$ that is not in $\mathcal{BPP}$ does not imply any form of average-case hardness; it merely implies the existence of a function that is easy to compute but hard to invert *in the worst case* (see Section 2.1).

**ZK for "Hard" Languages Yields One-Way Functions.** Our notion of hard languages is the following:

**Definition 4.5.6:** *We say that a language $L$ is* **hard to approximate** *if there exists a probabilistic polynomial-time algorithm $S$ such that for every probabilistic polynomial-time algorithm $A$, every polynomial $p(\cdot)$, and all sufficiently large $n$'s,*

$$\mathsf{Pr}[A(X_n) = \chi_L(X_n)] < \frac{1}{2} + \frac{1}{p(n)}$$

*where $X_n \stackrel{\text{def}}{=} S(1^n)$, and $\chi_L$ is the characteristic function of the language $L$ (i.e., $\chi_L(x) = 1$ if $x \in L$, and $\chi_L(x) = 0$ otherwise).*

For example, if $f$ is a one-way permutation and $b$ is a hard-core predicate for $f$, then the language $L_f \stackrel{\text{def}}{=} \{x \in \{0,1\}^* : b(f^{-1}(x)) = 1\} \in \mathcal{NP}$ is hard to approximate (under the uniform distribution).

**Theorem 4.5.7:** *If there exist zero-knowledge proofs for languages that are hard to approximate, then there exist one-way functions.*

We stress that the mere existence of languages that are hard to approximate is not known to imply the existence of one-way functions (see Section 2.1).

### 4.5.3. Limitations of Statistical ZK Proofs

A theorem bounding the class of languages possessing *perfect* zero-knowledge proof systems follows. In fact, the bound refers even to *statistical* (i.e., *almost-perfect*) zero-knowledge proof systems (see Section 4.3.1.4). We start with some background. By $\mathcal{AM}$ we denote the class of languages having interactive proofs that proceed as follows. First the verifier sends a random string to the prover, next the prover answers with some string, and finally the verifier decides whether to accept or reject based on a deterministic computation (depending on the common input and the two strings). It is believed that co$\mathcal{NP}$ is not contained in $\mathcal{AM}$ (or, equivalently, $\mathcal{NP}$ is not contained in co$\mathcal{AM}$). Additional support for this belief is provided by the fact that co$\mathcal{NP} \subseteq \mathcal{AM}$ implies the collapse of the Polynomial-Time Hierarchy. In any case, the result we wish to mention is the following:

**Theorem 4.5.8:** *If there exists a statistical (almost-perfect) zero-knowledge proof system for a language $L$, then $L \in \text{co}\mathcal{AM}$. (In fact, $L \in \text{co}\mathcal{AM} \cap \mathcal{AM}$.)*

The theorem remains valid under several relaxations of statistical zero-knowledge (e.g., allowing the simulator to run in expected polynomial-time). Hence, if some $\mathcal{NP}$-complete language has a statistical zero-knowledge proof system, then co$\mathcal{NP} \subseteq \mathcal{AM}$, which is unlikely.

We stress that Theorem 4.5.8 *does not* apply to perfect (or statistical) zero-knowledge *arguments*, defined and discussed in Section 4.8. Hence, there is no conflict between Theorem 4.5.8 and the fact that under some reasonable complexity assumptions, perfect zero-knowledge arguments do exist for every language in $\mathcal{NP}$.

### 4.5.4. Zero-Knowledge and Parallel Composition

We present two negative results regarding parallel composition of zero-knowledge protocols. These results are very different in terms of their conceptual standing: The first result asserts the failure (in general) of the *parallel-composition conjecture* (i.e., the conjecture that running *any* two zero-knowledge protocols in parallel will result in a zero-knowledge protocol), but says nothing about specific natural candidates. The second result refers to a class of interactive proofs that contains several interesting and natural examples, and it asserts that the members of this class cannot be proved zero-knowledge using a general paradigm (known by the name "black-box simulation"). The relation of the second result to this subsection follows from the fact that some of the members in this class are obtained by parallel composition of natural zero-knowledge proofs. We mention that it is hard to conceive an alternative way of demonstrating the zero-knowledge property of protocols (other than by providing a black-box simulator).

We stress that by "parallel composition" we mean playing several copies of the protocol in parallel, where the prescribed (honest) parties execute each copy independently of the other copies. Specifically, if a party is required to toss coins in a certain round, then it will toss independent coins for each of the copies.

#### 4.5.4.1. Failure of the Parallel-Composition Conjecture

As a warning about trusting unsound intuitions, we mention that for several years (following the introduction of zero-knowledge proofs) some researchers insisted that the following must be true:

> **Parallel-Composition Conjecture:** *Let $P_1$ and $P_2$ be two zero-knowledge provers. Then the prover that results from running both of them in parallel is also zero-knowledge.*

However, the parallel-composition conjecture is simply wrong.

> **Proposition 4.5.9:** *There exist two provers, $P_1$ and $P_2$, such that each is zero-knowledge, and yet the prover that results from running both of them in parallel yields knowledge (e.g., a cheating verifier can extract from this prover a solution to a problem that is not solvable in polynomial time). Furthermore, the foregoing holds even if the zero-knowledge property of each of the $P_i$'s can be demonstrated with a simulator that uses the verifier as a black box (as in Definition 4.5.10).*

> ***Proof Idea:*** Consider a prover, denoted $P_1$, that sends "knowledge" to the verifier if and only if the verifier can answer some randomly chosen *hard question* (i.e., we stress that the question is chosen by $P_1$). Answers to such hard questions

look pseudorandom, yet $P_1$ (which is not computationally bounded) can verify their correctness. Now consider a second (computationally unbounded) prover, denoted $P_2$, that answers these hard questions. Each of these provers (by itself) is zero-knowledge: $P_1$ is zero-knowledge because it is unlikely that any probabilistic polynomial-time verifier can answer its questions, whereas $P_2$ is zero-knowledge because its answers can be simulated by random strings. Yet, once they are played in parallel, a cheating verifier can answer the question of $P_1$ by sending it to $P_2$ and using the answer obtained from $P_2$ to gain knowledge from $P_1$. To turn this idea into a proof we need to construct a hard problem with the previously postulated properties. ■

The foregoing proposition refutes the parallel-composition conjecture by means of exponential-time provers. Assuming the existence of one-way functions, the parallel-composition conjecture can also be refuted for probabilistic polynomial-time provers (with auxiliary inputs). For example, consider the following two provers $P_1$ and $P_2$, which make use of proofs of knowledge (see Section 4.7). Let $C$ be a bit-commitment scheme (which we know to exist provided that one-way functions exist). On common input $C(1^n, \sigma)$, where $\sigma \in \{0, 1\}$, prover $P_1$ proves to the verifier, in zero-knowledge, that it knows $\sigma$. (To this end the prover is given as auxiliary input the coins used in the commitment.) In contrast, on common input $C(1^n, \sigma)$, prover $P_2$ asks the verifier to prove that it knows $\sigma$, and if $P_2$ is convinced, then it sends $\sigma$ to the verifier. This verifier employs the same proof-of-knowledge system used by the prover $P_1$. Clearly, each prover is zero-knowledge, and yet their parallel composition is not.

Similarly, using stronger intractability assumptions, one can also refute the parallel-composition conjecture with respect to almost-perfect zero-knowledge (rather than with respect to computational zero-knowledge). (Here we let the provers use a perfect zero-knowledge, computationally sound proof of knowledge; see Section 4.8.)

### 4.5.4.2. Problems Occurring with "Natural" Candidates

By definition, to show that a prover is zero-knowledge, one has to present, for each prospective verifier $V^*$, a corresponding simulator $M^*$ (which simulates the interaction of $V^*$ with the prover). However, all known demonstrations of zero-knowledge proceed by presenting one "universal" simulator that uses any prospective verifier $V^*$ as a black box. In fact, these demonstrations use as a black box (or oracle) the next-message function determined by the verifier program (i.e., $V^*$), its auxiliary input, and its random input. (This property of the simulators is implicit in our constructions of the simulators in previous sections.) We remark that it is hard to conceive an alternative way of demonstrating the zero-knowledge property (because a non-black-box usage of a verifier seems to require some "reverse engineering" of its code). This difficulty is greatly amplified in the context of auxiliary-input zero-knowledge.

**Definition 4.5.10 (Black-Box Zero-Knowledge):**

- Next-message function: *Let $B$ be an interactive Turing machine, and let $x$, $z$, and $r$ be strings representing a common input, an auxiliary input, and a random*

*input, respectively. Consider the function $B_{x,z,r}(\cdot)$ describing the messages sent by machine $B$ such that $B_{x,z,r}(\overline{m})$ denotes the message sent by $B$ on common input $x$, auxiliary input $z$, random input $r$, and sequence of incoming messages $\overline{m}$. For simplicity, we assume that the output of $B$ appears as its last message.*

- Black-box simulator: *We say that a probabilistic polynomial-time oracle machine $M$ is a* **black-box simulator** for the prover $P$ and the language $L$ *if for every polynomial-time interactive machine $B$, every probabilistic polynomial-time oracle machine $D$, every polynomial $p(\cdot)$, all sufficiently large $x \in L$, and every $z, r \in \{0,1\}^*$,*

$$\left|\mathsf{Pr}\left[D^{B_{x,z,r}}(\langle P, B_r(z)\rangle(x)) = 1\right] - \mathsf{Pr}\left[D^{B_{x,z,r}}\left(M^{B_{x,z,r}}(x)\right) = 1\right]\right| < \frac{1}{p(|x|)}$$

*where $B_r(z)$ denotes the interaction of machine $B$ with auxiliary input $z$ and random input $r$.*
- *We say that $P$ is* **black-box zero-knowledge** *if it has a black-box simulator.*

Essentially, the definition says that a black-box simulator mimics the interaction of prover $P$ with any polynomial-time verifier $B$ relative to any auxiliary input (i.e., $z$) that $B$ may get and any random input (i.e., $r$) that $B$ may choose. The simulator does so (efficiently) merely by using oracle calls to $B_{x,z,r}$ (which specifies the next message that $B$ sends on input $x$, auxiliary input $z$, and random input $r$). The simulation is indistinguishable from the true interaction even if the distinguishing algorithm (i.e., $D$) is given access to the oracle $B_{x,z,r}$. An equivalent formulation is presented in Exercise 21. Clearly, if $P$ is black-box zero-knowledge, then it is zero-knowledge with respect to auxiliary input (and has polynomially bounded knowledge tightness, see Definition 4.4.13).

**Theorem 4.5.11:** *Suppose that $(P, V)$ is an interactive proof system* with negligible error probability *for the language $L$. Further suppose that $(P, V)$ has the following properties:*

- Constant round*: There exists an integer $k$ such that for every $x \in L$, on input $x$ the prover $P$ sends at most $k$ messages.*
- Public coins*: The messages sent by the verifier $V$ are predetermined consecutive segments of its random tape.*
- Black-box zero-knowledge*: The prover $P$ has a black-box simulator (over the language $L$).*

*Then $L \in \mathcal{BPP}$.*

The theorem also holds for computationally sound zero-knowledge proof systems defined and discussed in Section 4.8.

We remark that both Construction 4.3.8 (zero-knowledge proof for Graph Isomorphism) and Construction 4.4.7 (zero-knowledge proof for Graph Colorability) are constant-round, use public coins, and are black-box zero-knowledge (for the corresponding language). However, they *do not* have negligible error probability. Yet, repeating each of these constructions polynomially many times *in parallel* yields an

interactive proof, with negligible error probability, for the corresponding language.[16] Clearly the resulting proof systems are constant-round and use public coins. Hence, unless the corresponding languages are in $\mathcal{BPP}$, these resulting proof systems *are not* black-box zero-knowledge.

Theorem 4.5.11 is sometimes interpreted as pointing to an inherent limitation of interactive proofs with public coins (also known as *Arthur-Merlin* games). Such proofs cannot be both *round-efficient* (i.e., have constant number of rounds and negligible error) and black-box zero-knowledge (unless they are trivially so, i.e., the language is in $\mathcal{BPP}$). In other words, *when constructing round-efficient zero-knowledge proof systems* (for languages not in $\mathcal{BPP}$), *one should use "private coins"* (i.e., let the verifier send messages depending upon, but not revealing, its coin tosses). This is indeed the approach taken in Section 4.9.

## 4.6.* Witness Indistinguishability and Hiding

In light of the non-closure of zero-knowledge under parallel composition (see Section 4.5.4), alternative "privacy" criteria that are preserved under parallel composition are of practical and theoretical importance. Two notions, called witness indistinguishability and witness hiding, that refer to the "privacy" of interactive proof systems (of languages in $\mathcal{NP}$) are presented in this section. Both notions seem weaker than zero-knowledge, yet they suffice for some specific applications.

We remark that witness indistinguishability and witness hiding, like zero-knowledge, are properties of the prover (and, more generally, of any interactive machine).

### 4.6.1. Definitions

In this section we confine ourselves to proof systems for languages in $\mathcal{NP}$. Recall that a *witness relation* for a language $L \in \mathcal{NP}$ is a binary relation $R_L$ that is polynomially bounded (i.e., $(x, y) \in R_L$ implies $|y| \leq \text{poly}(|x|)$), is polynomial-time-recognizable and characterizes $L$ by

$$L = \{x : \exists y \text{ s.t. } (x, y) \in R_L\}$$

For $x \in L$, any $y$ satisfying $(x, y) \in R_L$ is called a *witness* (for the membership $x \in L$). We let $R_L(x)$ denote the set of witnesses for the membership $x \in L$; that is, $R_L(x) \stackrel{\text{def}}{=} \{y : (x, y) \in R_L\}$.

[16] In fact, a super-logarithmic number of repetitions will suffice in the case of Construction 4.3.8, as well as for a modified version of Construction 4.4.7. The modified proof system invokes Construction 4.4.7 on the graph resulting from the input graph by applying a special polynomial-time reduction that is guaranteed by the so-called PCP theorem. Specifically, this reduction reduces $G3C$ to itself, so that non-members of $G3C$ are mapped into graphs for which every three-way partition of the vertex set has at least a constant fraction of violating edges (i.e., edges with both endpoints on the same side of the partition). Let $\varepsilon > 0$ be the constant guaranteed by the PCP theorem. Then the resulting proof system has perfect completeness and soundness error at most $1 - \varepsilon$, and so a super-logarithmic number of repetitions will yield negligible error probability.

### 4.6.1.1. Witness Indistinguishability

Loosely speaking, an interactive proof for a language $L \in \mathcal{NP}$ is *witness-independent* (resp., *witness-indistinguishable*) if the verifier's view of the interaction with the prover is statistically independent (resp., "computationally independent") of the auxiliary input of the prover. Actually, we shall specialize the requirement to the case in which the auxiliary input constitutes an $\mathcal{NP}$-witness to the common input; namely, for a witness relation $R_L$ of the language $L \in \mathcal{NP}$, we consider only interactions on common input $x \in L$, where the prover is given an auxiliary input in $R_L(x)$. By saying that the view is *computationally independent* of the witness, we mean that for every two choices of auxiliary inputs, the resulting views are computationally indistinguishable. Analogously to the discussion in Section 4.3, we obtain equivalent definitions by considering the verifier's view of the interaction with the prover or the verifier's output after such an interaction. In the actual definition, we adopt the latter (i.e., "output") formulation and use the notation of Definition 4.3.10.

**Definition 4.6.1 (Witness Indistinguishability/Independence):** *Let $(P, V)$, $L \in \mathcal{NP}$ and $V^*$ be as in Definition 4.3.10, and let $R_L$ be a fixed witness relation for the language $L$. We say that $(P, V)$ is* **witness-indistinguishable for** *$R_L$ if for every probabilistic polynomial-time interactive machine $V^*$ and every two sequences $W^1 = \{w_x^1\}_{x\in L}$ and $W^2 = \{w_x^2\}_{x\in L}$, such that $w_x^1, w_x^2 \in R_L(x)$, the following two ensembles are computationally indistinguishable:*

- $\{\langle P(w_x^1), V^*(z)\rangle(x)\}_{x\in L, z\in\{0,1\}^*}$
- $\{\langle P(w_x^2), V^*(z)\rangle(x)\}_{x\in L, z\in\{0,1\}^*}$

*Namely, for every probabilistic polynomial-time algorithm $D$, every polynomial $p(\cdot)$, all sufficiently long $x \in L$, and all $z \in \{0,1\}^*$, it holds that*

$$\left|\mathsf{Pr}\left[D\left(x, z, \left\langle P\left(w_x^1\right), V^*(z)\right\rangle(x)\right) = 1\right]\right.$$
$$\left. - \mathsf{Pr}\left[D\left(x, z, \left\langle P\left(w_x^2\right), V^*(z)\right\rangle(x)\right) = 1\right]\right| < \frac{1}{p(|x|)}$$

*We say that $(P, V)$ is* **witness-independent for** *$R_L$ if the foregoing ensembles are identically distributed. Namely, for every $x \in L$, every $w_x^1, w_x^2 \in R_L(x)$, and $z \in \{0,1\}^*$, the random variables $\langle P(w_x^1), V^*(z)\rangle(x)$ and $\langle P(w_x^2), V^*(z)\rangle(x)$ are identically distributed.*

In particular, $z$ may equal $(w_x^1, w_x^2)$. A few additional comments are in order:

- Proof systems in which the prover ignores its auxiliary input are (trivially) witness-independent. In particular, exponential-time provers can afford to ignore their auxiliary input (without any decrease in the probability that they will convince the verifier) and so can be trivially witness-independent. Yet probabilistic polynomial-time provers cannot afford to ignore their auxiliary input (since otherwise they become useless). Hence, for probabilistic polynomial-time provers (for languages outside $\mathcal{BPP}$), the witness-indistinguishability requirement may be non-trivial.

- Any zero-knowledge proof system for a language in $\mathcal{NP}$ is witness-indistinguishable (since the the distribution corresponding to each witness can be approximated by the same simulator; see details later). Likewise, perfect zero-knowledge proofs are witness-independent.
- On the other hand, witness indistinguishability does NOT imply zero-knowledge. In particular, any proof system for a language having unique witnesses is trivially witness-indistinguishable, but may not be zero-knowledge. For example, for a one-way permutation $f$, consider the ("unnatural") witness relation $\{(f(w), w) : w \in \{0,1\}^*\}$, characterizing the set of all strings, and a prover that on common input $f(w)$ and auxiliary input $w$ sends $w$ to the verifier.
- It is relatively easy to see that witness indistinguishability and witness independence are preserved under sequential composition. In the next subsection, we show that they are also preserved under parallel composition.

**An Augmented Notion.** An augmented notion of witness indistinguishability requires that whenever the common inputs to the proof system are computationally indistinguishable, so are the corresponding views of the verifier. That is, we augment Definition 4.6.1 as follows:

**Definition 4.6.2 (Strong Witness Indistinguishability):** *Let $(P, V)$ and all other notation be as in Definition 4.6.1. We say that $(P, V)$ is* **strongly witness-indistinguishable for** *$R_L$ if for every probabilistic polynomial-time interactive machine $V^*$ and for every two probability ensembles $\{(X_n^1, Y_n^1, Z_n^1)\}_{n\in\mathbb{N}}$ and $\{(X_n^2, Y_n^2, Z_n^2)\}_{n\in\mathbb{N}}$, such that each $(X_n^i, Y_n^i, Z_n^i)$ ranges over $(R_L \times \{0,1\}^*) \cap (\{0,1\}^n \times \{0,1\}^* \times \{0,1\}^*)$, the following holds:*

*If $\{(X_n^1, Z_n^1)\}_{n\in\mathbb{N}}$ and $\{(X_n^2, Z_n^2)\}_{n\in\mathbb{N}}$ are computationally indistinguishable, then so are $\{\langle P(Y_n^1), V^*(Z_n^1)\rangle(X_n^1)\}_{n\in\mathbb{N}}$ and $\{\langle P(Y_n^2), V^*(Z_n^2)\rangle(X_n^2)\}_{n\in\mathbb{N}}$.*

We stress that $\{(X_n^1, Y_n^1, Z_n^1)\}_{n\in\mathbb{N}}$ and $\{(X_n^2, Y_n^2, Z_n^2)\}_{n\in\mathbb{N}}$ are not required to be computationally indistinguishable; such a requirement would trivialize the definition at least as far as probabilistic polynomial-time provers are concerned. Definition 4.6.1 can be obtained from Definition 4.6.2 by considering the special case in which $X_n^1$ and $X_n^2$ are identically distributed (and observing that no computational requirement was placed on the $\{(X_n^i, Y_n^i, Z_n^i)\}_{n\in\mathbb{N}}$'s). On the other hand, assuming that one-way permutations exist, witness indistinguishability does *not* imply strong witness indistinguishability (see Exercise 25). Still, one can easily show that any zero-knowledge proof system for a language in $\mathcal{NP}$ is strongly witness-indistinguishable.

**Proposition 4.6.3:** *Let $(P, V)$ be an auxiliary-input zero-knowledge proof system for a language $L \in \mathcal{NP}$. Then $(P, V)$ is strongly witness-indistinguishable.*

***Proof Idea:*** Using the simulator $M^*$, guaranteed for $V^*$, we obtain that $E^i \stackrel{\text{def}}{=} \{\langle P(Y_n^i), V^*(Z_n^i)\rangle(X_n^i)\}_{n\in\mathbb{N}}$ and $S^i \stackrel{\text{def}}{=} \{M^*(X_n^i, Z_n^i)\}_{n\in\mathbb{N}}$ are computationally indistinguishable for both $i$'s. Thus, if $E^1$ and $E^2$ are not computationally

indistinguishable, then $S^1$ and $S^2$ are not computationally indistinguishable. Incorporating $M^*$ into the distinguisher, it follows that $\{(X_n^1, Z_n^1)\}_{n\in\mathbb{N}}$ and $\{(X_n^2, Z_n^2)\}_{n\in\mathbb{N}}$ are not computationally indistinguishable either. ■

### 4.6.1.2. Witness-Hiding

We now turn to the notion of witness-hiding. Intuitively, a proof system for a language in $\mathcal{NP}$ is witness-hiding if after interacting with the prover it is still infeasible for the verifier to find an $\mathcal{NP}$-witness for the common input. Clearly, such a requirement can hold only if it is infeasible to find witnesses from scratch. Because each $\mathcal{NP}$ language has instances for which witness-finding is easy, we must consider the task of witness-finding for specially selected hard instances. This leads to the following definitions.

**Definition 4.6.4 (Distribution of Hard Instances):** *Let $L \in \mathcal{NP}$, and let $R_L$ be a witness relation for $L$. Let $X \stackrel{\text{def}}{=} \{X_n\}_{n\in\mathbb{N}}$ be a probability ensemble such that $X_n$ ranges over $L \cap \{0,1\}^n$. We say that $X$ is* **hard for** *$R_L$ if for every probabilistic polynomial-time (*witness-finding*) algorithm $F$, every polynomial $p(\cdot)$, all sufficiently large $n$'s, and all $z \in \{0,1\}^{\text{poly}(n)}$,*

$$\mathsf{Pr}[F(X_n, z) \in R_L(X_n)] < \frac{1}{p(n)}$$

For example, if $f$ is a (length-preserving and non-uniformly) one-way function, then the probability ensemble $\{f(U_n)\}_{n\in\mathbb{N}}$ is hard for the witness relation $\{(f(w), w) : w \in \{0,1\}^*\}$, where $U_n$ is uniform over $\{0,1\}^n$.

**Definition 4.6.5 (Witness-Hiding):** *Let $(P, V)$, $L \in \mathcal{NP}$, and $R_L$ be as in the foregoing definitions. Let $X = \{X_n\}_{n\in\mathbb{N}}$ be a hard-instance ensemble for $R_L$. We say that $(P, V)$ is* **witness-hiding for** the relation $R_L$ **under** the instance ensemble *$X$ if for every probabilistic polynomial-time machine $V^*$, every polynomial $p(\cdot)$, all sufficiently large $n$'s, and all $z \in \{0,1\}^*$,*

$$\mathsf{Pr}[\langle P(Y_n), V^*(z)\rangle(X_n) \in R_L(X_n)] < \frac{1}{p(n)}$$

*where $Y_n$ is arbitrarily distributed over $R_L(X_n)$.*

We remark that the relationship between the two privacy criteria (i.e., witness indistinguishability and witness-hiding) is not obvious. Yet zero-knowledge proofs (for $\mathcal{NP}$) are also witness-hiding (for any corresponding witness relation and any hard distribution). We mention a couple of extensions of Definition 4.6.5:

1. One can say that $(P, V)$ is *universally witness-hiding for $R_L$* if the proof system $(P, V)$ is witness-hiding for $R_L$ under every ensemble of hard instances for $R_L$. (Alternatively, one can require only that $(P, V)$ be witness-hiding for $R_L$ under every *efficiently constructible*[17] ensemble of hard instances for $R_L$.)

[17] See Definition 3.2.5.

**2.** Variants of the foregoing definitions, in which the auxiliary input $z$ is replaced by a distribution $Z_n$ that may depend on $X_n$, are of interest too. Here we consider ensembles $\{(X_n, Z_n)\}_{n\in\mathbb{N}}$, where $X_n$ ranges over $L \cap \{0,1\}^n$. Such an ensemble is *hard for* $R_L$ if for every probabilistic polynomial-time algorithm $F$, the probability that $F(X_n, Z_n) \in R_L(X_n)$ is negligible. The system $(P, V)$ is witness-hiding for $R_L$ under $\{(X_n, Z_n)\}_{n\in\mathbb{N}}$ if for every probabilistic polynomial-time verifier $V^*$, the probability that $\langle P(Y_n), V^*(Z_n)\rangle(X_n) \in R_L(X_n)$ is negligible.

### 4.6.2. Parallel Composition

In contrast to zero-knowledge proof systems, witness-indistinguishable proofs offer some robustness under parallel composition. Specifically, parallel composition of witness-indistinguishable proof systems results in a witness-indistinguishable system, *provided that the original prover is probabilistic polynomial-time*.

**Lemma 4.6.6 (Parallel-Composition Lemma for Witness Indistinguishability):** *Let $L \in \mathcal{NP}$ and $R_L$ be as in Definition 4.6.1, and suppose that $P$ is probabilistic polynomial-time and $(P, V)$ is* witness-indistinguishable (resp., witness-independent) *for $R_L$. Let $Q(\cdot)$ be a polynomial, and let $P_Q$ denote a program that on common input $x_1, \ldots, x_{Q(n)} \in \{0,1\}^n$ and auxiliary input $w_1, \ldots, w_{Q(n)} \in \{0,1\}^*$ invokes $P$ in parallel $Q(n)$ times, so that in the $i$th copy $P$ is invoked on common input $x_i$ and auxiliary input $w_i$. Then $P_Q$ is* witness-indistinguishable (resp., witness-independent) *for*

$$R_L^Q \stackrel{\text{def}}{=} \{(\overline{x}, \overline{w}) : \forall i, (x_i, w_i) \in R_L\}$$

*where $\overline{x} = (x_1, \ldots, x_m)$ and $\overline{w} = (w_1, \ldots, w_m)$, so that $m = Q(n)$ and $|x_i| = n$ for each $i$.*

***Proof Sketch:*** Both the computational and information-theoretic versions follow by a hybrid argument. We concentrate on the computational version. To avoid cumbersome notation, we consider a generic $n$ for which the claim of the lemma fails. (By contradiction, there must be infinitely many such $n$'s, and a precise argument will actually handle all these $n$'s together.) Namely, suppose that by using a verifier program $V_Q^*$ it is feasible to distinguish the witnesses $\overline{w}^1 = (w_1^1, \ldots, w_m^1)$ and $\overline{w}^2 = (w_1^2, \ldots, w_m^2)$ used by $P_Q$ in an interaction on common input $\overline{x} \in L^m$. Then for some $i$, the program $V_Q^*$ also distinguishes the hybrid witnesses $\overline{h}^{(i)} = (w_1^1, \ldots, w_i^1, w_{i+1}^2, \ldots, w_m^2)$ and $\overline{h}^{(i+1)} = (w_1^1, \ldots, w_{i+1}^1, w_{i+2}^2, \ldots, w_m^2)$. Rewrite $\overline{h}^{(i)} = (w_1, \ldots, w_i, w_{i+1}^2, w_{i+2}, \ldots, w_m)$ and $\overline{h}^{(i+1)} = (w_1, \ldots, w_i, w_{i+1}^1, w_{i+2}, \ldots, w_m)$, where $w_j \stackrel{\text{def}}{=} w_j^1$ if $j \leq i$, and $w_j \stackrel{\text{def}}{=} w_j^2$ if $j \geq i+2$. We derive a contradiction by constructing a verifier $V^*$ that distinguishes (the witnesses used by $P$ in) interactions with the original prover $P$. Details follow.

The program $V^*$ incorporates the programs $P$ and $V_Q^*$ and proceeds by interacting with the actual prover $P$ and simulating $m-1$ other interactions with (a copy of program) $P$. The real interaction with $P$ is viewed as the $i+1$ copy in

an interaction of $V_Q^*$ (with $P_Q$), whereas the simulated interactions are associated with the other copies. Specifically, in addition to the common input $x$, machine $V^*$ gets the appropriate $i$ and the sequences $\overline{x}$ and $(w_1, \ldots, w_i, w_{i+2}, \ldots, w_m)$ as part of its auxiliary input. For each $j \neq i+1$, machine $V^*$ will use $x_j$ as common input and $w_j$ as auxiliary input to the $j$th copy of $P$. Machine $V^*$ invokes $V_Q^*$ on common input $\overline{x}$ and provides it an interface to a virtual interaction with $P_Q$. The $i+1$ component of a message $\overline{\alpha} = (\alpha_1, \ldots, \alpha_m)$ sent by $V_Q^*$ is forwarded to the prover $P$, and all other components are kept for the simulation of the other copies. When $P$ answers with a message $\beta$, machine $V^*$ computes the answers for the other copies of $P$ (by feeding the program $P$ the corresponding auxiliary input and the corresponding sequence of incoming messages). It follows that $V^*$ can distinguish the case in which $P$ uses the witness $w_{i+1}^1$ from the case in which $P$ uses $w_{i+1}^2$. ■

This proof easily extends to the case in which several proof systems are executed concurrently in a totally asynchronous manner (i.e., sequential and parallel executions being two special cases).[18] The proof also extends to strong witness indistinguishability. Thus we have the following:

**Lemma 4.6.7 (Parallel Composition for Strong Witness Indistinguishability):** *Let $L \in \mathcal{NP}$, $R_L$, $(P, V)$, $Q$, $R_L^Q$, and $P_Q$ be as in Lemma 4.6.6. Then if $P$ is strongly witness-indistinguishable* (for $R_L$)*, then so is $P_Q$* (for $R_L^Q$)*.*

### 4.6.3. Constructions

In this section we present constructions of witness-indistinguishable and witness-hiding proof systems.

#### 4.6.3.1. Constructions of Witness-Indistinguishable Proofs

Using the parallel-composition lemma (Lemma 4.6.7) and the observation that zero-knowledge proofs are (strongly) witness-indistinguishable, we derive the following:

**Theorem 4.6.8:** *Assuming the existence of (non-uniformly) one-way functions, every language in $\mathcal{NP}$ has a* constant-round (strongly) *witness-indistinguishable proof system with* negligible error probability. *In fact, the error probability can be made exponentially small.*

We remark that no such result is known for *zero-knowledge* proof systems. Namely, the known proof systems for $\mathcal{NP}$ variously

- are not constant-round (e.g., Construction 4.4.9), or
- have noticeable error probability (e.g., Construction 4.4.7), or

[18] That is, executions of polynomially many instances of the proof system are arbitrarily interleaved (in a manner determined by the adversary); see suggestions for further reading in Section 4.12.2.

- require stronger intractability assumptions (see Section 4.9.1), or
- are only computationally sound (see Section 4.9.2).

### 4.6.3.2. Constructions of Witness-Hiding Proofs

Witness-indistinguishable proof systems are not necessarily witness-hiding. For example, any language with unique witnesses has a proof system that yields the unique witness (and so may fail to be witness-hiding), yet this proof system is trivially witness-independent. On the other hand, for some relations, witness indistinguishability implies witness-hiding, *provided that the prover is probabilistic polynomial-time*. For example:

**Proposition 4.6.9:** *Let* $\{(f_i^0, f_i^1) : i \in \overline{I}\}$ *be a collection of (non-uniform) claw-free functions, and let*

$$R \stackrel{\text{def}}{=} \{(x, w) : w = (\sigma, r) \wedge x = (i, x') \wedge x' = f_i^\sigma(r)\}$$

*Suppose that* $P$ *is a probabilistic polynomial-time interactive machine that is witness-indistinguishable for* $R$*. Then* $P$ *is also witness-hiding for* $R$ *under the distribution generated by setting* $i = I(1^n)$ *and* $x' = f_i^0(D(0, i))$*, where* $I$ *and* $D$ *are as in Definition 2.4.6.*

By a collection of non-uniform claw-free functions we mean that even non-uniform families of circuits $\{C_n\}$ fail to form claws on input distribution $I(1^n)$, except with negligible probability. We remark that the foregoing proposition does not relate to the purpose of interacting with $P$ (e.g., whether $P$ is proving membership in a language, knowledge of a witness, and so on).

***Proof Idea:*** The proposition is proved by contradiction. Suppose that some probabilistic polynomial-time interactive machine $V^*$ finds witnesses after interacting with $P$. By the witness indistinguishability of $P$, it follows that $V^*$ is performing equally well regardless of whether the witness used by $P$ is of the form $(0, \cdot)$ or is of the form $(1, \cdot)$. Combining the programs $V^*$ and $P$ with the algorithm $D$, we derive a claw-forming algorithm (and hence a contradiction). Specifically, the claw-forming algorithm, on input $i \in \overline{I}$, uniformly selects $\sigma \in \{0, 1\}$, randomly generates $r = D(\sigma, i)$, computes $x = (i, f_i^\sigma(r))$, and emulates an interaction of $V^*$ with $P$ on common input $x$ and auxiliary input $(\sigma, r)$ to $P$. By the witness indistinguishability of $P$, the output of $V^*$ is computationally independent of the value of $\sigma$. Therefore, if on common input $x$, machine $V^*$ outputs a witness $w \in R(x)$, then, with probability approximately $\frac{1}{2}$, we have $w = (1 - \sigma, r')$, and a claw is formed (since $f_i^\sigma(r) = f_i^{1-\sigma}(r')$). Finally, observe that we need to analyze the performance of the claw-forming algorithm on input distribution $I(1^n)$, and in this case the common input in the emulation of $(P, V^*)$ is distributed as in the hypothesis of the proposition. ■

Furthermore, every $\mathcal{NP}$-relation can be "slightly modified" so that, for the modified relation, witness indistinguishability implies witness hiding. Given a relation $R$, the

modified relation, denoted $R_2$, is defined by

$$R_2 \stackrel{\text{def}}{=} \{((x_1, x_2), w) : |x_1| = |x_2| \wedge \exists i \text{ s.t. } (x_i, w) \in R\} \tag{4.5}$$

Namely, $w$ is a witness under $R_2$ for the instance $(x_1, x_2)$ if and only if $w$ is a witness under $R$ for either $x_1$ or $x_2$.

**Proposition 4.6.10:** *Let $R$ and $R_2$ be as before, and let $P$ be a probabilistic polynomial-time interactive machine that is witness-indistinguishable for $R_2$. Then $P$ is witness-hiding for $R_2$ under every distribution of pairs of hard instances induced by an efficient algorithm that randomly selects pairs in $R$. That is:*

> *Let $S$ be a probabilistic polynomial-time algorithm that on input $1^n$ outputs $(x, w) \in R$, so that $|x| = n$ and $X_n$ denotes the distribution induced on the first element in the output of $S(1^n)$. Suppose that $\{X_n\}_{n\in\mathbb{N}}$ is an ensemble of hard instances for $R$. Then $P$ is witness-hiding under the ensemble $\{(X_n^{(1)}, X_n^{(2)})\}_{n\in\mathbb{N}}$, where $X_n^{(1)}$ and $X_n^{(2)}$ denote two independent copies of $X_n$.*

***Proof Idea:*** Let $S$ and $\{X_n\}_{n\in\mathbb{N}}$ be in the hypothesis. Suppose that some interactive machine $V^*$ finds witnesses, with non-negligible probability (under the foregoing distribution), after interacting with $P$. By the witness indistinguishability of $P$ it follows that $V^*$ is performing equally well regardless of whether the witness $w$ used by $P$ on common input $(x_1, x_2)$ satisfies $(x_1, w) \in R$ or $(x_2, w) \in R$. Combining the programs $V^*$ and $P$ with the algorithm $S$, we derive an algorithm, denoted $F^*$, that finds witnesses for $R$ (under the distribution $X_n$): On input $x \in L$, algorithm $F^*$ generates at random $(x', w') = S(1^{|x|})$ and sets $\overline{x} = (x, x')$ with probability $\frac{1}{2}$, and $\overline{x} = (x', x)$ otherwise. Algorithm $F^*$ emulates an interaction of $V^*$ with $P$ on common input $\overline{x}$ and auxiliary input $w'$ to $P$, and when $V^*$ outputs a witness $w$, algorithm $F^*$ checks whether or not $(x, w) \in R$. By the witness indistinguishability of $P$, the verifier cannot tell the case in which $P$ uses a witness to the first element of $\overline{x}$ from the case in which it uses a witness to the second. Also, by construction of $F^*$, if the input to $F^*$ is distributed as $X_n$, then the proof system is emulated on common input $(X_n^{(1)}, X_n^{(2)})$, where $X_n^{(1)}$ and $X_n^{(2)}$ denote two independent copies of $X_n$. Thus, by the foregoing hypothesis, $V^*$ finds a witness for $x$ with non-negligible probability (taken over the distribution of $x$ and the random choices of $F^*$). It follows that $\{X_n\}_{n\in\mathbb{N}}$ is not hard for $R$. ■

### 4.6.4. Applications

Applications of the notions presented in this section are scattered in various places in this book. In particular, strong witness-indistinguishable proof systems are used in the construction of constant-round zero-knowledge arguments for $\mathcal{NP}$ (see Section 4.9.2), witness-independent proof systems are used in the zero-knowledge proof for Graph Non-Isomorphism (see Section 4.7.4.3), and witness-hiding proof systems are used for the efficient identification scheme based on factoring (in Section 4.7.5).

## 4.7.* Proofs of Knowledge

This section addresses the concept of "proofs of knowledge." Loosely speaking, these are proofs in which the prover asserts "knowledge" of some object (e.g., a 3-coloring of a graph) and not merely its existence (e.g., the existence of a 3-coloring of the graph, which in turn implies that the graph is in the language $G3C$). But what is meant by saying that a machine knows something? Indeed, the main thrust of this section is to address this question. Before doing so, we point out that "proofs of knowledge," and in particular zero-knowledge "proofs of knowledge," have many applications to the design of cryptographic schemes and cryptographic protocols. Some of these applications are discussed in Section 4.7.4. Of special interest is the application to identification schemes, which is discussed in Section 4.7.5. Finally, in Section 4.7.6 we introduce the notion of strong proofs of knowledge.

### 4.7.1. Definition

#### 4.7.1.1. A Motivating Discussion

*What does it mean to say that a* MACHINE *knows something?* Any standard dictionary suggests several meanings for the verb *to know,* and most meanings are phrased with reference to *awareness*, a notion that is certainly inapplicable in our context. We must look for a *behavioristic* interpretation of the verb *to know*. Indeed, it is reasonable to link knowledge with ability to do something, be it (at the least) the ability to write down whatever one knows. Hence, we shall say that a machine knows a string $\alpha$ if it *can* output the string $\alpha$. But this seems total nonsense: A machine has a well-defined output – either the output equals $\alpha$ or it does not. *So what can be meant by saying that a machine can do something?* Loosely speaking, it means that the machine can be *easily modified* so that it will do whatever is claimed. More precisely, it means that there exists an *efficient* machine that, using the original machine as a black box, outputs whatever is claimed.

So much for defining the "knowledge of machines." Yet, whatever a machine knows or does not know is "its own business." What can be of interest and reference *to the outside* is the question of what can be deduced about the knowledge of a machine after interacting with it. Hence, we are interested in proofs of knowledge (rather than in mere knowledge).

For the sake of simplicity, let us consider a concrete question: *How can a machine prove that it knows a 3-coloring of a graph?* An obvious way is simply to send the 3-coloring to the verifier. Yet we claim that applying Construction 4.4.7 (i.e., the zero-knowledge proof system for $G3C$) sufficiently many times results in an alternative way of proving knowledge of a 3-coloring of the graph.

Loosely speaking, we say that an interactive machine $V$ constitutes a *verifier for knowledge* of 3-coloring if the probability that the verifier is convinced by a machine $P$ to accept the graph $G$ is inversely proportional to the difficulty of extracting a 3-coloring of $G$ when using machine $P$ as a black box. Namely, the extraction of the 3-coloring is done by an oracle machine, called an *extractor*, that is given access

to a function specifying the behavior of $P$ (i.e., the messages it sends in response to particular messages it may receive). We require that the (expected) running time of the extractor, on input $G$ and with access to an oracle specifying $P$'s messages, be inversely related (by a factor polynomial in $|G|$) to the probability that $P$ convinces $V$ to accept $G$. In case $P$ always convinces $V$ to accept $G$, the extractor runs in expected polynomial time. The same holds in case $P$ convinces $V$ to accept with noticeable probability. (We stress that the latter special cases do not suffice for a satisfactory definition; see advanced comment in Section 4.7.1.4.)

### 4.7.1.2. Technical Preliminaries

Let $R \subseteq \{0,1\}^* \times \{0,1\}^*$ be a binary relation. Then $R(x) \stackrel{\text{def}}{=} \{s : (x,s) \in R\}$ and $L_R \stackrel{\text{def}}{=} \{x : \exists s \text{ s.t. } (x,s) \in R\}$. If $(x,s) \in R$, then we call $s$ a *solution* for $x$. We say that $R$ is *polynomially bounded* if there exists a polynomial $p$ such that $|s| \leq p(|x|)$ for all $(x,s) \in R$. We say that $R$ is an *$\mathcal{NP}$-relation* if $R$ is polynomially bounded and, in addition, there exists a polynomial-time algorithm for deciding membership in $R$ (indeed, it follows that $L_R \in \mathcal{NP}$). In the sequel, we confine ourselves to polynomially bounded relations. In fact, all the applications presented in this book refer to $\mathcal{NP}$-relations.

We wish to be able to consider in a uniform manner all potential provers, without making distinctions based on their running time, internal structure, and so forth. Yet we observe that these interactive machines can be given auxiliary input that will enable them to "know" and to prove more. Likewise, they may have the good fortune to select a random input that will be more enabling than another. Hence, statements concerning the knowledge of the prover refer not only to the prover's program but also to the specific auxiliary and random inputs it receives. Therefore, we fix an interactive machine as well as all the inputs (i.e., the common input, the auxiliary input, and the random input) to this machine. For such a prover + inputs template, we consider both the acceptance probability (of the verifier) when interacting with this template and the use of this template as an oracle to a "knowledge extractor." This motivates the following definition.

**Definition 4.7.1 (Message-Specification Function):** *Denote by $P_{x,y,r}(\overline{m})$ the message sent by machine $P$ on common input $x$, auxiliary input $y$, and random input $r$, after receiving messages $\overline{m}$. The function $P_{x,y,r}$ is called the* **message-specification function** *of machine $P$ with common input $x$, auxiliary input $y$, and random input $r$.*

An oracle machine with access to the function $P_{x,y,r}$ will represent the knowledge of machine $P$ on common input $x$, auxiliary input $y$, and random input $r$. This oracle machine, called the knowledge extractor, will try to find a solution to $x$ (i.e., an $s \in R(x)$). The running time of the extractor will be required to be inversely related to the corresponding acceptance probability (of the verifier when interacting with $P$ on common input $x$ and when $P$ has auxiliary input $y$ and random input $r$.)

### 4.7.1.3. Knowledge Verifiers

Now that all the machinery is ready, we present the definition of a system for proofs of knowledge. Actually, the definition is a generalization (to be motivated by the subsequent applications) in which we allow an error parameter specified by the function $\kappa$. At first reading, one can set the function $\kappa$ to be identically zero.

**Definition 4.7.2 (System for Proofs of Knowledge):** *Let $R$ be a binary relation and $\kappa : \mathbb{N} \to [0, 1]$. We say that an interactive function $V$ is a* **knowledge verifier for the relation** *$R$ with* **knowledge error** *$\kappa$ if the following two conditions hold:*

- Non-triviality: *There exists an interactive machine $P$ such that for every $(x, y) \in R$ all possible interactions of $V$ with $P$ on common input $x$ and auxiliary input $y$ are accepting.*
- Validity (with error $\kappa$): *There exists a polynomial $q(\cdot)$ and a probabilistic oracle machine $K$ such that for every interactive function $P$, every $x \in L_R$, and every $y, r \in \{0, 1\}^*$, machine $K$ satisfies the following condition:*

  *Denote by $p(x, y, r)$ the probability that the interactive machine $V$ accepts, on input $x$, when interacting with the prover specified by $P_{x,y,r}$. If $p(x, y, r) > \kappa(|x|)$, then, on input $x$ and with access to oracle $P_{x,y,r}$, machine $K$ outputs a solution $s \in R(x)$ within an expected number of steps bounded by*

  $$\frac{q(|x|)}{p(x, y, r) - \kappa(|x|)}$$

  *The oracle machine $K$ is called a* universal knowledge extractor.

*When $\kappa(\cdot)$ is identically zero, we simply say that $V$ is a* knowledge verifier for the relation *$R$. An interactive pair $(P, V)$ such that $V$ is a knowledge verifier for a relation $R$ and $P$ is a machine satisfying the non-triviality condition (with respect to $V$ and $R$) is called a* system for proofs of knowledge for the relation *$R$*.

An alternative formulation of the validity condition follows. It postulates the existence of a probabilistic oracle machine $K$ (as before). However, instead of requiring $K^{P_{x,y,r}}(x)$ to *always* output a solution within an expected time inversely proportional to $p(x, y, r) - \kappa(|x|)$, the alternative requires $K^{P_{x,y,r}}(x)$ to run in expected *polynomial time* and output a solution with probability at least $p(x, y, r) - \kappa(|x|)$. In fact, we can further relax the alternative formulation by requiring that a solution be output with probability at least $(p(x, y, r) - \kappa(|x|))/\text{poly}(|x|)$.

**Definition 4.7.3 (Validity with Error $\kappa$, Alternative Formulation):** *Let $V$, $P_{x,y,r}$ (with $x \in L_R$), and $p(x, y, r)$ be as in Definition 4.7.2. We say that $V$* **satisfies the alternative validity condition with error** *$\kappa$ if there exists a probabilistic oracle machine $K$ and a positive polynomial $q$ such that on input $x$ and with access to oracle $P_{x,y,r}$, machine $K$ runs in expected polynomial time and outputs a solution $s \in R(x)$ with probability at least $(p(x, y, r) - \kappa(|x|))/q(|x|)$.*

The two formulations of validity are equivalent in the case of $\mathcal{NP}$-relations. The idea underlying the equivalence is that, in the current context, success probability and expected running time can be converted from one to the other.

**Proposition 4.7.4:** *Let $R$ be an $\mathcal{NP}$-relation, and let $V$ be an interactive machine. Referring to this relation $R$, machine $V$ satisfies (with error $\kappa$) the validity condition of Definition 4.7.2 if and only if $V$ satisfies (with error $\kappa$) the alternative validity condition of Definition 4.7.3.*

***Proof Sketch:*** Suppose that $V$ satisfies the alternative formulation (with error $\kappa$), and let $K$ be an adequate extractor and $q$ an adequate polynomial. Using the hypothesis that $R$ is an $\mathcal{NP}$-relation, it follows that when invoking $K$ we can determine whether or not $K$ has succeeded. Thus, we can iteratively invoke $K$ until it succeeds. If $K$ succeeds with probability $s(x, y, r) \geq (p(x, y, r) - \kappa(|x|))/q(|x|)$, then the expected number of invocations is $1/s(x, y, r)$, which is as required in Definition 4.7.2.

Suppose that $V$ satisfies (with error $\kappa$) the validity requirement of Definition 4.7.2, and let $K$ be an adequate extractor and $q$ an adequate polynomial (such that $K$ runs in expected time $q(|x|)/(p(x, y, r) - \kappa(|x|))$). Let $p$ be a polynomial bounding the length of solutions for $R$ (i.e., $(x, s) \in R$ implies $|s| \leq p(|x|)$). Then we proceed with up to $p(|x|)$ iterations: In the $i$th iteration, we emulate the computation of $K^{P_{x,y,r}}(x)$ with time bound $2^{i+1} \cdot q(|x|)$. In case the current iteration yields a solution, we halt outputting this solution. Otherwise, with probability $\frac{1}{2}$, we continue to the next iteration (and with probability $\frac{1}{2}$ we halt with a special failure symbol). In case the last iteration is completed without obtaining a solution, we simply find a solution by exhaustive search (using time $2^{p(|x|)} \cdot \mathrm{poly}(|x|)$). Observe that the $i$th iteration is executed with probability at most $2^{-(i-1)}$, and so our expected running time is at most

$$\begin{aligned}\sum_{i=1}^{p(|x|)} 2^{-(i-1)} \cdot (2^{i+1} \cdot q(|x|)) + 2^{-p(|x|)} \cdot \left(2^{p(|x|)} \cdot \mathrm{poly}(|x|)\right) \\ = 4 \cdot p(|x|) \cdot q(|x|) + \mathrm{poly}(|x|)\end{aligned}$$

To evaluate the success probability of the new extractor, note that the probability that $K^{P_{x,y,r}}(x)$ will run for more than twice its expected running time (i.e., twice $q(|x|)/(p(x, y, r) - \kappa(|x|))$) is less than $\frac{1}{2}$. Also observe that in iteration $i \stackrel{\text{def}}{=} -\log_2(p(x, y, r) - \kappa(|x|))$ we emulate these many steps (i.e., $2q(|x|)/(p(x, y, r) - \kappa(|x|))$ steps). Thus, the probability that we can extract a solution in one of the first $i$ iterations is at least $\frac{1}{2} \cdot 2^{-(i-1)} = p(x, y, r) - \kappa(|x|)$, as required in the alternative formulation. ■

**Comment.** The proof of Proposition 4.7.4 actually establishes that the formulation of Definition 4.7.2 implies the formulation of Definition 4.7.3 with $q \equiv 1$. Thus, the formulation of Definition 4.7.3 with $q \equiv 1$ is equivalent to its general formulation (i.e.,

with an arbitrary polynomial $q$). We shall use this fact in the proofs of Propositions 4.7.5 and 4.7.6.

### 4.7.1.4. Discussion

In view of Proposition 4.7.4, we can freely use either of the two formulations of validity. The formulation of Definition 4.7.2 typically is more convenient when analyzing the effect of a proof of knowledge as a sub-protocol, whereas the formulation of Definition 4.7.3 typically is more convenient when demonstrating that a given system is a proof of knowledge. We mention that variants of Proposition 4.7.4 also hold when $R$ is not an $\mathcal{NP}$-relation (see Exercise 29).

**A Reflection.** The notion of a proof of knowledge (and, more so, the notion of a knowledge extractor used in formalizing it) is related to the simulation paradigm. This relation is evident in applications in which the knowledge verifier takes some action $\mathcal{A}$ after being convinced that the knowledge prover knows $\mathcal{K}$, where action $\mathcal{A}$ is such that it causes no harm to the knowledge verifier if the knowledge prover indeed knows $\mathcal{K}$. Following the simulation paradigm, our definition asserts that if action $\mathcal{A}$ is taken after the verifier becomes convinced that the prover knows $\mathcal{K}$, then no harm is caused, since in some sense we can simulate a situation in which the prover actually knows $\mathcal{K}$. Indeed, using the knowledge extractor, we can simulate the prover's view of the entire interaction (i.e., the proof process and the action taken afterward by the convinced verifier): In case the prover fails, action $\mathcal{A}$ is not taken, and so the entire interaction is easy to simulate. In case the prover succeeds in convincing the verifier, we extract the relevant knowledge $\mathcal{K}$ and reach a situation in which action $\mathcal{A}$ causes no harm (i.e., $\mathcal{A}$ can be simulated based on $\mathcal{K}$).

**About Soundness.** In the foregoing definitions, we imposed no requirements regarding what happens when the knowledge verifier for $R$ is invoked on common input not in $L_R$. The natural requirement is that on input $x \notin L_R$ the verifier will accept with probability at most $\kappa(|x|)$. This holds in many natural cases, but not in the conclusion of Proposition 4.7.6. See further comments in Sections 4.7.3–4.7.6.

**An Advanced Comment.** A key feature of both formulations of validity is that they handle all possible values of $p(x, y, r)$ in a "uniform" fashion. This is crucial to most applications (e.g., see Section 4.7.4) in which a proof of knowledge is used as a sub-protocol (rather than as the end protocol). Typically, in the former applications (i.e., using a proof of knowledge as a sub-protocol), the knowledge error function is required to be negligible (or even zero). In such cases, we need to deal with all possible values of $p(x, y, r)$ that are not negligible, but we do not know a priori the value of $p(x, y, r)$. We warn that the fact that $p(x, y, r)$ is not negligible (as a function of $|x|$) does *not* mean that it is noticeable (as a function of $|x|$).[19]

[19] Recall that a function $\mu : \mathbb{N} \to \mathbb{R}$ is negligible if for every positive polynomial $p$ and all sufficiently large $n$'s, it holds that $\mu(n) < 1/p(n)$, whereas a function $\nu : \mathbb{N} \to \mathbb{R}$ is noticeable if there exists a polynomial $p$ such that for all sufficiently large $n$'s, it holds that $\nu(n) > 1/p(n)$. A function $f : \mathbb{N} \to \mathbb{R}$ may be neither negligible nor noticeable: For example, consider the function $f$, defined by $f(n) \stackrel{\text{def}}{=} 2^{-n}$ if $n$ is odd, and $f(n) \stackrel{\text{def}}{=} n^{-2}$ otherwise.

### 4.7.2. Reducing the Knowledge Error

The knowledge error can be reduced by sequential repetitions of the proof system. Specifically, the error drops exponentially with the number of repetitions.

**Proposition 4.7.5:** *Let $R$ be a polynomially bounded relation, and let $t : \mathbb{N} \to \mathbb{N}$ be a polynomially bounded function. Suppose that $(P, V)$ is a system for proof of knowledge for the relation $R$ with knowledge error $\kappa$. Then the proof system that results by repeating $(P, V)$ sequentially $t(|x|)$ times on common input $x$ is a system for proof of knowledge for the relation $R$ with knowledge error $\kappa'(n) \stackrel{\text{def}}{=} \kappa(n)^{t(n)}$.*

***Proof Sketch:*** Let $(P', V')$ denote the protocol obtained by $t$ sequential repetitions of $(P, V)$, as in the proposition. To analyze the validity property of $V'$, we use the formulation of Definition 4.7.3. Given an extractor $K_1$ for the basic system, we construct an extractor $K$ for the composed system $(P', V')$ as follows. On input $x$, machine $K$ uniformly selects $i \in \{1, \ldots, t(|x|)\}$, emulates the first $i - 1$ iterations of the basic proof system $(P, V)$, and invokes $K_1$ with oracle access to the residual prover determined by the transcript of these $i - 1$ iterations. That is, given oracle access to a prover strategy for the composed proof system, we first use it to emulate the first $i - 1$ iterations in $(P', V')$, resulting in a transcript $\alpha$. We then define a prover strategy for the basic proof system by considering the way in which the composed-system prover behaves in the $i$th iteration given that $\alpha$ is the transcript of the first $i - 1$ iterations. Using this (basic-system) prover strategy as oracle, we invoke $K_1$ in an attempt to extract a solution to $x$.

It is left to analyze the success probability of extractor $K$ for fixed $x \in L_R$ and $y$ and $r$ as before. Let $t \stackrel{\text{def}}{=} t(|x|)$, $\kappa \stackrel{\text{def}}{=} \kappa(|x|)$, and $\delta \stackrel{\text{def}}{=} p(x, y, r) - \kappa^t$. (We shall omit this fixed $(x, y, r)$ also from the following notations.) Our aim is to show that $K$ extracts a solution for $x$ with probability at least $\delta/\text{poly}(|x|)$. Toward this goal, let us denote by $a_{i-1}$ the probability that the verifier accepts in each of the first $i - 1$ iterations of the composed proof system. For every possible transcript $\alpha$ of the first $i - 1$ iterations, we denote by $p'(\alpha)$ the probability that the verifier accepts in the $i$th iteration when $\alpha$ describes the transcript of the first $i - 1$ iterations. Note that if $K$ tries to extract a solution after emulating $i - 1$ iterations resulting in transcript $\alpha$, then its success probability is at least $p'(\alpha) - \kappa$. Let $c_i$ be the expected value of $p'(\alpha)$ when $\alpha$ is selected at random among all $(i - 1)$-iteration transcripts in which the verifier accepts. Then $a_i = a_{i-1} \cdot c_i$, and the probability that $K$ extracts a solution after emulating $i - 1$ iterations is at least $a_{i-1} \cdot (c_i - \kappa)$.

**Claim:** Either $c_1 - \kappa \geq \delta/t$ or there exists an $i \geq 2$ such that $a_{i-1} \cdot (c_i - \kappa) > \delta/t$.

In both cases we have established an adequate lower bound on the success probability of $K$; that is, $K$ succeeds with probability at least $\delta/t^2$, where an extra factor of $t$ is to account for the probability that $K$ will select a good $i$.

**Proof:** Observe that if $a_1 = c_1 < \kappa + (\delta/t)$, then there must exist an $i \geq 2$ such that $a_{i-1} < \kappa^{i-1} + ((i - 1)\delta/t)$ and $a_i \geq \kappa^i + (i\delta/t)$, since $a_t = \kappa^t + \delta$. Using

this $i$, we have

$$\begin{aligned} a_{i-1} \cdot (c_i - \kappa) &= a_i - a_{i-1} \cdot \kappa \\ &> \left(\kappa^i + \frac{i\delta}{t}\right) - \left(\kappa^i + \frac{(i-1)\delta}{t}\right) \\ &= \frac{\delta}{t} \end{aligned}$$

The claim follows, and so does the proposition. ■

**What About Parallel Composition?** As usual (see Section 4.3.4), the effect of parallel composition is more complex than the effect of sequential composition. Consequently, a result analogous to Proposition 4.7.5 is not known to hold in general. Still, parallel execution of some popular zero-knowledge proofs of knowledge can be shown to reduce the knowledge error exponentially in the number of repetitions; see Exercise 27.

**Getting Rid of Tiny Error.** For $\mathcal{NP}$-relations, whenever the knowledge error is smaller than the probability of finding a solution by a random guess, one can set the knowledge error to zero.

**Proposition 4.7.6:** *Let $R$ be an $\mathcal{NP}$-relation, and let $q(\cdot)$ be a polynomial such that $(x, y) \in R$ implies $|y| \leq q(|x|)$. Suppose that $(P, V)$ is a system for proofs of knowledge for the relation $R$, with knowledge error $\kappa(n) \stackrel{\text{def}}{=} 2^{-q(n)}$. Then $(P, V)$ is a system for proofs of knowledge for the relation $R$ (with zero knowledge error).*

***Proof Sketch:*** Again, we use the formulation of Definition 4.7.3. Given a knowledge extractor $K$ substantiating the hypothesis, we construct a new knowledge extractor that first invokes $K$, and in case $K$ fails, it uniformly selects a $q(|x|)$-bit-long string and outputs it if and only if it is a valid solution for $x$. Let $p(x, y, r)$ be as in Definitions 4.7.2 and 4.7.3, and let $s(x, y, r) \geq p(x, y, r) - \kappa(|x|)$ denote the success probability of $K^{P_{x,y,r}}(x)$. Then the new knowledge extractor succeeds with probability at least

$$s'(x, y, r) \stackrel{\text{def}}{=} s(x, y, r) + (1 - s(x, y, r)) \cdot 2^{-q(|x|)}$$

The reader can easily verify that $s'(x, y, r) \geq p(x, y, r)/2$ (by separately considering the cases $p(x, y, r) \geq 2 \cdot \kappa(|x|)$ and $p(x, y, r) \leq 2 \cdot \kappa(|x|)$), and the proposition follows. ■

### 4.7.3. Zero-Knowledge Proofs of Knowledge for $\mathcal{NP}$

The zero-knowledge proof systems for Graph Isomorphism (i.e., Construction 4.3.8) and for Graph 3-Coloring (i.e., Construction 4.4.7) are in fact proofs of knowledge (with some knowledge error) for the corresponding languages. Specifically, Construction 4.3.8 is a proof of knowledge of an isomorphism with knowledge error $\frac{1}{2}$, whereas

Construction 4.4.7 (when applied on common input $G = (V, E)$) is a proof of knowledge of a 3-coloring with knowledge error $1 - \frac{1}{|E|}$; see Exercise 26. By iterating each construction sufficiently many times, we can get the knowledge error to be exponentially small (see Proposition 4.7.5). In fact, using Proposition 4.7.6, we get proofs of knowledge with zero error. In particular, we have the following:

**Theorem 4.7.7:** *Assuming the existence of (non-uniformly) one-way functions, every $\mathcal{NP}$-relation has a zero-knowledge system for proofs of knowledge. Furthermore, inputs not in the corresponding language are accepted by the verifier with exponentially vanishing probability.*

### 4.7.4. Applications

We briefly review some of the applications of (zero-knowledge) proofs of knowledge. Typically, zero-knowledge proofs of knowledge are used for "mutual disclosure" of the same information. Suppose that `Alice` and `Bob` both claim that they know something (e.g., a 3-coloring of a common-input graph), but each is doubtful of the other's claim. Employing a zero-knowledge proof of knowledge in both directions is indeed a (conceptually) simple solution to the problem of convincing each other of their knowledge.

Before describing the applications, let us briefly comment on how their security is proved. Typically, a zero-knowledge proof of knowledge is used as a sub-protocol, and rejecting in this sub-protocol means that the verifying party detects cheating. The proof of security for the high-level protocol is by a simulation argument that utilizes the knowledge extractor, but invokes it only in case the verifying party does not detect cheating. Our definition of (the validity condition of) proofs of knowledge guarantees that the simulation will run in expected polynomial time, regardless of the (a priori unknown) probability that the verifying party will accept.

In all applications, the proof of knowledge employed has negligible soundness error (i.e., inputs not in the corresponding language are accepted by the verifier with negligible probability).

#### 4.7.4.1. Non-Oblivious Commitment Schemes

When using a commitment scheme, the receiver is guaranteed that after the commit phase the sender is committed to at most one value (in the sense that it can later "reveal" only this value). Yet the receiver is not guaranteed that the sender "knows" to what value the sender is committed. Such a guarantee can be useful in many settings and can be obtained by using a proof of knowledge. For more details, see Section 4.9.2.

#### 4.7.4.2. Protecting against Chosen Message Attacks

An obvious way of protecting against chosen message attacks on a (public-key) encryption scheme is to augment the ciphertext by a zero-knowledge proof of knowledge of the cleartext. Thus, the benefit (to the adversary) of a chosen message attack is essentially eliminated. However, one should note that the resulting encryption scheme employs bidirectional communication between the sender and the receiver (of the

encrypted message). (Definitions and alternative constructions of encryption schemes secure against chosen message attacks will be presented in Chapter 5 of Volume 2.)

#### 4.7.4.3. A Zero-Knowledge Proof System for *GNI*

The interactive proof of Graph Non-Isomorphism ($GNI$) presented in Construction 4.2.8 is not zero-knowledge (unless $GNI \in \mathcal{BPP}$). A cheating verifier can construct an arbitrary graph $H$ and learn whether or not $H$ is isomorphic to the first input graph by sending $H$ as a query to the prover. There is an even more appealing refutation of the claim that Construction 4.2.8 is auxiliary-input zero-knowledge (e.g., the verifier can check whether or not its auxiliary input is isomorphic to one of the common-input graphs). We observe, however, that Construction 4.2.8 "would have been zero-knowledge" if the verifier had always known the answers to its queries (as is the case for an honest verifier). Thus, we can modify Construction 4.2.8 to obtain a zero-knowledge proof for $GNI$ by having the verifier prove to the prover that he (i.e., the verifier) knows the answer to his query graph (i.e., that he knows an isomorphism to the appropriate input graph), and the prover answers the query only if she is convinced of this claim. Certainly, the verifier's proof of knowledge should *not* yield the answer (otherwise the prover could use that information in order to cheat, thus foiling the soundness requirement). If the verifier's proof of knowledge is perfect zero-knowledge, then certainly it does not yield the answer. In fact, it suffices that the verifier's proof of knowledge is *witness-independent* (as defined in Section 4.6).

### 4.7.5. Proofs of Identity (Identification Schemes)

Identification schemes are useful in large distributed systems in which the users are not acquainted with one another. In such distributed systems, one wishes to allow users to authenticate themselves to other users. This goal is achieved by identification schemes, defined next. In the sequel, we shall also see that identification schemes are intimately related to proofs of knowledge. We hint that a person's identity can be linked to his ability to do something and in particular to his ability to prove knowledge of some sort.

#### 4.7.5.1. Definition

Loosely speaking, an identification scheme consists of a *public file* containing *records* for each user and an *identification protocol*. Each (public) record consists of the name (or identity) of a user and auxiliary *identification information* to be used when invoking the identification protocol (as discussed later). The public file is established and maintained by a trusted party that vouches for the authenticity of the records (i.e., that each record has been submitted by the user whose name is specified in it). All users can read the public file at all times. Alternatively, the trusted party can supply each user with a signed copy of its public record. Suppose, now, that `Alice` wishes to prove to `Bob` that it is indeed she who is communicating with him. To this end, `Alice` invokes the identification protocol, with the (public-file) record corresponding to her name as a parameter. `Bob` verifies that the parameter in use indeed matches `Alice`'s public record

and proceeds by executing his role in the protocol. It is required that `Alice` always be able to convince `Bob` (that she is indeed `Alice`), whereas nobody else can fool `Bob` into believing that she/he is `Alice`. Furthermore, `Carol` should not be able to impersonate `Alice` even after receiving polynomially many proofs of identity from `Alice`.

The identification information is generated by `Alice` using a randomized algorithm. Clearly, if the identification information is to be of any use, then `Alice` must keep secret the random coins she used to generate her record. Furthermore, `Alice` must use these stored coins during the execution of the identification protocol, but this must be done in a way that will not allow anyone else to impersonate her later.

**Conventions.** In the following definition, we adopt the formalism and notations of interactive machines with auxiliary input (presented in Definition 4.2.10). We recall that when $M$ is an interactive machine, we denote by $M(y)$ the machine that results by fixing $y$ to be the auxiliary input of machine $M$. In the following definition, $n$ is the security parameter, and we assume with little loss of generality that the names (i.e., identities) of the users are encoded by strings of length $n$. If $A$ is a probabilistic algorithm and $x, r \in \{0,1\}^*$, then $A_r(x)$ denotes the output of algorithm $A$ on input $x$ and random coins $r$.

**Motivation.** Algorithm $I$ in the following definition corresponds to the procedure used to generate identification information, and $(P, V)$ corresponds to the identification protocol itself. The interactive machines $B'$ and $B''$ represent two components of the adversary behavior (i.e., interacting with the user in order to extract its secrets and later trying to impersonate it). On a first reading, the reader can ignore algorithm $B'$ and the random variable $T_n$ (in the security condition). Doing so, however, yields a weaker condition that typically is unsatisfactory.

**Definition 4.7.8 (Identification Scheme):** *An* **identification scheme** *consists of a pair $(I, \Pi)$, where $I$ is a probabilistic polynomial-time algorithm and $\Pi = (P, V)$ is a pair of probabilistic polynomial-time interactive machines satisfying the following conditions:*

- Viability: *For every $n \in \mathbb{N}$, every $\alpha \in \{0,1\}^n$, and every $s \in \{0,1\}^{\mathrm{poly}(n)}$,*

$$\Pr[\langle P(s), V\rangle(\alpha, I_s(\alpha)) = 1] = 1$$

- Security: *For every pair of probabilistic polynomial-time interactive machines $B'$ and $B''$, every polynomial $p(\cdot)$, all sufficiently large $n \in \mathbb{N}$, every $\alpha \in \{0,1\}^n$, and every $z$,*

$$\Pr\left[\langle B''(z, T_n), V\rangle\left(\alpha, I_{S_n}(\alpha)\right) = 1\right] < \frac{1}{p(n)}$$

  *where $S_n$ is a random variable uniformly distributed over $\{0,1\}^{\mathrm{poly}(n)}$ and $T_n$ is a random variable describing the output of $B'(z)$ after interacting with $P(S_n)$, on common input $(\alpha, I_{S_n}(\alpha))$, for polynomially many times.*

*Algorithm $I$ is called the* **information-generating algorithm**, *and the pair $(P, V)$ is called the* **identification protocol**.

Hence, to use the identification scheme, a user, say `Alice`, whose identity is encoded by the string $\alpha$, should first uniformly select a secret string $s$, compute $i \stackrel{\text{def}}{=} I_s(\alpha)$, ask the trusted third party to place the record $(\alpha, i)$ in the public file, and store the string $s$ in a safe place. The viability condition asserts that `Alice` can convince `Bob` of her identity by executing the identification protocol: `Alice` invokes the program $P$ using the stored string $s$ as auxiliary input, and `Bob` uses the program $V$ and makes sure that the common input is the public record containing $\alpha$ (which is in the public file). Ignoring for a moment the algorithm $B'$ and the random variable $T_n$, the security condition implies that it is infeasible for a party to impersonate `Alice` if all that this party has is the public record of `Alice` and some unrelated auxiliary information (represented by the auxiliary input $z$). However, such a security condition may not suffice in many applications, since a user wishing to impersonate `Alice` may ask her first to prove her identity to him/her. The (full) security condition asserts that even if `Alice` has proved her identity to `Carol` many times in the past, still it is infeasible for `Carol` to impersonate `Alice`. We stress that `Carol` cannot impersonate `Alice` to `Bob` *provided that she cannot interact concurrently with both* `Alice` *and* `Bob`. In case this condition does not hold, nothing is guaranteed (and indeed `Carol` can easily impersonate `Alice` by referring `Bob`'s questions to `Alice` and answering as `Alice` does).

### 4.7.5.2. Identification Schemes and Proofs of Knowledge

A natural way to determine a person's identity is to ask him/her to supply a proof of knowledge of a fact that the person is supposed to know. Let us consider a specific (but in fact quite generic) example.

**Construction 4.7.9 (Identification Scheme Based on a One-Way Function):** *Let $f$ be a function. On input an identity $\alpha \in \{0,1\}^n$, the* information-generating algorithm *uniformly selects a string $s \in \{0,1\}^n$ and outputs $f(s)$.* (The pair $(\alpha, f(s))$ is the public record for the user named $\alpha$.) *The* identification protocol *consists of a proof of knowledge of the inverse of the second element in the public record. Namely, in order to prove its identity, user $\alpha$ proves that it knows a string $s$ such that $f(s) = r$, where $(\alpha, r)$ is a record in the public file. (The proof of knowledge in use is allowed to have negligible knowledge error.)*

**Proposition 4.7.10:** *If $f$ is a one-way function and the proof of knowledge in use is zero-knowledge, then Construction 4.7.9 constitutes an identification scheme.*

Hence, identification schemes exist if one-way functions exist. Practical identification schemes can be constructed based on specific intractability assumptions. For example, assuming the intractability of factoring, the so-called Fiat-Shamir identification scheme, which is actually a proof of knowledge of a modular square root, follows.

**Construction 4.7.11 (The Fiat-Shamir Identification Scheme, Basic Version):** *On input an identity $\alpha \in \{0,1\}^n$, the* information-generating algorithm *uniformly selects a composite number $N$ that is the product of two $n$-bit-long primes and a residue $s \bmod N$, and it outputs the pair $(N, s^2 \bmod N)$.* (The pair $(\alpha, (N, s^2$

mod $N$)) is the public record for user $\alpha$.) *The* identification protocol *consists of a proof of knowledge of the corresponding modular square root. Namely, in order to prove its identity, user $\alpha$ proves that it knows a modular square root of $r \stackrel{\text{def}}{=} s^2 \bmod N$, where $(\alpha, (r, N))$ is a record in the public file. (Again, negligible knowledge error is allowed.)*

The proof of knowledge of modular square roots is analogous to the proof system for Graph Isomorphism presented in Construction 4.3.8. Namely, in order to prove knowledge of a square root of $r \equiv s^2 \pmod N$, the prover repeats the following steps sufficiently many times:

**Construction 4.7.12 (Atomic Proof of Knowledge of Modular Square Root):** *This refers to the common input $(r, N)$, where the prescribed prover has auxiliary input $s$ such that $r \equiv s^2 \pmod N$:*

- *The prover randomly selects a residue $g$* modulo $N$ *and sends $h \stackrel{\text{def}}{=} g^2 \bmod N$ to the verifier.*
- *The verifier uniformly selects $\sigma \in \{0, 1\}$ and sends it to the prover.*
- Motivation: *In case $\sigma = 0$, the verifier asks for a square root of $h$* mod $N$*, whereas in case $\sigma = 1$ the verifier asks for a square root of $h \cdot r \bmod N$. In the sequel, we assume, without loss of generality, that $\sigma \in \{0, 1\}$.*
- *The prover replies with $a \stackrel{\text{def}}{=} g \cdot s^\sigma \bmod N$.*
- *The verifier accepts if and only if the messages $h$ and $a$ sent by the prover satisfy $a^2 \equiv h \cdot r^\sigma \bmod N$.*

When Construction 4.7.12 is repeated $k$ times, either sequentially or in parallel, the resulting protocol constitutes a proof of knowledge of a modular square root, with knowledge error $2^{-k}$ (see Exercise 27). In case these repetitions are conducted sequentially, the resulting protocol is zero-knowledge. Yet, for use in Construction 4.7.11, it suffices that the proof of knowledge be *witness-hiding* under the relevant distribution (see Definition 4.6.5), even when polynomially many executions take place concurrently (in an asynchronous manner). Hence the resulting identification scheme has constant-round complexity. We remark that for identification purposes it suffices to perform Construction 4.7.12 super-logarithmically many times. Furthermore, fewer repetitions can also be of value: When applying Construction 4.7.12 for $k = O(\log n)$ times and using the resulting protocol in Construction 4.7.11, we get a scheme (for identification) in which impersonation can occur with probability at most $2^{-k}$.

### 4.7.5.3. Identification Schemes and Proofs of Ability

As hinted earlier, a proof of knowledge of a string (i.e., the ability to output the string) is a special case of a proof of ability to do something. It turns out that identification schemes can also be based on the more general concept of proofs of ability. We avoid defining this concept and confine ourself to two "natural" examples of using a proof of ability as a basis for identification.

It is everyday practice to identify people by their ability to produce their signatures. This practice can be carried into the digital setting. Specifically, the public record of

`Alice` consists of her name and the verification key corresponding to her secret signing key in a predetermined signature scheme. The identification protocol consists of `Alice` signing a random message chosen by the verifier.

A second popular means of identification consists of identifying people by their ability to answer personal questions correctly. A digital analogue of this common practice follows. We use pseudorandom functions (see Section 3.6) and zero-knowledge proofs (of membership in a language). The public record of `Alice` consists of her name and a "commitment" to a randomly selected pseudorandom function (e.g., either via a string-commitment to the index of the function or via a pair consisting of a random domain element and the value of the function at that point). The identification protocol consists of `Alice` returning the value of the function at a random location chosen by the verifier and supplying a zero-knowledge proof that the value returned indeed matches the function appearing in the public record. We remark that the digital implementation offers more security than the everyday practice. In the everyday setting, the verifier is given the list of all possible question-and-answer pairs and is trusted not to try to impersonate the user. Here we have replaced the possession of the correct answers with a zero-knowledge proof that the answer is correct.

### 4.7.6. Strong Proofs of Knowledge

Definitions 4.7.2 and 4.7.3 rely in a fundamental way on the notion of *expected* running time. Specifically, these definitions refer to the *expected* running time of the knowledge extractor. For reasons discussed in Section 4.3.1.6, we prefer to avoid the notion of *expected* running time whenever possible. Thus, we consider next a more stringent definition in which the knowledge extractor is required to run in *strict* polynomial time, rather than in *expected* time inversely proportional to the acceptance probability (as in Definition 4.7.2). (We also take the opportunity to postulate, in the definition, that instances not in $L_R$ are accepted with negligible probability; this is done by extending the scope of the validity condition also to $x$'s not in $L_R$.)

#### 4.7.6.1. Definition

**Definition 4.7.13 (System of Strong Proofs of Knowledge):** *Let $R$ be a binary relation. We say that an interactive function $V$ is a* **strong knowledge verifier for the relation** *$R$ if the following two conditions hold:*

- Non-triviality: *As in Definition 4.7.2.*
- Strong validity: *There exists a negligible function $\mu : \mathbb{N} \to [0, 1]$ and a probabilistic* (strict) polynomial-time *oracle machine $K$ such that for every interactive function $P$ and every $x, y, r \in \{0,1\}^*$, machine $K$ satisfies the following condition:*

  > *Let $p(x, y, r)$ and $P_{x,y,r}$ be as in Definition 4.7.2. If $p(x, y, r) > \mu(|x|)$, then on input $x$ and access to oracle $P_{x,y,r}$, machine $K$ outputs a solution $s \in R(x)$ with probability at least $1 - \mu(|x|)$.*

*The oracle machine $K$ is called a* **strong knowledge extractor**.

*An interactive pair $(P, V)$ such that $V$ is a strong knowledge verifier for a relation $R$, and $P$ is a machine satisfying the non-triviality condition (with respect to $V$ and $R$), is called a* **system for strong proofs of knowledge** *for the relation $R$.*

Our choice of using $\mu$ (rather than a different negligible function $\mu'$) as an upper bound on the failure probability of the extractor (in the strong validity requirement) is immaterial. Furthermore, for $\mathcal{NP}$-relations, requiring the existence of an extractor that succeeds with noticeable probability is equivalent to requiring the existence of an extractor that fails with exponentially vanishing probability. (That is, in the case of $\mathcal{NP}$-relations, the failure probability can be decreased by successive applications of the extractor.) This *strong validity* requirement is stronger than the validity (with error $\mu$) requirement of Definition 4.7.2, in two ways:

1. The extractor in Definition 4.7.13 runs in (strict) polynomial time, regardless of the value of $p(x, y, r)$, whereas the extractor in Definition 4.7.2 runs in expected time $\text{poly}(n)/(p(x, y, r) - \mu(|x|))$. Note, however, that the extractor in Definition 4.7.13 is allowed to fail with probability at most $\mu(|x|)$, whereas the extractor in Definition 4.7.2 can never fail.
2. The strong validity requirement implies that $x \notin L_R$ is accepted by the verifier with probability at most $\mu(|x|)$, whereas this is not required in Definition 4.7.2. This soundness condition is natural in the context of the current definition that, unlike Definition 4.7.2, always allows for non-zero (but negligible) error probability.

### 4.7.6.2. An Example: Strong (ZK) Proof of Knowledge of Isomorphism

Sequentially repeating the (zero-knowledge) proof systems for Graph Isomorphism (i.e., Construction 4.3.8) sufficiently many times yields a strong proof of knowledge of isomorphism. The key observation is that each application of the basic proof system (i.e., Construction 4.3.8) results in one of two possible situations, depending on whether the verifier asks to see an isomorphism to the first or second graph. If the prover answers correctly in both cases, then we can retrieve an isomorphism between the input graphs (by composing the isomorphisms provided in the two cases). If the prover fails in both cases, then the verifier will reject regardless of what the prover does from that point on. Specifically, the preceding discussion suggests the following construction of a strong knowledge extractor (where we refer to repeating the basic proof systems $n$ times and set $\mu(n) = 2^{-n}$).

**Strong Knowledge Extractor for Graph Isomorphism.** On input $(G_1, G_2)$ and access to the prover-strategy oracle $P^*$, we proceed in $n$ iterations, starting with $i = 1$. Initially, $T$ (the transcript thus far) is empty.

1. Obtain the intermediate graph $G'$ from the prover strategy (i.e., $G' = P^*(T)$).
2. Extract the prover's answers to both possible verifier moves. That is, for $j = 1, 2$, let $\psi_j \leftarrow P^*(T, j)$. We say that $\psi_j$ is *correct* if it is an isomorphism between $G_j$ and $G'$.

3. If both $\psi_j$'s are correct, then $\phi \leftarrow \psi_2^{-1}\psi_1$ is an isomorphism between $G_1$ and $G_2$. In this case we output $\phi$ and halt.
4. In case $\psi_j$ is correct for a single $j$, and $i < n$, we let $T \leftarrow (T, j)$ and proceed to the next iteration (i.e., $i \leftarrow i + 1$). Otherwise, we halt, with no output.

It can be easily verified that if this extractor halts with no output in any iteration $i < n$, then the verifier (in the real interaction) accepts with probability zero. Similarly, if the extractor halts with no output in iteration $n$, then the verifier (in the real interaction) accepts with probability at most $2^{-n}$. Thus, whenever $p((G_1, G_2), \cdot, \cdot) > 2^{-n}$, the extractor succeeds in recovering an isomorphism between the two input graphs.

### 4.7.6.3. Strong (ZK) Proofs of Knowledge for $\mathcal{NP}$-Relations

A similar argument can be applied to some zero-knowledge proof systems for $\mathcal{NP}$. In particular, consider $n$ sequential repetitions of the following basic (zero-knowledge) proof system for the *Hamiltonian-cycle* (HC) problem. We consider directed graphs (and the existence of directed Hamiltonian cycles).

**Construction 4.7.14 (Basic Proof System for HC):**

- Common input: *a directed graph* $G = (V, E)$*, with* $n \stackrel{\text{def}}{=} |V|$.
- Auxiliary input to prover: *a directed Hamiltonian cycle,* $C \subset E$*, in* $G$.
- Prover's first step (P1): *The prover selects a random permutation* $\pi$ *of the vertices* $V$ *and commits to the entries of the adjacency matrix of the resulting permuted graph. That is, it sends an* $n$*-by-*$n$ *matrix of commitments such that the* $(\pi(i), \pi(j))$ *entry is a commitment to* 1 *if* $(i, j) \in E$ *and is a commitment to* 0 *otherwise.*
- Verifier's first step (V1): *The verifier uniformly selects* $\sigma \in \{0, 1\}$ *and sends it to the prover.*
- Motivation: $\sigma = 0$ *means that the verifier asks to check that the matrix of commitments is a legitimate one, whereas* $\sigma = 1$ *means that the verifier asks to reveal a Hamiltonian cycle in the permuted graph.*
- Prover's second step (P2): *If* $\sigma = 0$*, then the prover sends* $\pi$ *to the verifier along with the revealing (i.e., pre-images) of all commitments. Otherwise, the prover reveals to the verifier only the commitments to entries* $(\pi(i), \pi(j))$*, with* $(i, j) \in C$.
- Verifier's second step (V2): *If* $\sigma = 0$*, then the verifier checks that the revealed graph is indeed isomorphic, via* $\pi$*, to* $G$*. Otherwise, the verifier simply checks that all revealed values are* 1 *and that the corresponding entries form a simple* $n$*-cycle. (Of course, in both cases, the verifier checks that the revealed values do fit the commitments.) The verifier accepts if and only if the corresponding condition holds.*

We claim that the protocol resulting from sequentially repeating Construction 4.7.14 $n$ times is a (zero-knowledge) strong proof of knowledge of a Hamiltonian cycle; see Exercises 20 and 30. Because a Hamiltonian cycle is $\mathcal{NP}$-complete, we get such proof systems for any language in $\mathcal{NP}$. We mention that the known zero-knowledge strong proofs of knowledge for $\mathcal{NP}$-complete languages are all costly in terms of the round-complexity. Still, we have the following:

**Theorem 4.7.15:** *Assuming the existence of (non-uniformly) one-way functions, every $\mathcal{NP}$-relation has a zero-knowledge system for* strong *proofs of knowledge.*

## 4.8.* Computationally Sound Proofs (Arguments)

In this section we consider a relaxation of the notion of an interactive proof system. Specifically, we relax the soundness condition of interactive proof systems. Instead of requiring that it be *impossible* to fool the verifier into accepting false statements (with probability greater than some bound), we require only that it be *infeasible* to do so. We call such protocols *computationally sound proof systems* (or *arguments*). The advantage of computationally sound proof systems is that *perfect* zero-knowledge *computationally sound* proof systems can be constructed, under some reasonable complexity-assumptions, for all languages in $\mathcal{NP}$. Recall that *perfect* zero-knowledge proof systems are unlikely to exist for all languages in $\mathcal{NP}$ (see Section 4.5). Also recall that *computational* zero-knowledge proof systems do exist for all languages in $\mathcal{NP}$, provided that one-way functions exist. Hence, the previously quoted positive results exhibit some kind of a trade-off between the soundness and zero-knowledge properties of the zero-knowledge protocols of $\mathcal{NP}$. (We remark, however, that the perfect zero-knowledge computationally sound proofs for $\mathcal{NP}$ are constructed under stronger complexity-theoretic assumptions than are the ones used for the computational zero-knowledge proofs. It is indeed an interesting research project to try to construct perfect zero-knowledge computationally sound proofs for $\mathcal{NP}$ under weaker assumptions, in particular, assuming only the existence of one-way functions.)

We mention that it seems that computationally sound proof systems can be much more efficient than ordinary proof systems. Specifically, under some plausible complexity assumptions, extremely efficient computationally sound proof systems (i.e., requiring only poly-logarithmic communication and randomness) exist for any language in $\mathcal{NP}$. An analogous result cannot hold for ordinary proof systems unless $\mathcal{NP}$ is contained in deterministic quasi-polynomial time (i.e., $\mathcal{NP} \subseteq \mathrm{Dtime}(2^{\mathrm{polylog}})$).

### 4.8.1. Definition

The definition of computationally sound proof systems follows naturally from the foregoing discussion. The only issue to consider is that *merely* replacing the soundness condition of Definition 4.2.4 with a *computational-soundness* condition leads to an unnatural definition, since the computational power of the prover in the completeness condition (in Definition 4.2.4) is not restricted. Hence, it is natural to restrict the prover in *both* (the completeness and soundness) conditions to be an efficient one. It is crucial to interpret "efficient" as being probabilistic polynomial-time *given auxiliary input* (otherwise, only languages in $\mathcal{BPP}$ will have such proof systems). Hence, our starting point is Definition 4.2.10 (rather than Definition 4.2.4).

**Definition 4.8.1 (Computationally Sound Proof System (Arguments)):** *A pair of interactive machines $(P, V)$ is called a* **computationally sound proof system**

*(or an argument)* **for a language** $L$ *if both machines are polynomial-time* (with auxiliary inputs) *and the following two conditions hold:*

- Completeness: *For every* $x \in L$*, there exists a string* $y$ *such that for every string* $z$*,*

$$\Pr[\langle P(y), V(z)\rangle(x) = 1] \geq \frac{2}{3}$$

- Computational soundness: *For every polynomial-time interactive machine* $B$*, and for all sufficiently long* $x \notin L$ *and every* $y$ *and* $z$*,*

$$\Pr[\langle B(y), V(z)\rangle(x) = 1] \leq \frac{1}{3}$$

As usual, the error probability (in both the completeness and soundness conditions) can be reduced (from $\frac{1}{3}$) down to as much as $2^{-\mathrm{poly}(|x|)}$ by sequentially repeating the protocol sufficiently many times; see Exercise 31. We mention that *parallel* repetitions may fail to reduce the (computational) soundness error in some cases.

### 4.8.2. Perfectly Hiding Commitment Schemes

The thrust of the current section is toward a method for constructing *perfect* zero-knowledge *arguments* for every language in $\mathcal{NP}$. This method makes essential use of the concept of a commitment scheme with a *perfect* (or "information-theoretic") secrecy property. Hence, we start with an exposition of such perfectly hiding commitment schemes. We remark that such schemes may also be useful in other settings (e.g., other settings in which the receiver of the commitment is computationally unbounded; see, for example, Section 4.9.1).

The difference between commitment schemes (as defined in Section 4.4.1) and perfectly hiding commitment schemes (defined later) consists in a switch in the scope of the secrecy and unambiguity requirements: In commitment schemes (see Definition 4.4.1), the secrecy requirement is computational (i.e., refers only to probabilistic polynomial-time adversaries), whereas the unambiguity requirement is information-theoretic (and makes no reference to the computational power of the adversary). On the other hand, in perfectly hiding commitment schemes (as defined later), the secrecy requirement is information-theoretic, whereas the unambiguity requirement is computational (i.e., refers only to probabilistic polynomial-time adversaries).

**Comments about Terminology.** From this point on, we explicitly mention the "perfect" feature of a commitment scheme to which we refer. That is, a commitment scheme as in Definition 4.4.1 will be referred to as *perfectly binding*, whereas a commitment scheme as in Definition 4.8.2 (presented later) will be referred to as *perfectly hiding*. Consequently, when we talk of a commitment scheme without specifying any "perfect" feature, it may be that the scheme is only computationally hiding and computationally binding. We remark that it is impossible to have a commitment scheme that is both perfectly hiding and perfectly binding (see Exercise 32).

We stress that the terminology just suggested is inconsistent with the exposition in Section 4.4 (in which schemes such as in Definition 4.4.1 were referred to as "commitment schemes," without the extra qualification of "perfectly binding").[20] Furthermore, the terminology just suggested is inconsistent with significant parts of the literature, in which a variety of terms can be found.[21]

### 4.8.2.1. Definition

Loosely speaking, a perfectly hiding commitment scheme is an efficient *two-phase* two-party protocol through which the *sender* can commit itself to a *value* such that the following two conflicting requirements are satisfied:

1. (*Perfect*) *secrecy* (or *hiding*): At the end of the *commit phase*, the *receiver* does not gain any *information* about the sender's value.
2. *Unambiguity* (or *binding*): It is infeasible for the sender to interact with the receiver, so the commit phase is successfully terminated, and yet later it is feasible for the sender to perform the *reveal phase* in two different ways, leading the receiver to accept (as legal "openings") two different values.

Using conventions analogous to those in Section 4.4.1, we state the following definition. Again, $S$ and $R$ are the specified strategies of the commitment's sender and receiver, respectively.

**Definition 4.8.2 (Perfectly Hiding Bit-Commitment Scheme):** *A* **perfectly hiding bit-commitment scheme** *is a pair of probabilistic polynomial-time interactive machines, denoted* $(S, R)$*, satisfying the following:*

- Input specification: *The common input is an integer n presented in unary (serving as the security parameter). The private input to the sender is a bit denoted v.*
- Secrecy (hiding): *For every probabilistic (not necessarily polynomial-time) machine* $R^*$ *interacting with S, the random variables describing the output of* $R^*$ *in the two cases, namely* $\langle S(0), R^*\rangle(1^n)$ *and* $\langle S(1), R^*\rangle(1^n)$*, are identically distributed.*
- Unambiguity (binding): *Preliminaries: For simplicity,* $v \in \{0,1\}$ *and* $n \in \mathbb{N}$ *are implicit in all notations. Fix any probabilistic polynomial-time algorithm* $F^*$ *and any polynomial* $p(\cdot)$*.*

  1. *As in Definition 4.4.1, a* receiver's view of an interaction *with the sender, denoted* $(r, \overline{m})$*, consists of the random coins used by the receiver (i.e., r) and the sequence of messages received from the sender (i.e.,* $\overline{m}$*). A* sender's view *of the same interaction, denoted* $(s, \tilde{m})$*, consists of the random coins used by the sender (i.e., s) and the sequence of messages received from the receiver (i.e.,* $\tilde{m}$*). A* joint view *of the interaction is a pair consisting of corresponding receiver and sender views of the same interaction.*

[20] The extra qualification was omitted from the terminology of Section 4.4 in order to simplify the basic text.

[21] For example, as in Section 4.4, many works refer to schemes such as in Definition 4.4.1 merely by the term "commitment schemes," and many refer to schemes such as in Definition 4.8.2 by the term "perfect commitment schemes." Furthermore, in some works the term "commitment schemes" means schemes such as in Definition 4.8.2.