Chapter 4

Zero-Knowledge Proof Systems

In this chapter we discuss zero-knowledge proof systems. Loosely speaking, such proof systems have the remarkable property of being convincing and yielding nothing (beyond the validity of the assertion). The main result presented is a method to generate 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. In addition, we discuss more refined aspects of the concept of zero-knowledge and their affect 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), then we define and exemplify the notions of interactive proofs (Section 4.2) and of zero-knowledge (Section 4.3), and finally we present a zero-knowledge proof systems for every language in $\mathcal{NP}$ (Section 4.4). Sections dedicated to advanced topics follow. Unless stated differently, 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 relaxion of zero-knowledge and prove that it is closed under parallel composition (which is not the case, in general, for zero-knowledge).
- In Section 4.7 we define and discuss (zero-knowledge) proofs of knowledge.
- In Section 4.8 we discuss a relaxion 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. Subsection 4.8.2 is a prerequisite for the first construction; whereas Sections 4.8, 4.7, and 4.6 constitute a prerequisite for the second.

- In Section 4.10 we discuss non-interactive zero-knowledge proofs. A basic definition in Section 4.6 is a prerequisite for the last result in Section 4.10.
- In Section 4.11 we discuss multi-prover proof systems.

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

4.1 Zero-Knowledge Proofs: Motivation

An archetypical "cryptographic" problem consists of providing mutually distrustful parties with a means of "exchanging" (predetermined) "pieces of information". The setting consists of several parties, each wishing to obtain some predetermined partial information concerning the secrets of the other parties. Yet each party wishes to reveal as little information as possible about its own secret. To clarify the issue, let us consider a specific example.

> Suppose that all users in a system keep backups of their entire file system, encrypted using their public-key encryption, in a publicly accessible storage media. Suppose that at some point, one user, called `Alice`, wishes to reveal to another user, called `Bob`, the cleartext of one of her files (which appears in one of her backups). A trivial "solution" is for `Alice` just to send the (cleartext) file to `Bob`. The problem with this "solution" is that `Bob` has no way of verifying that `Alice` really sent him a file from her public backup, rather than just sending him an arbitrary file. `Alice` can simply prove that she sends the correct file by revealing to `Bob` her private encryption key. However, doing so, will reveal to `Bob` the contents of all her files, which is certainly something that `Alice` does not want to happen. The question is whether `Alice` can convince `Bob` that she indeed revealed the correct file without yielding any additional "knowledge".
>
> An analogous question 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, only has $f(x)$. 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 above, B will have no way of verifying whether A has really sent the correct bit (and not its complement). Party A can indeed prove that it sends the correct bit (i.e., $b(x)$) by sending x as well, but revealing x to B is much more than what A had originally in mind. Again, the question is whether A can convince B that it indeed revealed the correct bit (i.e., $b(x)$) without yielding any additional "knowledge".

In general, the question is whether *it is possible to prove a statement without yielding anything beyond its validity*. Such proofs, whenever they exist, are called *zero-knowledge*, and play a central role (as we shall see in the subsequent chapter) 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

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

4.1.1.1 A Proof as a fixed sequence or as an interactive process

Traditionally in mathematics, a "proof" is a *fixed* sequence consisting of statements which are either self-evident or are derived from previous statements via self-evident rules. Actually, it is more accurate to substitute the phrase "self-evident" by 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 mathematics proofs:

1. proofs are viewed as fixed objects;
2. proofs are considered at least as fundamental as their consequence (i.e., the theorem).

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, the cross-examination of a witness in court is considered a proof in law, and failure to answer a rival's claim is considered a proof in philosophical, political and sometimes even technical discussions. In addition, in real-life situations, proofs are considered secondary (in importance) to their consequence.

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). The dynamic interpretation of the notion of a proof is more adequate to our setting in which proofs are used as tools (i.e., subprotocols) inside "cryptographic" protocols. Furthermore, the 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. Instead, the emphasis is placed on the *verification process*, or in other words on (the role of) the verifier. Both in mathematics and in real-life situations, proofs are defined in terms of the verification procedure. Typically, 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 and the theorem-proving tasks 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 so 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. Hence, any y satisfying $(x, y) \in R_L$ is considered a *proof* of membership of $x \in L$. Hence, correct statements (i.e., $x \in L$) and only them 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".

It is worthwhile to stress the distrustful attitude towards the prover in 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 skeptic of anything the prover says.

4.1.1.3 Completeness and Validity

Two fundamental properties of a proof system (i.e., a verification procedure) are its *validity* and *completeness*. The validity property asserts that the verification procedure cannot be "tricked" into accepting false statements. In other words, *validity* captures the verifier ability of protecting itself from being convinced of false statements (no matter what the prover does in order to fool it). 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 these statements can be proven (while no false statement can be "proven"). This fundamental fact is given a precise meaning in results such as Gödel's Incompleteness Theorem and Turing's proof of the unsolvability of the Halting Problem. We stress that in this chapter we confine ourself to the class of sets 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 is knowledge and what is a gain of knowledge. 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 "conservative" 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 since he knows this fact himself. If, on the other hand, `Alice` tells `Bob` a proof of Fermat's Theorem then certainly he gained knowledge from the conversation.

To give a better flavour 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 the graph is Eulerian or not. Clearly, we say that `Bob` gains no knowledge from `Alice`'s answer, since he could have determined the answer easily by himself (e.g., by using Euler's Theorem which asserts that a graph is Eulerian if and only if all its vertices have even degree). On the other hand, if `Bob` asks `Alice` whether the graph is Hamiltonian or not, and `Alice` (somehow) answers this question then we cannot say that `Bob` gained no knowledge (since we do not know of an efficient procedure by which `Bob` can determine the answer by himself, and assuming $\mathcal{P} \neq \mathcal{NP}$ no such efficient procedure exists). Hence, we say that `Bob` *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` *gained no knowledge* from the interaction. Hence, `Bob` gains knowledge only if he receives the result of a computation which is infeasible for `Bob`. The question of how could `Alice` conduct this infeasible computation (e.g., answer `Bob`'s question of whether the graph is Hamiltonian) has been ignored so far. Jumping ahead, we remark that `Alice` may be a mere abstraction or may be in possession of additional hints, that enables to efficiently conduct computations that are otherwise infeasible (and in particular are infeasible for `Bob` who does not have these hints). (Yet, these hints are not necessarily "information" in the information theoretic sense as they may be determined by the common input, but not efficiently computed from it.)

Knowledge vs. information. We wish to stress that *knowledge* (as discussed above) is very different from *information* (in the sense of information theory).

- *Knowledge* is related to computational difficulty, whereas *information* is not. In the above examples, there was a different between the knowledge revealed in case `Alice` answers questions of the form "is the graph Eulerian" and the case she answers questions of the form "is the graph Hamilton". From an information theoretic point of view there is no difference between the two cases (i.e., in both `Bob` gets no information).

- *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`, since he can 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 towards the introduction of zero-knowledge interactive proofs. Interactive proof systems are interesting for their own sake, and have important complexity theoretic applications, that are discussed in Chapter 9.

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. 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.

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 proven. We first define the notion of an interactive machine, and next the notion of interaction between two such machines. The reader may skip to the next part of this subsection (titled "Conventions regarding interactive machines") with little loss (if at all).

Definition 4.2.1 (an interactive machine):

- *An* interactive Turing machine (ITM) *is a (deterministic) multi-tape Turing machine. The tapes consists of 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 initiated to contents 0. 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 contents 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 location of its heads on the various tapes, and the contents of the writeable tapes of the ITM is not modified.*

- *The contents of the input-tape is called* input, *the contents of the random-tape is called* random-input, *and the contents of the output-tape at termination is called* output. *The contents written on the write-only communication-tape during a (time) period in which the machine is active is called the* message sent *at this period. Likewise, the contents 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 only in one direction, say left to right).*

The above definition, taken by itself, seems quite nonintuitive. In particular, one may say that once being idle the machine never becomes active again. One may also wonder 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 so that some of their tapes coincide. Intuitively, the messages sent by an interactive machine are received by a second machine which 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 contents of the shared switch-tape and by doing so the other machine (having opposite identity) becomes active. The computation of such a pair of machines consists of the machines alternatingly sending messages to one another, based on their initial (common) input, their (distinct) random-inputs, and the messages each machine has received so 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. Each pair consists of the local configuration of each 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.*

- *If one machine halts while the switch-tape still holds its identity the we say that both machines have halted.*

At this point, the reader may object to the above definition, saying that the individual machines are deprived of individual local inputs (and observing that they are given individual and unshared random-tapes). This restriction is removed in Subsection 4.2.3, and in fact removing it 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 above simpler definition. The convention of individual random-tapes is however essential to the power of interactive proofs (see Exercise 4).

4.2.1.2 Conventions regarding interactive machines

Typically, we consider executions when the contents 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 since during a finite computation only a finite prefix is read (and matters). The contents 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.

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

Definition 4.2.3 (the complexity of an interactive machine): *We say that an interactive machine A has* time complexity $t : \mathbb{N} \mapsto \mathbb{N}$ *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 contents of its random-tape and B's random-tape) halts within $t(|x|)$ steps.*

We stress that the time complexity, so defined, is independent of the contents of the messages that machine A receives. In other word, it is an upper bound which holds for all possible incoming messages. In particular, an interactive machine with time complexity $t(\cdot)$ reads, 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 may be viewed as one entity called the verifier). A "proof" to 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 may be very complex and may consist of many message exchanges, as long as the total time spent by the verifier is polynomial.

In light of the association of efficient procedures with probabilistic polynomial-time algorithms, it is natural to consider probabilistic polynomial-time verifiers. Furthermore, the verifier's verdict of whether to accept or reject the claim is probabilistic, and a bounded error probability is allowed. (The error can of course be decreased to be negligible by repeating the verification procedure sufficiently many times.) Loosely speaking, we require that the prover can convince the verifier of the validity of valid statement, while nobody can fool the verifier into believing false statements. In fact, it is only required that the verifier accepts valid statements with "high" probability, whereas the probability that it accepts a false statement is "small" (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 $x \in L$*

$$\mathsf{Pr}\left(\langle P, V\rangle(x)=1\right) \geq \frac{2}{3}$$

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

$$\mathsf{Pr}\left(\langle B, V\rangle(x)=1\right) \leq \frac{1}{3}$$

Some remarks are in place. We first stress that the soundness condition refers to all potential "provers" whereas the completeness condition refers only to the prescribed prover P. Secondly, the verifier is required to be (probabilistic) polynomial-time, while no resource bounds are placed on the computing power of the prover (in either completeness or soundness conditions!). Thirdly, as in the case of $\mathcal{BPP}$, the error probability in the above definition can be made exponentially small by repeating the interaction (polynomially) many times (see below).

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 above), 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 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 proof system both parties are deterministic (i.e., make no use of their random-tape). 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 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 further discussion in Chapter 9).

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

By the above discussion $\mathcal{NP} \subseteq \mathcal{IP}$. Since languages in $\mathcal{BPP}$ can be viewed as 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 $\mathcal{BPP} \subseteq \mathcal{NP}$.

We stress that the definition of the class $\mathcal{IP}$ remains invariant if one replaced the (constant) bounds in the completeness and soundness conditions by two functions $\mathsf{c}, \mathsf{s} : \mathbb{N} \mapsto \mathbb{N}$ 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,

Definition 4.2.6 (generalized interactive proof): *Let $\mathsf{c}, \mathsf{s} : \mathbb{N} \mapsto \mathbb{N}$ 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 $x \in L$*

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

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

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

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

Proposition 4.2.7 *The following three conditions are equivalent*

1. $L \in \mathcal{IP}$. *Namely, there exists a 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 imply 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 IP)

All examples of interactive proof systems presented so far were degenerate (e.g., the interaction, if at all, was 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.

Two graphs, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, are called *isomorphic* if there exists a 1-1 and onto mapping, π, from the vertex set V_1 to the vertex set V_2 so that $(u, v) \in E_1$ if and only if $(\pi(v), \pi(u)) \in E_2$. The mapping π, if existing, is called an *isomorphism* between the graphs.

Construction 4.2.8 (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* $V_1 = \{1, 2, ..., |V_1|\}$, *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* π *from the set of permutations over the vertex set* V_σ. *The verifier constructs a graph with vertex set* V_σ *and edge set*

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

and sends (V_σ, 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 algorithm) 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 is 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 a $\tau \in \{1, 2\}$ so that the graph G' is isomorphic to the input graph G_τ. (If both $\tau = 1, 2$ satisfy the condition then τ is selected arbitrarily. In case no $\tau \in \{1, 2\}$ satisfies the condition, τ is set to 0). The prover sends τ to the verifier.*

- Verifier's second Step (V2): *If the message, τ, received from the prover equals σ (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).*

The verifier program presented above is easily implemented in probabilistic polynomial-time. We do not known of a probabilistic polynomial-time implementation of the prover's program, but this is not required. We now show that the above 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. Namely,*

1. *If G_1 and G_2 are not isomorphic (i.e., $(G_1, G_2) \in GNI$) then the verifier always accept (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 τ such that the graph G' (received by the prover in Step P1) is isomorphic to the input graph G_τ. Hence, τ found by the prover in Step (P1) always equals σ 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 will show that in this case the graph G' yields no information about σ, and consequently no machine can (on input G_1, G_2 and G') set τ so that it equal σ, with probability greater than $\frac{1}{2}$. Details follow.

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

Claim 4.2.9.1: If the graphs G_1 and G_2 are isomorphic then for every graph G' it holds that

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

proof: We first claim that the sets $S_1 \stackrel{\rm def}{=} \{\pi : \pi(G_1) = G')$ and $S_2 \stackrel{\rm def}{=} \{\pi : \pi(G_2) = G')$ are of equal 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,

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

Using Bayes Rule, the claim follows.□

Using Claim 4.2.9.1, it follows that for every pair, (G_1, G_2), of isomorphic graphs and for every randomized process, R, (possibly depending on this pair) it holds that

$$\begin{aligned}
\mathsf{Pr}(R(\Pi(G_\xi))=\xi) &= \sum_{G'} \mathsf{Pr}(\Pi(G_\xi))=G') \cdot \mathsf{Pr}(R(G'))=\xi|\Pi(G_\xi)=G') \\
&= \sum_{G'} \mathsf{Pr}(\Pi(G_\xi))=G') \\
&\quad \cdot \sum_{b\in\{1,2\}} \mathsf{Pr}(R(G'))=b) \cdot \mathsf{Pr}(b=\xi|\Pi(G_\xi)=G') \\
&= \sum_{G'} \mathsf{Pr}(\Pi(G_\xi))=G') \cdot \mathsf{Pr}(R(G'))\in\{1,2\}) \cdot \frac{1}{2} \\
&\leq \frac{1}{2}
\end{aligned}$$

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 error probability in these cases equals zero). All interactive proof systems we shall consider will share this property. In fact it can be shown that every interactive proof system can be *transformed* 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 system in which the verifier *always rejects* inputs *not in* the language.

The fact that $GNI \in \mathcal{IP}$, whereas it is not known whether $GNI \in \mathcal{NP}$, is an indication to the power of interaction and randomness in the context of theorem proving. A much stronger indication is provided by the fact that every language in $\mathcal{PSPACE}$ has an interactive proof system (in fact $\mathcal{IP}$ equals $\mathcal{PSPACE}$). For further discussion see Chapter 9.

4.2.3 Augmentation to the Model

For purposes that will become more clear in the sequel 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 subprotocols inside larger protocols, the private inputs are associated with the local configurations of the machines before entering the subprotocol. In particular, the private input of the prover may contain information which enables an efficient implementation of the prover's task.

Definition 4.2.10 (interactive proof systems - revisited):

- *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 contents of this tape is call* auxiliary input.

- *The complexity of such an interactive machine is still measured as a function of the (common) input. Namely, the interactive machine A has* time complexity $t : \mathbb{N} \mapsto \mathbb{N}$ *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 contents of its random-tape and its auxiliary-input-tape as well as the contents of B's tapes) halts within $t(|x|)$ steps.*

- *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).*

- *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 $z \in \{0,1\}^*$*
  $$\mathsf{Pr}\left(\langle P(y), V(z)\rangle(x) = 1\right) \geq \frac{2}{3}$$

– Soundness: *For every $x \notin L$, every interactive machine B, and every $y, z \in \{0,1\}^*$*

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

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.

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 the above 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), actually P, is *perfect zero-knowledge* if for every probabilistic polynomial-time interactive machine V^* there exists an (*ordinary*) probabilistic polynomial-time algorithm M^* so 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. This fact is taken as evidence to the claim that V^* did not gain any knowledge from P (since the same output could have been generated without any access to P).

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 the common input or not). To demonstrate the zero-knowledge property of this "dummy prover", one may present for every verifier V^* a simulator M^* which is essentially identical to V^* (except that the communication tapes of V^* are considered as ordinary work tapes of M^*).
Unfortunately, the above formulation of perfect zero-knowledge is slightly too strict to be useful. We relax the formulation by allowing the simulator to fail, with bounded probability, to produce an interaction.

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^* so 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., $\mathsf{Pr}(m^*(x) = \alpha) = \mathsf{Pr}(M^*(x) = \alpha | M^*(x) \neq \perp)$, 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 (above) can be replaced by a stronger condition requiring that M^* outputs the special symbol (i.e., $\perp$) only with negligible probability. For example, one can require that on input x machine M^* outputs $\perp$ 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 7. Hence, whatever the verifier efficiently computes after interacting with the prover, can be efficiently computed (up to an overwhelmingly small error) by the simulator (and hence by the verifier himself).

Following the spirit of Chapters 3 and 5, we observe that for practical purposes there is no need to be able to "perfectly simulate" the output of V^* after interacting with P. Instead, it suffices to generate a probability distribution which is computationally indistinguishable from the output of V^* after interacting 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. 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 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^* so 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 interacting 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 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.

We stress that both definitions of zero-knowledge apply to interactive proof systems in the general sense (i.e., having any non-negligible gap in 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 may say that the interactive machine A is *zero-knowledge on L* if whatever can be efficiently computed after interacting with A on common input $x \in L$, can also be efficiently computed from x itself.

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 contents 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 as well as the final output are determine by these 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* $\text{view}_{V^*}^P(x)$ *a random variable describing the contents 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^* so that the ensembles $\{\text{view}_{V^*}^P(x)\}_{x\in L}$ and $\{M^*(x)\}_{x\in L}$ are computationally indistinguishable.*

A few remarks are in place. Definition 4.3.3 is obtained from Definition 4.3.2 by replacing $\langle P,V^*\rangle(x)$ for $\text{view}_{V^*}^P(x)$. The simulator M^* used in Definition 4.3.3 is related, but not equal, to the simulator used in Definition 4.3.2 (yet, this fact is not reflected in the text of these definitions). Clearly, $V^*(x)$ can be computed in (deterministic) polynomial-time from $\text{view}_{V^*}^P(x)$, for every V^*. Although the opposite direction is not always true, Definition 4.3.3 is equivalent to Definition 4.3.2 (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 alternative formulation of perfect zero-knowledge is straightforward, and clearly it is equivalent to Definition 4.3.1.

* Complexity classes based on Zero-Knowledge

Definition 4.3.4 (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, $\mathcal{PZK}$ denotes the class of languages having* perfect *zero-knowledge interactive proof systems.*

Clearly, $\mathcal{BPP} \subseteq \mathcal{PZK} \subseteq \mathcal{CZK} \subseteq \mathcal{IP}$. We believe that the first two inclusions are strict. 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 4.4.11.

* Expected polynomial-time simulators

The formulation of perfect zero-knowledge presented in Definition 4.3.1 is different from the standard definition used in the literature. The standard definition requires that the simulator always outputs 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 time. We stress that the expectation is taken over the coin tosses of the simulator (whereas the input to the simulator is fixed).

Definition 4.3.5 (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^* so 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 is "polynomial on the average". Clearly, Definition 4.3.1 implies Definition 4.3.5 – see Exercise 8. Interestingly, there exists interactive proofs which are perfect zero-knowledge with respect to the liberal definition but not known to be perfect zero-knowledge with respect to Definition 4.3.1. We prefer to adopt Definition 4.3.1, rather than Definition 4.3.5, because we wanted to avoid the notion of expected polynomial-time that is much more subtle than one realizes at first glance.

> *A parenthetical remark concerning the notion of average polynomial-time*: The naive interpretation of expected polynomial-time is having *average* running-time that is *bounded by a polynomial* in the input length. This definition of expected polynomial-time is unsatisfactory since it is *not closed under reductions* and is (too) *machine dependent*. Both aggravating phenomenon follow from the fact that a function may have an average (say over $\{0,1\}^n$) that is bounded by polynomial (in n) and yet squaring the function yields a function which is not bounded by a polynomial (in n). Hence, a better interpretation of expected polynomial-time is having running-time that is *bounded by a polynomial in a function which has average linear growing rate*.

Furthermore, the correspondence between average polynomial-time and efficient computations is more controversial than the more standard association of strict polynomial-time with efficient computations.

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

For sake of elegancy, it is customary to modify the definitions allowing *expected* polynomial-time simulators, by requiring that such simulators exist also for the interaction of *expected* polynomial-time verifiers with the prover.

4.3.2 An Example (Graph Isomorphism in PZK)

As mentioned above, 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).

Construction 4.3.6 (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 ϕ be an isomorphism between the input graphs, namely ϕ is a 1-1 and onto mapping of the vertex set V_1 to the vertex set V_2 so that $(u, v) \in E_1$ if and only if $(\pi(v), \pi(u)) \in E_2$.*

- 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 π 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 verifiers asks the prover to show an isomorphism between G' and one of the input graph, 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_σ and G').*

- Prover's second Step (P2): *If the message, σ, received from the verifier equals 2 then the prover sends π to the verifier. Otherwise (i.e., $\sigma \neq 2$), the prover sends $\pi \circ \phi$ (i.e., the composition of π on ϕ, 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 ψ, received from the prover is an isomorphism between G_σ and G' then the verifier outputs 1, otherwise it outputs 0.*

Let use denote the prover's program by P_{GI}.

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

Proposition 4.3.7 *The language GI has a perfect zero-knowledge interactive proof system. Furthermore, the programs specified in Construction 4.3.6 satisfy*

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 the prover).*

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

3. *The above 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 above protocol, sequentially, k times. We stress that in each repetition, of the above protocol, both (the prescribed) prover and verifier use coin tosses which are independent of the coins used in the other repetitions of the protocol. For further discussion see Section 4.3.4. We remark that k parallel executions will decrease the error in the soundness condition to 2^{-k} as well, but the resulting interactive proof is not known to be zero-knowledge in case 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 $GI \in \mathcal{BPP}$. Hence, Proposition 4.3.7 asserts the existence of perfect zero-knowledge proofs for languages not known to be in $\mathcal{BPP}$.

proof: We first show that the above 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) is isomorphic to both of them. Hence, if each party follows its prescribed program then the verifier always accepts (i.e., outputs 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 $\sigma \in \{1, 2\}$ so that G' and G_σ are *not* isomorphic. Hence, when the verifier follows its program, the verifier rejects (i.e., outputs 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 4.3.6 (since its output is identically 1 on inputs in the language GI). It is also not hard to simulate the output of a verifier which follows the program specified in Construction 4.3.6, except that at termination it 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 which deviates arbitrarily from the specified program.

We will 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 above it is not hard to simulate the verifier's view of the interaction with P_{GI} in case 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. Then, if V^* asks to see the isomorphism between G_1 and G'', 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" ϕ) 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 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 which is identical to the view of V^* when interacting with P_{GI} (on common input (G_1, G_2)). 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 contents of the random-tape of V^*.

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 ψ from the set of permutations over the vertex set V_τ. It then constructs a graph with vertex set V_τ and edge set
$$F \stackrel{\text{def}}{=} \{(\psi(u), \psi(v)) : (u, v) \in E_\tau\}$$
Set $G'' \stackrel{\text{def}}{=} (V_\tau, F)$.

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) above) on V^*'s random-tape, and placing G'' (constructed in step (2) above) 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 σ. To simplify the rest of the description, we *normalize* σ by setting $\sigma = 1$ if $\sigma \neq 2$ (and leave σ unchanged if $\sigma = 2$).

4. *Simulating the prover's second step (P2):* If $\sigma = \tau$ then the simulator halts with output (x, r, G'', ψ).

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 4.3.7.1: Suppose that the graphs G_1 and G_2 are isomorphic. Let ξ be a random variable uniformly distributed in $\{1, 2\}$, and $\Pi(G)$ be a random variable (independent of ξ) describing the graph obtained from the graph G by randomly relabelling its nodes (cf. Claim 4.2.9.1). Then, for every graph G'', it holds that

$$\mathsf{Pr}\left(\xi = 1 | \Pi(G_\xi) = G''\right) = \mathsf{Pr}\left(\xi = 2 | \Pi(G_\xi) = G''\right)$$

Claim 4.3.7.1 is identical to Claim 4.2.9.1 (used to demonstrate that Construction 4.2.8 constitutes an interactive proof for GNI). 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 ξ with probability exactly $\frac{1}{2}$. Hence, given G'' (constructed by the simulator in step (2)), the verifier's program yields (normalized) σ 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 4.3.7.2: Let $x = (G_1, G_2) \in GI$. Then, for every string r, graph H, and permutation ψ, it holds that

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

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 equals r. Similarly, let $\mu(x, r)$ describe the last two elements of $m^*(x)$ (conditioned on the second element equals r). Clearly, it suffices to show that $\nu(x, r)$ 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 $\nu(x, r)$ and $\mu(x, r)$ are uniformly distributed over the set

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

where $\psi(G)$ denotes the graph obtained from G by relabelling the vertices using the permutation ψ (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 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 π for which $\pi(G_2)$ is identical, 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 π 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 ψ is the 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 ψ 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 $\text{aut}(G_2)$ denotes 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 contain a different isomorphism between H and $G_{v^*(x,r,H)}$.

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 (τ, ψ), 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 (τ, ψ) 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, (τ, ψ), yielding the graph $H = \psi(G_\tau)$, only the pairs satisfying $\tau = v^*(x, r, H)$ lead to an output. Hence, for each H (which is isomorphic to G_2), the probability that $\mu(x, r) = (H, \cdot)$ equals $\text{aut}(G_2)/(|V_2|!)$. Furthermore, for each H (which is isomorphic to G_2),

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

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

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

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

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 w.r.t. Auxiliary Inputs

The definitions of zero-knowledge presented above 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 bigger protocol) the verifier interacting with the prover, on common input x, may have some additional a-priori information, encoded by a string z, which 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 4.3.6 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 below using the augmented notion of interactive proofs presented in Definition 4.2.10.

Definition 4.3.8 (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., for every $z \in \{0,1\}^*$ $\mathsf{Pr}(\langle P(y), V(z)\rangle(x)=1) \geq \frac{2}{3}$). We say that (P, V) is* **zero-knowledge with respect to auxiliary input** (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, so that the following two ensembles are computationally indistinguishable (when the distinguishing gap is considered as a function of $|x|$)*

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

Namely, for every probabilistic algorithm, D, with running-time polynomial in length of the first input, every polynomial $p(\cdot)$, and 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 the above 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. We stress that the local inputs (i.e., y and z) may not be known, even in part, to the counterpart. We also stress that the auxiliary input z is also given to the distinguishing algorithm (which may 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 length of x; see Exercise 11). For example, having allowed even one of these machines 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 which run in time polynomial in the common-input yet have huge auxiliary inputs of length exponential in the common-input).

Definition 4.3.8 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 4.3.6, 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 4.3.7 (i.e., when invoking the verifier, the simulator should provide it with the auxiliary input which is given to the simulator). In general, a demonstration of zero-knowledge can be extended to yield zero-knowledge with respect to auxiliary input, provided that the simulator used in the original demonstration works by invoking the verifier's program as a black box. All simulators presented in this book have this property.

Comment: Implicit non-uniformity in Definition 4.3.8. The non-uniform nature of Definition 4.3.8 is captured by the fact that the simulator gets an auxiliary input. It is true that this auxiliary input is also given to both the verifier program and the simulator, however if it is sufficiently long then only the distinguisher can make any use of its suffix. It follows that the simulator guaranteed in Definition 4.3.8 produces output that is indistinguishable from the real interactions also by non-uniform polynomial-size machines. Namely, for every (even non-uniform) polynomial-size circuit family, $\{C_n\}_{n\in\mathbb{N}}$, every polynomial $p(\cdot)$, and 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. 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 exists 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 a way that both V^* and M^* have no sufficient time to reach the description of C_n. For example, let $q(\cdot)$ be a polynomial bounding the running-time of both V^* and M^*, as well as the size of C_n. Then, we let z' be the string which 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., if $|z| > q(n)$ then 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 circuit evaluating algorithm, we get an algorithm D such that $D(x, z', \alpha) = C_n(x, z)$, and contradiction follows.

Comment: why not go for absolute non-uniformity? A simplified version of Definition 4.3.8 allows the verifier to be modelled by a (non-uniform) family of circuits, and requires the same of the simulator. Such a version is actually weaker as it does not guarantee an "effective" transformation of verifiers to simulators. Indeed, such a transformation is not required in Definition 4.3.8 either, but there the objects (i.e., machines) are of fixed size (whereas the circuits are of polynomial-size). Furthermore, a simplified version does not guarantee a relation between the size of the non-uniform part of the verifier and the corresponding part in the simulator, whereas in Definition 4.3.8 this part is the auxiliary-input which remains unchanged. These issues arise when trying to prove a sequential composition of a non-constant number of zero-knowledge proof systems.

Author's Note: *Re-examine the discussions above!*

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 are closed under sequential composition. Namely, if one executes one zero-knowledge proof after another then the composed execution must be zero-knowledge. The same should remain valid even if one executes polynomially many proofs one after the other. Indeed, as we will shortly see, the revised definition of zero-knowledge (i.e., Definition 4.3.8) 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 to the necessity of augmenting this definition (as done in Definition 4.3.8).

In addition to its conceptual importance, the Sequential Composition Lemma is an important tool in the design of zero-knowledge proof systems. Typically, these proof system consists of many repetitions of a atomic zero-knowledge proof. Loosely speaking, the atomic proof provides some (but not much) statistical evidence to 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 accepting probability of string in the language and strings outside the language. For example, in Construction 4.3.6 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 4.3.6 for k times yields 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 by 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 on zero-knowledge with respect to auxiliary input we refer to all possible auxiliary inputs for the verifier.

Lemma 4.3.9 (Sequential Composition Lemma): *Let P be an interactive machine (i.e., a prover) which 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 "end of proof" is introduced for technical purposes (and is redundant in all known provers for which the number of messages sent is easily computed from the length of the common input). Clearly, every machine P can be easily modified so that its last message bears an appropriate symbol (as assumed above), and doing so preserves the zero-knowledge properties of P (as well as completeness and soundness conditions).

The Lemma remain valid also if one allows auxiliary input to the prover. The extension is straightforward. The lemma ignores other aspects of repeating an interactive proof several times; specifically, the effect on the gap between the accepting probability of inputs inside and outside of the language. This aspect of repetition is discussed in the previous section (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^*, which simulates 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.

The first step in carrying-out the above 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 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 interaction of a related machine, denoted V^{**}, with P. Namely,

Claim 4.3.9.1: There exists a probabilistic polynomial-time V^{**} so that for every common input x and auxiliary input z it holds that

$$\begin{aligned} \langle P_Q, V^*(z)\rangle(x) &= Z^{(Q(|x|))} \\ & \quad \text{where } Z^{(0)} \stackrel{\text{def}}{=} z \text{ and } Z^{(i+1)} \stackrel{\text{def}}{=} \langle P, V^{**}(Z^{(i)})\rangle(x) \end{aligned}$$

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 $i+1^{\text{st}}$ interaction equals the output of V^{**} after the i^{th} interaction (i.e., $Z^{(i)}$).

proof: Consider an interaction of $V^*(z)$ with P_Q, on common input x. 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 above read-only tapes again. Likewise, V^* is modified so that it starts each active period 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, the above 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 V^*.

Considering the contents of the work-tape of V^* at the end of each of the $Q(|x|)$ phases (of interactions with P_Q), naturally leads us to the construction of V^{**}. Namely, on common input x and auxiliary input z', machine V^{**} starts by copying z' into the work-tape of V^*. Next, machine V^{**} simulates a *single phase* of the interaction of V^* with P_Q (on input x) starting with the above contents of the work-tape of V^* (instead of starting with an empty work-tape). The invoked machine V^* regards the communication-tapes of machine V^{**} as its own communication-tapes. Finally, V^{**} terminates by outputting the current contents of the work-tape of V^*. Actually, the above description should be slightly modified to deal differently with the first phase in the interaction with P_Q. Specifically, V^{**} copies z' into the work-tape of V^* only if z' encodes a contents of the work-tape of V^* (we assume, w.l.o.g., that the contents 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, the random-input-tape and the auxiliary-input-tape) as its own. Observe that $Z^{(1)} \stackrel{\text{def}}{=} \langle P, V^{**}(z)\rangle(x)$ describes the contents of the work-tape of V^* after one phase, and $Z^{(i)} \stackrel{\text{def}}{=} \langle P, V^{**}(Z^{(i-1)})\rangle(x)$ describes the contents of the work-tape of V^* after i phases. The claim follows. □

Since 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 which simulates these interactions in time polynomial in the length of the first input. Let M^{**}

denote this simulator. We may assume, without loss of generality, that with overwhelmingly high probability M^{**} halts with output (as we can increase the probability of output by successive applications of M^{**}). Furthermore, for sake of simplicity, we assume in the rest of this proof that M^{**} always halts with output. Namely, 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

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

We are now ready to present the construction of a simulator, M^*, that simulates the "real" output of V^* after interaction with P_Q. Machine M^* uses the above guaranteed 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 M^{**} on input $(x, z^{(i-1)})$. After $Q(|x|)$ phases are completed, machine M^* stops outputting $z^{(Q(|x|))}$.

Clearly, machine M^*, constructed above, runs in time polynomial in its first input. (For non-constant $Q(\cdot)$ it is crucial here that the running-time of M^* is polynomial in the length of the first input, rather than being polynomial in the length of both inputs.) It is left to show that machine M^* indeed produces output which is polynomially indistinguishable from the output of V^* (after interacting with P_Q). Namely,

Claim 4.3.9.2: 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

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

proof sketch: We use a hybrid argument. In particular, we define the following $Q(|x|) + 1$ hybrids. The i^{th} hybrid, $0 \leq i \leq Q(|x|)$, 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 $Q(m) - k$ 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.

$$\begin{aligned} H^{(i)}(x, z) \quad &\stackrel{\text{def}}{=} \quad M^{**}_{Q(m)-i}(x, Z^{(i)}) \\ &\qquad \text{where } \; Z^{(0)} \stackrel{\text{def}}{=} z \; \text{ and } \; Z^{(j+1)} \stackrel{\text{def}}{=} \langle P, V^{**}(Z^{(j)})\rangle(x) \\ &\qquad M^{**}_0(x, z') \stackrel{\text{def}}{=} (x, z') \; \text{ and } \; M^{**}_j(x, z') \stackrel{\text{def}}{=} M^{**}_{j-1}(x, M^{**}(x, z')) \end{aligned}$$

Using Claim 4.3.9.1, the $Q(|x|)^{\text{th}}$ hybrid (i.e., $H^{(Q(|x|))}(x, z)$) 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., $H^{(0)}(x, z)$) equals $M^*(x, z)$). Hence, all that is required to complete the proof is to show that every two adjacent hybrids are polynomially indistinguishable (as this would imply that the extreme hybrids, $H^{(Q(m))}$ and $H^{(0)}$, are indistinguishable too). To this end, we rewrite the $i^{\rm th}$ and $i-1^{\rm st}$ hybrids as follows.

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

where $Z^{(i-1)}$ is as defined above (in the definition of the hybrids).

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' so 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. Incorporating algorithm M^{**} into D, we get a new algorithm D', with running time polynomially related to the former algorithms, which distinguishes the random variables $(x, z', \langle P, V^{**}(z')\rangle(x))$ and $(x, z', M^{**}(x,z'))$ at least as well. (Further details are presented below.) Contradiction (to the hypothesis that M^{**} simulates (P, V^{**})) follows. $\Box$

The lemma follows. $\blacksquare$

Further details concerning the proof of Claim 4.3.9.2: The proof of Claim 4.3.9.2 is rather sketchy. The main thing which is missing are details 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 4.3.9.2. These details are presented below, and the reader is encouraged *not* to skip them.

Let us start with the non-problematic part. We assume, to the contradiction, that there exists a probabilistic polynomial-time algorithm, D, and a polynomial $p(\cdot)$, so 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 $i \in \{1, ..., Q(|x|)\}$ such that

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

Denote $\epsilon(n) \stackrel{\rm def}{=} 1/(Q(n) \cdot p(n))$. Combining the definition of the $i^{\rm th}$ and $i-1^{\rm st}$ hybrids with an averaging argument, it follows that for each such x, z and i, there exists a z', in the support of $Z^{(i-1)}$ (defined as above), such that

$$
\begin{aligned}
&|\mathsf{Pr}(D(x,z',M^{**}_{Q(|x|)-i}\langle P, V^{**}(z')\rangle(x)) = 1) \\
&\quad -\mathsf{Pr}(D(x,z',M^{**}_{Q(|x|)-i}(M^{**}(x,z'))) = 1)| > \epsilon(|x|)
\end{aligned}
$$

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 algorithms D and M^{**}, provided we "know" i. The problem is resolved using the fact, pointed out at the end of Subsection 4.3.3, that the output of M^{**} is undistinguished from the interactions of V^{**} with the prover even with respect to non-uniform polynomial-size circuits. Details follow.

We construct a polynomial-size circuit family, denoted $\{C_n\}$, which distinguishes $(x, z', \langle P, V^{**}(z'')\rangle(x))$ and $(x, z', M^{**}(x, z''))$, for the above-mentioned (x, z') pairs. On input x (supposedly in $L \cap \{0,1\}^n$) and α (supposedly in either $(x, z', \langle P, V^{**}(z'')\rangle(x))$ or $(x, z', M^{**}(x, z'')))$, the circuit C_n, incorporating (the above-mentioned) i, uses algorithm M^{**} to compute $\beta = M_{Q(|x|)-i}(x, \alpha)$. Next C_n, using algorithm D, computes $\sigma = D((x, z'), \beta)$ and halts outputting σ. Contradiction (to the hypothesis that M^{**} is a simulator for (P, V^{**})) follows. $\square$

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 zero-knowledge proofs that when played twice in parallel do yield knowledge (to a "cheating verifier"). For further details see Subsection 4.5.

The fact that zero-knowledge is not preserved under parallel composition of protocols is indeed bad news. One may even think that this fact is a conceptually annoying phenomenon. We disagree with this feeling. Our feeling is that the behaviour 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 the behaviour under sequential composition. Furthermore, the only advantage of parallel composition over sequential composition is in efficiency. Hence, we don't consider the non-closure under parallel composition to be a conceptual weakness of the formulation of zero-knowledge. Yet, the "non-closure" of zero-knowledge motivates the search for either weaker or stronger notions which are preserved under parallel composition. For further details, the reader is referred to Sections 4.9 and 4.6.

4.4 Zero-Knowledge Proofs for NP

This section presents the main thrust of the entire 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, we observe that almost all statements one wish to prove in practice can be encoded as claims concerning membership in languages in $\mathcal{NP}$.

The method, for constructing zero-knowledge proofs for NP-languages, makes essential use of the concept of *bit commitment*. Hence, we start with a presentation of this concept.

4.4.1 Commitment Schemes

Commitment schemes are a basic ingredient in many cryptographic protocols. The are used to enable a party to commit itself to a value while keeping it secret. In a latter 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 analogue of nontransparent sealed envelopes. By putting a note in such an envelope a party commits itself to the contents of the note while keeping it 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* so the following two conflicting requirements are satisfied.

1. *Secrecy*: 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*: Given the transcript of the interaction in the first phase, there exists at most one value which the receiver may 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 is *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 not of the sender's value) to the receiver, whereas the commit phase does "commit" the sender to a unique value (in the sense that in the reveal phase the receiver may 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:*

- 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 v.*

- Secrecy: *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 random variables describ-*

ing 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 polynomially-indistinguishable.*

- Unambiguity:
  Preliminaries

  – *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* ($\overline{m}$).
  – *Let* $\sigma \in \{0,1\}$. *We say that a receiver's view (of such interaction),* $(r, \overline{m})$*, is a* possible σ-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 *which uses local coins* s *and has input* $(\sigma, 1^n)$*. (Using the notation of Definition 4.3.3, the condition may be expressed as* $\overline{m} = \mathrm{view}_{R(1^n,r)}^{S(\sigma,1^n,s)}$*.)*
  – *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) which together with these coin tosses forms an ambiguous receiver view. Namely, that 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* (above) is analogous to the definition of indistinguishability of encryptions (i.e., Definition `[missing(enc-indist.def)]`). An equivalent formulation analogous to semantic security (i.e., Definition `[missing(enc-semant.def)]`) can be presented, but is less useful in typical applications of commitment schemes. In any case, the secrecy requirement is a computational one. On the other hand, the *unambiguity requirement* has an information theoretic flavour (i.e., it does not refer to computational powers). A dual definition, requiring information theoretic secrecy and computational unfeasibility of creating ambiguities, is presented in Subsection 4.8.2.

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 the above 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 (e.g., the set of legitimate public-keys should be efficiently recognizable). In any case, public-key encryption schemes have additional properties not required of commitment schemes and their existence seems to require stronger intractability assumptions. An alternative construction, presented below, 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).

Construction 4.4.2 (simple bit commitment): *Let $f : \{0,1\}^* \mapsto \{0,1\}^*$ be a function, and $b : \{0,1\}^* \mapsto \{0,1\}$ 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. reveal phase: *In the reveal phase, the sender reveals 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 (α, σ) is the receiver's view of the commit phase.*

Proposition 4.4.3 *Let $f : \{0,1\}^* \mapsto \{0,1\}^*$ be a length preserving 1-1 one-way function, and $b : \{0,1\}^* \mapsto \{0,1\}$ 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 receiver view (α, σ), there is a unique $s \in \{0,1\}^{|\alpha|}$ so that $f(s) = \alpha$ and hence a unique $v \in \{0,1\}$ so 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 which is based on the weakest assumption possible: the existence of one-way function. Proving the 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 3.3.4 and 3.5.12), the existence of one-way functions imply the existence of pseudorandom generators expanding n-bit strings into $3n$-bit strings. We will use such a pseudorandom generator in the construction presented below.

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, were $\alpha \oplus \beta$ denote the bit-by-bit exclusive-or of the strings α and β. Then, the sender may 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's k-bit long string). Unfortunately, the above assumption cannot be justified, in general, and a slightly more complex variant is required. The key observation is that for most strings $\beta \in \{0,1\}^{3n}$ the sets $\{G(s) : s \in \{0,1\}^n\}$ and $\{G(s) \oplus \beta : s \in \{0,1\}^n\}$ are disjoint. Such a string β is called *good*. This observation suggests the following protocol. The receiver uniformly selects $\beta \in \{0,1\}^{3n}$, hoping that it is good, and 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 $G(s) \oplus \beta$ otherwise.

Construction 4.4.4 (bit commitment under general assumptions): *Let $G : \{0,1\}^* \mapsto \{0,1\}^*$ be a function so 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. 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 the value 1 if $G(s) \oplus r = \alpha$, where (r, α) is the receiver's view of the commit phase.*

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 $\beta \in \{0,1\}^{3n}$ *good* if the sets $\{G(s) : s \in \{0,1\}^n\}$ and $\{G(s) \oplus \beta : s \in \{0,1\}^n\}$ are disjoint. We say that $\beta \in \{0,1\}^{3n}$ *yields a collision between the seeds s_1 and s_2* if $G(s_1) = G(s_2) \oplus \beta$. Clearly, β is good if it does not yield a collision between any pair of seeds. On the other hand, there is a unique string β which yields a collision between a given pair of seeds (i.e., $\beta = G(s_1) \oplus G(s_2)$). Since there are 2^{2n} possible pairs of seeds, at most 2^{2n} strings yield collisions between seeds and 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. 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. When defining the secrecy of such schemes the reader is advised to consult Definition [missing(enc-indist.def)]). 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 case is even more straightforward.

In the rest of this section we will need commitment schemes with a seemingly stronger secrecy requirement than defined above. Specifically, instead of requiring secrecy with respect to all polynomial-time machines, we will 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 nonuniform secrecy can be constructed, following the same constructions used in the uniform case.

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 subsection 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 graphs (i.e., no parallel edges or self-loops) that can be *vertex-colored* using 3 colors so that no two adjacent vertices are given the same color. Formally, a graph $G=(V,E)$, is *3-colorable*, if there exists a mapping $\phi : V \mapsto \{1,2,3\}$ so 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 yields no knowledge and yet all the templates put together 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 nontransparent 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 his confidence in the validity of the claim (that the graph is 3-colorable) increases. A concrete implementation of this abstract scheme follows.

To prove that the graph $G = (V,E)$ is 3-colorable, the prover generates a random 3-coloring of the graph, denoted ϕ (actually a random relabelling 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 $(\phi(u), \phi(v)), (u,v) \in E$, constitutes a template to the claim that G is 3-colorable). Each single template (being merely a random pair of distinct elements in $\{1,2,3\}$) yield no knowledge. However, if all the templates are OK then the graph must be 3-colorable. Consequently, graphs which are not 3-colorable must contain at least one bad template and hence are rejected with non-negligible 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 ψ be a 3-coloring of G. The prover selects a random permutation, π, over $\{1,2,3\}$, and sets $\phi(v) \stackrel{\mathrm{def}}{=} \pi(\psi(v))$, for each $v \in V$. Hence, the prover forms a random relabelling of the 3-coloring ψ. The prover sends the verifier a sequence of $|V|$ locked and nontransparent boxes so 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 accept always. On the other hand, if the input graph is not 3-colorable then any contents 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 above protocol exhibits a non-negligible gap in the accepting probabilities between the case of inputs in $G3C$ and inputs not in $G3C$. The zero-knowledge property follows easily, in this abstract setting, since 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 since the boxes are not totally ineffected by their contents (but are rather effected, yet in an indistinguishable manner). Finally, we remark that the confidence in the validity of the claim (that the input graph is 3-colorable) may be increased by sequentially applying the above proof sufficient many times. (In fact if the boxes are perfect as assumed above then one can also use parallel repetitions.)

4.4.2.2 The interactive proof

We now turn to the digital implementation of the above 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 in 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 σ.

Construction 4.4.6 (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* $V = \{1, ..., n\}$.
- Auxiliary Input to the Prover: *A 3-coloring of* G, *denoted* ψ.
- Prover's first step (P1): *The prover selects a random permutation,* π, *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* $s_1, ..., s_n \in \{0,1\}^n$, *computes* $c_i = C_{s_i}(\phi(i))$, *for each* $i \in V$, *and sends* $c_1, ..., c_n$ *to the verifier;*
- Verifier's first step (V1): *The verifier uniformly selects an edge* $(u, v) \in E$, *and sends it to the prover;*
- Motivating Remark: *The verifier asks to inspect the colors of vertices* u *and* v*;*
- Prover's second step (P2): *Without loss of generality, we may assume that the message received for 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 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 the values corresponding to commitments* u *and* v *were revealed correctly and whether these values are different. Namely, upon receiving* (s, σ) *and* (s', τ), *the verifier checks whether* $c_u = C_s(\sigma)$, $c_v = C_{s'}(\tau)$, *and* $\sigma \neq \tau$ *(and both in* $\{1, 2, 3\}$*). If all conditions hold then the verifier accepts. Otherwise it rejects.*

Let us denote the above prover's program by P_{G3C}.

We stress that both the programs of the verifier and of the prover can be implemented in probabilistic polynomial-time. In case of the prover's program this property is made possible by the use of the auxiliary input to the prover. As we will shortly see, the above 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 the above protocol is zero-knowledge (also with respect to auxiliary input). Using the Sequential Composition Lemma (Lemma 4.3.9), it follows that also polynomially many sequential applications of this protocol preserve the zero-knowledge property.

Proposition 4.4.7 *Suppose that the commitment scheme used in Construction 4.4.6 satisfies the (nonuniform) secrecy and the unambiguity requirements. Then Construction 4.4.6 constitutes an auxiliary input zero-knowledge (generalized) interactive proof for $G3C$.*

For further discussion of Construction 4.4.6 see remarks at the end of the current subsection.

4.4.2.3 Proof of Proposition 4.4.7

We first prove that Construction 4.4.6 constitutes a weak interactive proof for $G3C$. Assume first that the input graph is indeed 3-colorable. Then if the prover follows the program in the construction then 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 exists). 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 exists an edge $(u, v) \in E$ so 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 $1/|E|$. Hence there is a non-negligible (in the input length) gap between the accepting probabilities in case the input is in $G3C$ and in case it is not.

We now turn to show that P_{G3C}, the prover in Construction 4.4.6, is indeed zero-knowledge for $G3C$. The claim is proven without reference to auxiliary input (to the verifier), yet extending the argument to auxiliary input zero-knowledge is straightforward. Again, we will 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 4.3.6) it is quite 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 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 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 $e_1, ..., e_n \in \{1, 2, 3\}$, and constructs a commitment to each of them. These e_i's constitute a "pseudo-coloring" of the graph, in which the end-points of each edge are colored differently with probability $\frac{2}{3}$. In doing so, the simulator behaves very differently from P_{G3C}, but nevertheless the sequence of commitments so 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) so that $e_u \neq e_v$ then the simulator can indeed answer correctly, and 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) so 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 don't assume that the simulator a-priori "knows" 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 which is 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 detail follow. We start with a detailed description of the simulator.

Simulator M^*. On input a graph $G = (V, E)$, 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 $r \in \{0, 1\}^{q(|x|)}$, to be used as the contents of the local random tape of V^*.

2. *Simulating the prover's first step (P1):* The simulator M^* uniformly and independently selects n values $e_1, ..., e_n \in \{1, 2, 3\}$ and n random strings $s_1, ..., s_n \in \{0, 1\}^n$ 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) above) on V^*'s "local random tape", and placing the sequence $(d_1, ..., d_n)$ (constructed in step (2) above) 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) in 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 $(G, r, (d_1, ..., d_n), (s_u, e_u, s_v, e_v))$.

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 above simulator n times and outputting the first output different from $\perp$. We now turn to prove the above two claims.

Claim 4.4.7.1: For every sufficiently large graph, $G=(V,E)$, the probability that $M^*(G) = \perp$ is bounded above by $\frac{1}{2}$.

proof: As above, n will denote the cardinality of the vertex set of G. Let us denote by $p_{u,v}(G, r, (e_1, ..., e_n))$ the probability, taken over all the choices of the $s_1, ..., s_n \in \{0,1\}^n$, that V^*, on input G, random coins r, and prover message $(C_{s_1}(e_1), ..., C_{s_n}(e_n))$, replies with the message (u, v). We assume, for simplicity, that V^* always answers with an edge of G (since otherwise its message is anyhow 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

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

Actually, we can prove the following.

Request Obliviousness Subclaim: 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 Subclaim is proven using the non-uniform secrecy of the commitment scheme. The reader should be able to fill-up the details of such a proof at this stage. Nevertheless, a proof of the subclaim follows.

> *Proof of the Request Obliviousness Subclaim*: 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-vertices 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 $\alpha_n, \beta_n \in \{1,2,3\}^n$ so that
>
> $$|p_{u_n,v_n}(G_n, r_n, \alpha_n) - p_{u_n,v_n}(G_n, r_n, \beta_n)| > \frac{1}{p(n)}$$
>
> 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 commitment to either α_n or β_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 α_n from commitments to β_n, since
>
> $$\mathsf{Pr}(A_n(C_{U_{n^2}}(\gamma)) = 1) = p_{u_n,v_n}(G_n, r_n, \gamma)$$

where U_k denotes, as usual, a random variable uniformly distributed over $\{0,1\}^k$. Contradiction to the (non-uniform) secrecy of the commitment scheme follows by a standard hybrid argument (which relates the indistinguishability of sequences to the indistinguishability of single commitments).

Returning to the proof of Claim 4.4.7.1, we now use the above subclaim to upper bound the probability that the simulator outputs $\perp$. The intuition is simple. Since 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 if he would have made the request without looking at the commitment. A formal (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 case the verifier on input G, random tape r, and a commitment to some pseudo-coloring $(e_1, ..., e_n)$, asks to inspect an edge (u,v) which is illegally colored (i.e., $e_u = e_v$). Let $E_{(e_1,...,e_n)}$ denote the set of edges $(u,v) \in E$ that are illegally colored (i.e., satisfy $e_u = e_v$) with respect to $(e_1, ..., e_n)$. Then, fixing an arbitrary r and considering all possible choices of $(e_1, ..., e_n) \in \{1,2,3\}^n$,

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

(Recall that $p_{u,v}(G, r, \overline{e})$ denotes the probability that the verifier asks 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 $(e_1, ..., e_n) \in \{1,2,3\}^n$ satisfying $e_u = e_v$. Clearly, $|B_{u,v}| = 3^{n-1}$. By straightforward calculation we get

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

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 4.4.7.2: 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 $\{\text{view}_{V^*}^{P_{G3C}}(G)\}_{G \in G3C}$. Namely, for every probabilistic polynomial-time algorithm, A, every polynomial $p(\cdot)$, and all sufficiently large graph $G=(V,E)$,

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

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. In first glance it seems that Claim 4.4.7.2 follows easily from the secrecy property of the commitment scheme. Indeed, Claim 4.4.7.2 is proven using the secrecy property of the commitment scheme, yet the proof is more complex than one anticipates (at first glance). The difficulty lies in the fact that the above ensembles consist not only of commitments to values, but also of an opening of some of the values. Furthermore, the choice of which commitments are to be opened depends on the entire sequence of commitments.

proof: 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 case the verifier asked to inspect the edge (u,v). Specifically

- $\mu_{u,v}(G)$ describes $M^*(G)$ conditioned on $M^*(G)$ containing the "reveal information" for vertices u and v.
- $\nu_{u,v}(G)$ describes $\text{view}_{V^*}^{P_{G3C}}(G)$ conditioned on $\text{view}_{V^*}^{P_{G3C}}(G)$ containing the "reveal information" for vertices u and v.

Let $p_{u,v}(G)$ denote the probability that $M^*(G)$ contains "reveal information" for vertices u and v, conditioned on $M^*(G) \neq \perp$. Similarly, let $q_{u,v}(G)$ denote the probability that $\text{view}_{V^*}^{P_{G3C}}(G)$ contains "reveal information" for vertices u and v.

Assume, in the 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 noticeable difference between the probabilistic profile of the requests of V^* when interacting with $PG3C$ and the requests of V^* when invoked by M^*. 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-vertices graph $G_n=(V_n,E_n)$, and an edge $(u_n,v_n) \in E_n$, so that

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

Case 2: An algorithm distinguishing the above ensembles does so also conditioned on V^* asking for a particular edge. Furthermore, this request occurs with noticeable probability which is about the same in 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-vertices graph $G_n = (V_n, E_n)$, and an edge $(u_n, v_n) \in E_n$, so that the following conditions hold

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

Case 1 can be immediately discarded since it leads easily to contradiction (to the non-uniform secrecy of the commitment scheme). The idea is to use the Request Obliviousness Subclaim appearing in the proof of Claim 4.4.7.1. Details are omitted. We are thus left with Case 2.

We are now going to show that also Case 2 leads to contradiction. To this end we will 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 by either 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 generated an n-long sequence, the circuit feeds it to V^*, and depending on V^*'s behaviour 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 ψ_n the (lexicographically first) 3-coloring of G_n used by the prover. 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 ψ_n, and the edge (u_n, v_n), all being those 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.

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

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

- For each $i \in V_n - \{u_n, v_n\}$, 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 $2n+2$ of the y_j's are not used at all.
- The circuit uniformly selects $s_u, s_v \in \{0,1\}^n$, and sets $c_{u_n} = C_{s_{u_n}}(\phi(u_n))$ and $c_{v_n} = C_{s_{v_n}}(\phi(v_n))$.
- 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 $(c_1, ..., c_n)$ (constructed above) 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 1.
- Otherwise (i.e., $m = (u_n, v_n)$), the circuit invokes algorithm A and outputs
$$A(G_n, r, (c_1, ..., c_n), (s_{u_n}, \phi(u_n), s_{v_n}, \phi(v_n)))$$

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 outputs 1 on input a random commitment to the sequence $1^n2^n3^n$. The reader can easily verify that the sequence $(c_1, ..., c_n)$ constructed by circuit A_n is distributed identically to the sequence sent by the prover in step (P1). Hence, letting $C(\gamma)$ denote a random commitment to a sequence $\gamma \in \{1,2,3\}^*$, we get

$$\begin{aligned}\mathsf{Pr}(A_n(C(1^n2^n3^n)) = 1) &= (1 - q_{u_n,v_n}(G_n)) \\ &\quad + q_{u_n,v_n}(G_n) \cdot \mathsf{Pr}(A(\nu_{u_n,v_n}(G_n)) = 1)\end{aligned}$$

On the other hand, we consider the probability that circuit A_n outputs 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 $(c_1, ..., c_n)$ constructed by circuit A_n is distributed identically to the sequence $(d_1, ..., d_n)$ generated by the simulator in step (2), conditioned on $d_{u_n} \neq d_{v_n}$. Letting T_{3n} denote a random variable uniformly distributed over $\{1,2,3\}^{3n}$, we get

$$\begin{aligned}\mathsf{Pr}(A_n(C(T_{3n}) = 1) &= (1 - p_{u_n,v_n}(G_n)) \\ &\quad + p_{u_n,v_n}(G_n) \cdot \mathsf{Pr}(A(\mu_{u_n,v_n}(G_n)) = 1)\end{aligned}$$

Using the conditions of Case 2, and omitting G_n from the notation, we get

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

Hence, the circuit family $\{A_n\}$ distinguishes commitments to $\{1^n 2^n 3^n\}$ from commitments to $\{T_{3n}\}$. Combining an averaging argument with a hybrid argument, we conclude that there exists a polynomial-size circuit family which distinguishes commitments. This contradicts the non-uniform secrecy of the commitment scheme.

Having reached contradiction in both cases, Claim 4.4.7.2. $\square$

Combining Claims 4.4.7.1 and 4.4.7.2, the zero-knowledge property of P_{G3C} follows. This completes the proof of the proposition. ■

4.4.2.4 Concluding remarks

Construction 4.4.6 has been presented using a unidirectional commitment scheme. A fundamental property of such schemes is that their secrecy is preserved also in case (polynomially) many instances are invoked simultaneously. The proof of Proposition 4.4.7 indeed took advantage on this property. We remark that Construction 4.4.4 also possesses this simultaneous secrecy property, and hence the proof of Proposition 4.4.7 can be carried out also if the commitment scheme in 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 4.4.7 assumes the existence of a *nonuniformly* secure commitment scheme. The proof of the proposition makes essential use of the nonuniform security by incorporating instances on which the zero-knowledge property fails into circuits which contradict the security hypothesis. We stress that the sequence of "bad" instances is not necessarily constructible by efficient (uniform) machines. Put in other words, the zero-knowledge requirement has some nonuniform flavour. A *uniform analogue of zero-knowledge* would require only that it is infeasible to find instances on which a verifier gains knowledge (and not that such instances do not exist at all). Using a *uniformly* secure commitment scheme, Construction 4.4.6 can be shown to be *uniformly* zero-knowledge.

By itself, Construction 4.4.6 has little practical value, since it offers very moderate acceptance gap (between inputs 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 accepting) yields 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 4.3.9), 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 one can prove an analogous statement with respect to the interactive proof which results by performing these iteration in parallel. See Section 4.5.

An important property of Construction 4.4.6 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 to the applications of Construction 4.4.6 to the design of cryptographic protocols.

As admitted in the beginning of the current subsection, 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. Such constructions will use the same underlying ideas as those presented in 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 4.4.6) 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 4.4.6) 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 $|y| \leq p(|x|)$.
- There exists a polynomial-time algorithm for deciding membership in R.
- $L = \{x : \exists w \text{ s.t. } (x, w) \in R\}$.

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. Secondly, since $G3C$ 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 $f(x) \in G3C$. Thirdly, 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 $g_L(w)$ is a 3-coloring of $f_L(x)$.

We stress that the above additional property is not required by the standard definition of a Karp-reduction. Yet, it can be easily verified 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. (See Exercise 17.) Using these conventions, we are ready to "reduce" the construction of zero-knowledge proof for $\mathcal{NP}$ to a zero-knowledge proof system for $G3C$.

Construction 4.4.8 (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 $\psi \stackrel{\text{def}}{=} g_L(w)$.*
- 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 ψ.*

Proposition 4.4.9 *Suppose that the subprotocol used in the last step of Construction 4.4.8 is indeed an auxiliary input zero-knowledge proof for $G3C$. Then Construction 4.4.8 constitutes an auxiliary input zero-knowledge proof for L.*

Proof: The fact that Construction 4.4.8 constitutes an interactive proof for L is immediate from the validity of the reduction (and the fact that it uses an interactive proof for $G3C$). In first glance it seems that the zero-knowledge property of Construction 4.4.8 follows as immediately. There is however a minor issue that one should not ignore. The verifier in the zero-knowledge proof for $G3C$, invoked in Construction 4.4.8, possesses not only the common input graph G but also the original common input x which 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 zero-knowledge also with respect to auxiliary input. can be dealt with using auxiliary input to the verifier in Details follow.

Suppose we need to simulate the interaction of a machine V^* with the prover, on common input x. Without loss of generality we may assume that machine V^* invokes an interactive machine V^{**} which interacts with the prover of the $G3C$ interactive proof, on common input $G = f_L(x)$ and having 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 verifier's auxiliary input x). Hence, the simulator for Construction 4.4.8, 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. ■

We remark that an alternative way of resolving the minor difficulty addressed above 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 18). In any case, we immediately get

Theorem 4.4.10 *Suppose that there exists a commitment scheme satisfying the (nonuniform) secrecy and the 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 exists (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 4.4.10 is to enable one party to prove some property of its secrets without revealing the secrets. For concreteness, consider a party, denoted S, sending encrypted messages (over a public channel) to various parties, denoted $R_1, ..., R_t$, and wishing to prove to some other party, denoted V, that all the corresponding plaintext messages are identical. Further suppose that the messages are sent to the receivers (i.e., the R_i's) using a secure public-key encryption scheme, and let $E_i(\cdot)$ denote the (probabilistic) encryption employed when sending a message to R_i. Namely, to send message M_i to R_i, the sender uniformly chooses $r_i \in \{0,1\}^n$, computes the encryption $E_i(r_i, M_i)$, and transmits it over the public channel. In order to prove that $C_1 = E_1(r_1, M)$ and $C_2 = E_2(r_2, M)$ both encrypt the same message it suffices to reveal r_1, r_2 and M. However, doing so reveals the message M to the verifier. Instead, one can prove in zero-knowledge that there exists r_1, r_2 and M such that $C_1 = E_1(r_1, M)$ and $C_2 = E_2(r_2, M)$. The existence of such a zero-knowledge proof follows from Theorem 4.4.10 and the fact that the statement to be proven is of NP-type. Formally, we define a language

$$L \stackrel{\text{def}}{=} \{(C_1, C_2) : \exists r_1, r_2, M \text{ s.t. } C_1 = E_1(r_1, M) \text{ and } C_2 = E_2(r_2, M)\}$$

Clearly, the language L is in $\mathcal{NP}$, and hence Theorem 4.4.10 can be applied. Additional examples are presented in Exercise 19.

Zero-Knowledge for any language in IP

Interestingly, the result of Theorem 4.4.10 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 proof system. Namely,

Theorem 4.4.11 *Suppose that there exists a commitment scheme satisfying the (nonuniform) secrecy and unambiguity requirements. Then every language in $\mathcal{IP}$ has a zero-knowledge proof system.*

We believe that this extension does not have much practical significance. Theorem 4.4.11 is proven by first converting the interactive proof for L into one in which the verifier uses only "public coins" (i.e., an Arthur-Merlin proof); see Chapter 8. Next, the verifier's coin tosses are forced to be almost unbiased by using a coin tossing protocols (see section ****???). Finally, the prover's replies are sent using a commitment scheme, At the end of the interaction the prover proves in zero-knowledge that the original verifier would have accepted the hidden transcript (this is an NP-statement).

4.4.4 Efficiency 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 which 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 can they be.

In order to establish common grounds 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 proof is under what assumptions (if at all) are they valid. Throughout this entire subsection we stick to the assumption used so far (i.e., the existence of one-way functions).

4.4.4.1 Standard efficiency measures

Natural and standard efficiency measures to consider are

- 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 4.4.6. Recall that this proof system has very moderate acceptance gap, specifically $1/|E|$, on common input graph $G = (V, E)$. So Construction 4.4.6 has to be applied sequentially $k \cdot |E|$ 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 may 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 16, one can present zero-knowledge proofs for the Hamiltonian Circuit Problem (again with error 2^{-k}) having round complexity $O(k)$, bit complexity $O(k \cdot |V|^{2+\epsilon})$, and computational complexity $O(k \cdot |V|^{2+O(\epsilon)})$, where $\epsilon > 0$ is a constant depending on the desired security of the commitment scheme (in Construction 4.4.6 and in Exercise 16 we chose $\epsilon = 1$). Note that complexities depending on the instance size are effected 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 number of rounds and negligible error probability. This goal is pursued in Section 4.9.

4.4.4.2 Knowledge Tightness: a particular efficiency measure

The above efficiency measures are general in the sense that they are applicable to any protocol (independent on whether it is zero-knowledge or not). A particular measure of efficiency applicable to zero-knowledge protocols is their *knowledge tightness*. Intuitively, knowledge tightness is a refinement of zero-knowledge which is aimed at measuring the "actual security" of the proof system. Namely, how much harder does the verifier need to work, when not interacting with the prover, in order to compute something which it can computes 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 so far, as well as all known simulator, operate by repeated random trials and hence an instructive measure of tightness should consider their expected running-time (assuming they never err (i.e., output the special $\perp$ symbol)) rather than the worst case.

Definition 4.4.12 (knowledge tightness): *Let $t : \mathbb{N} \mapsto \mathbb{N}$ 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*

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

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

We assume a model of computation allowing one machine to invoke another machine at the cost of merely the running-time of the latter machine. The purpose of polynomial $p(\cdot)$, in the above definition, is to take care of generic overhead created by the simulation (this is important 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 proof, 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 4.3.6 has knowledge tightness 2, whereas Construction 4.4.6 has knowledge tightness 3/2. We believe that knowledge tightness is a very important efficiency consideration and that it desirable to have it be a constant.

4.5 * Negative Results

In this section we review some negative results concerning zero-knowledge. These results can be viewed as evidence to the belief that some of the shortcomings of the results and constructions presented in previous sections are unavoidable. Most importantly, Theorem 4.4.10 asserts the existence of (computational) zero-knowledge proof systems for $\mathcal{NP}$, assuming that one-way functions exist. Two natural questions arise

1. *An unconditional result*: 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?

The answer to both question seems to be negative.

Another important question concerning zero-knowledge proofs is their preservation under parallel composition. We 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 leak knowledge in a strong sense). Furthermore, we consider some natural proof systems, obtained via parallel composition of zero-knowledge proofs, and indicate that it is unlikely that the resulting composed proofs can be proven to be zero-knowledge.

4.5.1 Implausibility of an Unconditional "NP in ZK" Result

Recall that Theorem 4.4.11 asserts the existence of zero-knowledge proofs for any languages in $\mathcal{IP}$, *provided that nonuniform one-way functions exist.* In this subsection we consider the question of whether this sufficient condition is also necessary. The results, known to date, seem to provide some (yet, weak) indication in this direction. Specifically, the existence of zero-knowledge proof systems for languages out of $\mathcal{BPP}$ implies very weak forms of one-wayness. Also, the existence of zero-knowledge proof systems for languages which are hard to approximate, in some average case sense, implies the existence of one-way functions (but not of nonuniformly one-way functions). In the rest of this subsection we provide precise statements of the above results.

4.5.1.1 $\mathcal{BPP} \subset \mathcal{CZK}$ implies weak forms of one-wayness

Definition 4.5.1 (collection of functions with one-way instances): *A collection of functions, $\{f_i : D_i \mapsto \{0,1\}^*\}_{i\in\overline{I}}$, is said to* have one-way instances *if there exists three probabilistic polynomial-time algorithms, I, D and F, so that the following two conditions hold*

1. easy to sample and compute: *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's*

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

where X_n is a random variable describing the output of algorithm D on input i.

Actually, since the hardness condition does not refer to the distribution induced by I, we may assume, without loss of generality, that $\overline{I} = \{0,1\}^*$ and algorithm I uniformly selects a string (of length equal to the length of its input). Recall that collections of one-way functions (as defined in Definition 2.4.3) requires hardness to invert of all but a negligible measure of the functions f_i (where the probability measure is induced by algorithm I).

Theorem 4.5.2 *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 one-wayness. The existence of a language in $\mathcal{NP}$ which is not in $\mathcal{BPP}$ implies the existence of a function which is easy to compute but hard to invert in the worst-case (see Section 2.1). The latter consequence seems to be a much weaker form of one-wayness.

4.5.1.2 Zero-knowledge for "hard" languages yield one-way functions

Our notion of hard languages is the following

Definition 4.5.3 *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 χ_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).

Theorem 4.5.4 *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).

4.5.2 Implausibility of Perfect Zero-Knowledge proofs for all of NP

A theorem bounding the class of languages possessing *perfect* zero-knowledge proof systems follows. We start with some background (for more details see Section [missing(eff-ip.sec)]). By $\mathcal{AM}$ we denote the class of languages having an interactive proof which proceeds as follows. First the verifier sends a random string to the prover, next the prover answers with some string, and finally the verifier decided whether to accept or reject based on a deterministic computation (depending on the common input and the above two strings). The class $\mathcal{AM}$ seems to be a randomized counterpart of $\mathcal{NP}$, and it is believed that $co\mathcal{NP}$ is not contained in $\mathcal{AM}$. Additional support to this belief is given by the fact that $co\mathcal{NP} \subseteq \mathcal{AM}$ implies the collapse of the Polynomial-Time Hierarchy. In any case it is known that

Theorem 4.5.5 *The class of languages possessing perfect zero-knowledge proof systems is contained in the class co$\mathcal{AM}$. (In fact, these languages are also in $\mathcal{AM}$.)*

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

We stress that Theorem 4.5.5 *does not* apply to perfect zero-knowledge arguments, defined and discussed in Section 4.8. Hence, there is no conflict between Theorem 4.5.5 and the fact that, under some reasonable complexity assumptions, perfect zero-knowledge arguments do exist for every language in $\mathcal{NP}$.

4.5.3 Zero-Knowledge and Parallel Composition

We discuss two negative results of very different conceptual standing. The first result asserts the failure of the general "Parallel Composition Conjecture", but says nothing about specific natural candidates. The second result refers to a class of interactive proofs, which contains several interesting and natural examples, and assert that the members of this class cannot be proven zero-knowledge using a general paradigm (know by the name "black box simulation"). We mention that it is hard to conceive an alternative way of demonstrating the zero-knowledge property of protocols (rather than by following this paradigm).

4.5.3.1 Failure of the Parallel Composition Conjecture

For some time, after zero-knowledge proofs were first introduced, several 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 resulting by running both of them in parallel is also zero-knowledge.*

Some researchers even considered the failure to prove the Parallel Composition Conjecture as a sign of incompetence. However, the Parallel Composition Conjecture is just wrong.

Proposition 4.5.6 *There exists two provers, P_1 and P_2, such that each is zero-knowledge, and yet the prover resulting by running both of them in parallel yields knowledge (e.g., a cheating verifier may extract from this prover a solution to a problem that is not solvable in polynomial-time). Furthermore, the above holds even if the zero-knowledge property of each of the P_i's can be demonstrated using a simulator which uses the verifier as a black-box (see below).*

We remark that these provers can be incorporated into a single prover that randomly selects which of the two programs to execute. Alternatively, the choice may be determined by the verifier.

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

The above proposition refutes the Parallel Composition Conjecture by means of exponential time provers. Assuming the existence of one-way functions the Parallel Composition Conjecture can be refuted also 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 σ. (To this end the prover is give as auxiliary input the coins used in the commitment.) On input $C(1^n, \sigma)$, prover P_2 asks the verifier to prove that it knows σ and if P_2 is convinced then it sends σ to the verifier. This verifier employs the same system of proofs of knowledge used by the program P_1. Clearly, each prover is zero-knowledge and yet their parallel composition is not. Similarly, using stronger intractability assumptions, one can refute the Parallel Composition Conjecture also with respect to perfect zero-knowledge (rather than with respect to computational zero-knowledge).

4.5.3.2 Problems 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 which uses any prospective verifier V^* as a black-box. In fact, these demonstrations use as 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.

Definition 4.5.7 (black-box zero-knowledge):

- next message function: *Let B be an interactive turing machine, and x, z, r be strings representing a common-input, auxiliary-input, and 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 $z, r \in \{0,1\}^*$:*

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

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 23. 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 4.4.12)).

Theorem 4.5.8 *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 $L \in \mathcal{BPP}$.

We remark that both Construction 4.3.6 (zero-knowledge proof for Graph Isomorphism) and Construction 4.4.6 (zero-knowledge proof for Graph Coloring) are constant round, use public coins and are black-box zero-knowledge (for the corresponding languages). 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. Clearly the resulting proof system are constant round and use public coins. Hence, unless the corresponding languages are in $\mathcal{BPP}$, these parallelized proof systems *are not* black-box zero-knowledge.

Theorem 4.5.8 is sometimes interpreted as pointing to an inherent limitation of interactive proofs with public coins (also known as *Arthur Merlin* games; see Section `[missing(eff-ip.sec)]`). 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 is advised to use "private coins"* (i.e., to let the verifier send messages depending upon, but not revealing its coin tosses).

4.6 * Witness Indistinguishability and Hiding

In light of the non-closure of zero-knowledge under parallel composition, see Subsection 4.5.3, alternative "privacy" criteria that are preserved under parallel composition are of practical and theoretical importance. Two notions, called witness indistinguishability and witness hiding, which 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.

4.6.1 Definitions

In this section we confine ourself to 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 $|y| \leq \text{poly}(|x|)$), polynomial-time recognizable, and characterizes L by

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

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 will relax the requirement so that it applies only to the case in which the auxiliary input constitutes an NP-witness to the common input; namely, let R_L be the witness relation of the language L and suppose that $x \in L$, then we consider only auxiliary inputs in $R_L(x) \stackrel{\text{def}}{=} \{y; (x,y) \in R_L\}$. By saying that the view is computational independent of the witness we mean that for every two choices of auxiliary inputs the resulting views are computationally indistinguishable. In the actual definition we combine notations and conventions from Definitions 4.3.3 and 4.3.8.

Definition 4.6.1 (witness indistinguishability / independence): *Let (P,V), $L \in \mathcal{NP}$ and V^* be as in Definition 4.3.8, and let R_L be a fixed witness relation for the language L. We denote by $\text{view}_{V^*(z)}^{P(y)}(x)$ a random variable describing the contents of the random-tape of V^* and the messages V^* receives from P during a joint computation on common input x, when P has auxiliary input y and V^* has auxiliary input z. 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 $W^2 = \{w_x^2\}_{x\in L}$, so that $w_x^1, w_x^2 \in R_L(x)$, the following two ensembles are computationally indistinguishable*

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

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

$$|\mathsf{Pr}(D(x, \text{view}_{V^*(z)}^{P(w_x^1)}(x))=1) - \mathsf{Pr}(D(x, \text{view}_{V^*(z)}^{P(w_x^2)}(x))=1)| < \frac{1}{p(|x|)}$$

We say that (P,V) is witness independent *if the above ensembles are identically distributed. Namely, for every $x \in L$ every $w_x^1, w_x^2 \in R(x)$ and $z \in \{0,1\}^*$, the random variables $\text{view}_{V^*(z)}^{P(w_x^1)}(x)$ and $\text{view}_{V^*(z)}^{P(w_x^2)}(x)$ are identically distributed.*

A few remarks are in place. First, one may observe that any proof system in which the prover ignores its auxiliary-input is trivially witness independent. In particular, exponential-time provers may, without loss of generality, ignore their auxiliary-input (without any decrease in the probability that they convince the verifier). Yet, probabilistic polynomial-time provers can not afford to ignore their auxiliary input (since otherwise they become useless). Hence, for probabilistic polynomial-time provers for languages outside $\mathcal{BPP}$, witness indistinguishability is non-trivial. Secondly, one can easily show that any zero-knowledge proof system for a language in $\mathcal{NP}$ is witness indistinguishable (since the view corresponding to each witness can be approximated by the same simulator). Likewise, perfect zero-knowledge proofs are witness independent. Finally, 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.

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 NP witness for the common input. Clearly, such a requirement can hold only if it is infeasible to find witnesses from scratch. Since, 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 4.6.2 (distribution of hard instances): *Let $L \in \mathcal{NP}$ and R_L be a witness relation for L. Let $X \stackrel{\text{def}}{=} \{X_n\}_{n\in\mathbb{N}}$ be a probability ensemble so that X_n assign non-zero probability mass only to strings in $L \cap \{0,1\}^n$. 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 $z \in \{0,1\}^{\text{poly}(n)}$*

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

Definition 4.6.3 (witness hiding): *Let (P, V), $L \in \mathcal{NP}$, and R_L be as in the above 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 $p(\cdot)$ and 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 say that (P, V) is* universal witness hiding for the relation R_L *if the proof system (P, V) is witness hiding for R_L under every ensemble of hard instances, for R_L, that is efficiently constructible (see Definition 3.2.5)*

We remark that the relation between the two privacy criteria (i.e., witness indistinguishable and witness hiding) is not obvious. Yet, zero-knowledge proofs (for $\mathcal{NP}$) are also (universal) witness hiding (for any corresponding witness relation). We remark that witness indistinguishability and witness hiding, similarly to zero-knowledge, are properties of the prover (and more generally of a any interactive machine).

4.6.2 Parallel Composition

In contrary 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 4.6.4 (Parallel Composition Lemma): *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 P_Q denote a program that on common-input $x_1, ..., x_{Q(n)} \in \{0,1\}^n$ and auxiliary-input $w_1, ..., w_{Q(n)} \in \{0,1\}^*$, 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 $\overline{x} = (x_1, ..., x_m)$, and $\overline{w} = (w_1, ..., w_m)$, 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 $\overline{w}^1 = (w_1^1, ..., w_m^1)$ and $\overline{w}^2 = (w_1^2, ..., w_m^2)$, used by P_Q, in an interaction on common-input $\overline{x} \in L^m$. Then, for some i, the program V_Q^* distinguishes also the hybrid witnesses $\overline{h}^{(i)} = (w_1^1, ..., w_i^1, w_{i+1}^2, ..., w_m^2)$ and $\overline{h}^{(i+1)} = (w_1^1, ..., w_{i+1}^1, w_{i+2}^2, ..., w_m^2)$. Rewrite $\overline{h}^{(i)} = (w_1, ..., w_i, w_{i+1}^2, w_{i+2}, ..., w_m)$ and $\overline{h}^{(i+1)} = (w_1, ..., w_i, w_{i+1}^1, w_{i+2}, ..., w_m)$. 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 prover P in parallel to simulating $m-1$ other interactions with P. The real interaction with P is viewed as the $i+1^{\text{st}}$ copy in an interaction of V_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 $\overline{x}$, $\overline{h}^{(i)}$ and $\overline{h}^{(i+1)}$ 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 the auxiliary-input to the j^{th} copy of P. Machine V^* invokes V_Q^* on common input $\overline{x}$ and provides it with an interface to a virtual interaction with P_Q. The $i+1^{\text{st}}$ component of a message $\overline{\alpha} = (\alpha_1, ..., \alpha_m)$ 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 β, machine V^* computes the answers of the other copies of P (by feeding the program P with the corresponding auxiliary-input and the corresponding sequence of incoming messages). It follows, that V^* can distinguish the case P uses the witness w_{i+1}^1 from the case P uses w_{i+1}^2.
■

4.6.3 Constructions

In this subsection 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 and the observation that zero-knowledge proofs are witness indistinguishable we derive the following

Theorem 4.6.5 *Assuming the existence of (nonuniformly) one-way functions, every language in $\mathcal{NP}$ has a* constant-round *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 system. Namely, the known proof systems for $\mathcal{NP}$ are either

- not constant-round (e.g., Construction 4.4.8); or
- have non-negligible error probability (e.g., Construction 4.4.6); or
- require stronger intractability assumptions (see Subsection 4.9.1); or
- are only computationally sound (see Subsection 4.9.2).

Similarly, we can derive a constant-round witness independent proof system, with exponentially small error probability, for Graph Isomorphism. (Again, no analogous result is known for perfect zero-knowledge proofs.)

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 which yields the unique witness, and yet is trivially witness independent. On the other hand, for some relations, witness indistinguishability implies witness hiding. For example

Proposition 4.6.6 *Let $\{(f_i^0, f_i^1) : i \in \overline{I}\}$ be a collection of (nonuniform) clawfree functions, and let*

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

Then if a machine P is witness indistinguishable for R then it 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 nonuniform clawfree functions we mean that even nonuniform families of circuits $\{C_n\}$ fail to form claws on input distribution $I(1^n)$, except with negligible probability. We remark that the above 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). The proposition is proven by contradiction. Details follow.

Suppose that an interactive machine V^* finds witnesses after interacting with P. By the witness indistinguishability of P it follows that V^* is performing as well regardless on whether the witness is of the form $(0,\cdot)$ or $(1,\cdot)$. Combining the programs V^* and P with algorithm D we derive a claw forming algorithm (and hence 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 simulates an interaction of V^* with P on common-input x and auxiliary-input (σ, r) to P. If 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 $f_i^\sigma(r) = f_i^{1-\sigma}(r')$).□

Furthermore, every 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

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

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 4.6.7 *Let R and R_2 be as above. If a machine P is witness indistinguishable for R_2 then it is also witness hiding for R_2 under every distribution of hard instances induced (see below) by an efficient algorithm that randomly selects pairs in R.*

Let S be a probabilistic polynomial-time algorithm that on input 1^n outputs $(x,w) \in R$ so that $|x| = n$. Let X_n denotes the distribution induced on the first element in the output of $S(1^n)$. The proposition asserts that if P is witness indistinguishable and $\{X_n\}_{n\in\mathbb{N}}$ an ensemble of hard instances for R then P is witness hiding under the ensemble $\{\overline{X}_n\}_{n\in\mathbb{N}}$ where $\overline{X}_n$ consists of two independent copies of X_n. This assertion is proven by contradiction. Suppose that an interactive machine V^* finds witnesses after interacting with P. By the witness indistinguishability of P it follows that V^* is performing as well regardless on whether the witness w for (x_1, x_2) satisfies either $(x_1, w) \in R$ or $(x_2, w) \in R$. Combining the programs V^* and P with algorithm S we derive a 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^* simulates 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 $(x, w) \in R$. The reader can easily verifier that algorithm F^* performs well under the instance ensemble $\{X_n\}$, hence contradicting the hypothesis that X_n is hard for R. □

4.6.4 Applications

Applications for the notions presented in this section are scattered in various places in the book. In particular, witness-indistinguishable proof systems are used in the construction of constant-round arguments for $\mathcal{NP}$ (see Subsection 4.9.2), witness independent proof systems are used in the zero-knowledge proof for Graph Non-Isomorphism (see Section 4.7), and witness hiding proof systems are used for the efficient identification scheme based on factoring (in Section 4.7).

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 imply 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 in addressing 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 a special subsection. Of special interest is the application to identification schemes, which is discussed in a separate subsection. Finally, in the last subsection, we introduce the notion of strong proofs of knowledge.

4.7.1 Definition

We start with a motivating discussion.

> What does it mean to say that a machine knows something? Any standard dictionary suggests several meanings to the verb *know* and most meanings are phrased with reference to "awareness". We, however, must look for a behavioristic interpretation of the verb. 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 will say that a machine knows a string α if it *can* output the string α. This seems as total nonsense. A machine has a well defined output: either the output equals α or it does not. So what can be meant by saying that a machine can do something. Loosely speaking, it means that the machine can be modified so that it does whatever is claimed. More precisely, it means that there exists an *efficient* machine which, using the original machine as oracle, outputs whatever is claimed.

So far for defining the "knowledge of machines". Yet, whatever a machine knows or does not know is "its own business". What can be of interest 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 sake of simplicity let us consider a concrete question: how can a machine prove that it knows a 3-coloring of a graph? An obvious way is just to send the 3-coloring to the verifier. Yet, we claim that applying Construction 4.4.6 (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 messages sent by P (in response to particular messages that P receives). The (expected) running time of the extractor, on input G and access to an oracle specifying P's messages, is 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 non-negligible probability. We stress that the latter special cases do not suffice for a satisfactory definition.

4.7.1.1 Preliminaries

Let $R \subseteq \{0,1\}^* \times \{0,1\}^*$ be a binary relation. Then $R(x) \stackrel{\rm def}{=} \{s : (x,s) \in R\}$ and $L_R \stackrel{\rm def}{=} \{x : \exists s \text{ s.t. } (x,s) \in R\}$. 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 *NP relation* if R is polynomially bounded and, in addition, there exists a polynomial-time algorithm for deciding membership in R (i.e., $L_R \in \mathcal{NP}$). In the sequel, we confine ourselves to polynomially bounded relations.

We wish to be able to consider in a uniform manner all potential provers, without making distinction based on their running-time, internal structure, etc. Yet, we observe that these interactive machine can be given an auxiliary-input which enables them to "know" and to prove more. Likewise, they may be luck to select a random-input which enables more 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 has. Hence, we fix an interactive machine and all inputs (i.e., the common-input, the auxiliary-input, and the random-input) to this machine, and consider both the corresponding accepting probability (of the verifier) and the usage of this (prover+inputs) 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 is inversely related to the corresponding accepting probability (of the verifier).

4.7.1.2 Knowledge verifiers

Now that all the machinery is ready, we present the definition of a system for proofs of knowledge. Actually, the definition presented below is a generalization (to be motivated by the subsequent applications). At first reading, the reader may set the function κ to be identically zero.

Definition 4.7.2 (System of 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 κ *if the following two conditions hold.*

- Non-triviality: *There exists an interactive machine P so 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 κ): *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 $p(x)$ the probability that the interactive machine V accepts, on input x, when interacting with the prover specified by $P_{x,y,r}$. Then if $p(x) > \kappa(|x|)$ then, on input x and access to oracle $P_{x,y,r}$, machine K outputs a solution $s \in R(x)$ within an expected number of steps bounded by

$$\frac{q(|x|)}{p(x) - \kappa(|x|)} .$$

  The oracle machine K is called a universal knowledge extractor.

When $\kappa(\cdot)$ is identically zero, we just say that V is a knowledge verifier for the relation R*. An interactive pair (P,V) so 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.*

4.7.2 Observations

The zero-knowledge proof systems for Graph Isomorphism (i.e., Construction 4.3.6) and for Graph 3-Coloring (i.e., Construction 4.4.6) are in fact proofs of knowledge (with some knowledge error) for the corresponding languages. Specifically, Construction 4.3.6 is a proof of knowledge of an isomorphism with knowledge error $\frac{1}{2}$, whereas Construction 4.4.6 is a proof of knowledge of a 3-coloring with knowledge error $1 - \frac{1}{|E|}$ (on common input $G=(V,E)$). By iterating each construction sufficiently many times we can get the knowledge error to be exponentially small. (The proofs of all these claims are left as an exercise.) In fact, we get a proof of knowledge with zero error, since

Proposition 4.7.3 *Let R be an NP relation, and $q(\cdot)$ 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 $\kappa(n) \stackrel{\rm def}{=} 2^{-q(n)}$. Then (P,V) is a system for proofs of knowledge for the relation R (with zero knowledge error).*

Proof Sketch: Given a knowledge extractor, K, substantiating the hypothesis, we construct a new knowledge extractor which runs K in parallel to conducting an exhaustive search for a solution. Let $p(x)$ be as in Definition 4.7.2. To evaluate the performance of the new extractor consider two cases.

Case 1: $p(x) \geq 2 \cdot \kappa(|x|)$. In this case, we use the fact

$$\frac{1}{p(x) - \kappa(|x|)} \leq \frac{2}{p(x)}$$

Case 2: $p(x) \leq 2 \cdot \kappa(|x|)$. In this case, we use the fact that exhaustive search of a solution boils down to $2^{q(|x|)}$ trials, whereas $\frac{1}{p(x)} \geq \frac{1}{2} \cdot 2^{q(|x|)}$. ■

It follows that

Theorem 4.7.4 *Assuming the existence of (nonuniformly) one-way function, every NP relation has a zero-knowledge system for proofs of knowledge.*

4.7.3 Applications

We briefly review some of the applications for (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) but are each doubtful of the other person's claim. Employing a zero-knowledge proof of knowledge in both direction is indeed a (conceptually) simple solution to the problem of convincing each other of their knowledge.

4.7.3.1 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 it is committed. Such a guarantee may be useful in many settings, and can be obtained by using proof of knowledge. For more details see Subsection 4.9.2.

4.7.3.2 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. (For definition and alternative constructions of such schemes see Section [missing(enc-strong.sec)].) However, one should note that the resulting encryption scheme employs bidirectional communication between the sender and the receiver (of the encrypted message). It seems that the use of *non-interactive* zero-knowledge proofs of knowledge would yield unidirectional (public-key) encryption schemes. Such claims have been made, yet no proof has ever appeared (and we refrain from expressing an opinion on the issue). Non-interactive zero-knowledge proofs are discussed in Section 4.10.

4.7.3.3 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 may construct a graph H and learn whether it is isomorphic to the first input graph by sending H as query to the prover. A more appealing refutation can be presented to the claim that Construction 4.2.8 is auxiliary-input zero-knowledge (e.g., the verifier can check whether 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 always knew the answer to its queries (as is the case for the honest verifier). The idea then is to have the verifier prove to the prover that he (i.e., the verifier) knows the answer to the query (i.e., an isomorphism to the appropriate input graph), and the prover answers the query only if it is convinced of this claim. Certainly, the verifier's proof of knowledge should not yield the answer (otherwise the prover can use this information in order to cheat thus foiling the soundness requirement). If the verifier's proof of knowledge is zero-knowledge then certainly it does not yield the answer. In fact, it suffices that the verifier's proof of knowledge is *witness-independent* (see Section 4.6).

4.7.4 Proofs of Identity (Identification schemes)

Identification schemes are useful in large distributed systems in which the users are not acquainted with one another. A typical, everyday example is the consumer-retailer situation. In computer systems, a typical example is electronic mail (in communication networks containing sites allowing too loose local super-user access). In between, in technological sophistication, are the Automatic Teller Machine (ATM) system. In these distributed systems, one wishes to allow users to be able to authenticate themselves to other users. This goal is achieved by identification schemes, defined below. In the sequel, we shall also see that identification schemes are intimately related to proofs of knowledge. We just 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.

4.7.4.1 Definition

Loosely speaking, an identification scheme consists of a *public file* containing *records* for each user and an *identification protocol*. Each record consists of the name (or identity) of a user and auxiliary *identification information* to be used when invoking the identification protocol (as discussed below). The public file is established and maintained by a trusted party which vouches for the authenticity of the records (i.e., that each record has been submitted by the user the name of which is specified in it). All users have read access to 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 her 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 executing his role in the protocol. It is required that `Alice` will 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 as `Alice` even after receiving polynomially many proofs of identity from `Alice`.

Clearly, if the identification information is to be of any use, then `Alice` must keep in secret the random coins she has 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 which does not allow her counterparts to later impersonate her.

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 which 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.

Remark: In first reading, the reader may ignore algorithm A and the random variable T_n in the security condition. Doing so, however, yields a weaker condition, that is typically unsatisfactory.

Definition 4.7.5 (identification scheme): *An* identification scheme *consists of a pair,* (I, Π), *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* $s \in \{0,1\}^{\mathrm{poly}(n)}$

$$\mathsf{Pr}\left(\langle P(s), V\rangle(\alpha, I_s(\alpha)) = 1\right) = 1$$

- Security: *For every pair of probabilistic polynomial-time interactive machines,* A *and*

B, every polynomial $p(\cdot)$, all sufficiently large $n \in \mathbb{N}$, every $\alpha \in \{0,1\}^n$, and every z

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

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 $A(z)$ after interacting with $P(S_n)$ on common input α, 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`, the identity of which is encoded by the string α, should first uniformly select a secret string s, compute $i \stackrel{\mathrm{def}}{=} I_s(\alpha)$, ask the trusted party to place the record (α, 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: `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 α (which is in the public file). Ignoring, for a moment, algorithm A and the random variable T_n, the security condition yields that it is infeasible for a party to impersonate `Alice` if all this party has is the public record of `Alice` and some unrelated auxiliary input. However, such a security condition may not suffice in many applications since a user wishing to impersonate `Alice` may ask her first to prover her identity to him/her. The (full) security condition asserts that even if `Alice` has proven 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. In case this condition does not hold then nothing is guaranteed (and indeed `Carol` can easily cheat by referring `Bob`'s questions to `Alice` and answering as `Alice` does).

4.7.4.2 Identification schemes and proofs of knowledge

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

Construction 4.7.6 (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 with name α). *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 α proves that he knows a string s so that $f(s) = r$, where (α, r) is a record in the public file. (The proof of knowledge in used is allowed to have negligible knowledge error.)*

Proposition 4.7.7 *If f is a one-way function and the proof of knowledge in use is zero-knowledge then Construction 4.7.6 constitutes an identification scheme.*

Hence, identification schemes exist if one-way functions exist. More efficient 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 square root, follows.

Construction 4.7.8 (the Fiat-Shamir identification scheme): *On input an identity $\alpha \in \{0,1\}^n$, the* information generating algorithm *uniformly selects a composite number N, which is the product of two n-bit long primes, a residue s mod N, and outputs the pair $(N, s^2 \bmod N)$.* (The pair $(\alpha, (N, s^2 \bmod N))$ is the public record for user α). *The* identification protocol *consists of a proof of knowledge of the corresponding modular square root. Namely, in order to prove its identity, user α proves that he knows a 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 square root is analogous to the proof system for Graph Isomorphism presented in Construction 4.3.6. 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 4.7.9 (atomic proof of knowledge of square root):

- *The prover randomly selects a residue, q, modulo N and send $t \stackrel{\text{def}}{=} q^2 \bmod N$ 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 $t \bmod N$, whereas in case $\sigma = 1$ the verifier asks for a square root of $t \cdot r \bmod N$. In the sequel we assume, without loss of generality, that $\sigma \in \{0,1\}$.*
- *The prover replies with $p \stackrel{\text{def}}{=} q \cdot s^\sigma \bmod N$;*
- *The verifier accepts (this time) if and only if the messages t and p sent by the prover satisfies $p^2 \equiv t \cdot r^\sigma \bmod N$;*

When Construction 4.7.9 is repeated k times, either sequentially or in parallel, the resulting protocol constitutes a proof of knowledge of modular square root with knowledge error 2^{-k}. In case these repetitions are conducted sequentially, then the resulting protocol is zero-knowledge. Yet, for use in Construction 4.7.8 it suffices that the proof of knowledge is *witness-hiding*, and fortunately even polynomially many parallel executions can be shown to be witness-hiding (see Section 4.6). Hence the resulting identification scheme has constant round complexity. We remark that for identification purposes it suffices to perform Construction 4.7.9 superlogarithmically many times. Furthermore, also less repetitions are of value: when applying Construction 4.7.9 $k = O(\log n)$ times, and using the resulting protocol in Construction 4.7.8, we get a scheme (for identification) in which impersonation can occur with probability at most 2^{-k}.

4.7.4.3 Identification schemes and proofs of ability

As hinted above, 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 be based also on the more general concept of proofs of ability. We avoid defining this concept, and refrain ourself to two "natural" examples of using a proof of ability as basis for identification.

It is an everyday practice to identify people by their ability to produce their signature. 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 correctly personal questions. A digital analogue to this practice follows. To this end 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 this 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 as the user. Here we replaced the possession of the correct answers by a zero-knowledge proof that the answer is correct.

4.7.5 Strong Proofs of Knowledge

Definition 4.7.2 relies in a fundamental way on the notion of *expected* running-time. We thus prefer the following more stingent definition in which the knowledge extractor is required to run in *strict* polynomial-time (rather than in *expected* polynomial-time). (We also take the oppertumity to postulate – in the definition – that no-instances are accepted with negligible probability; this is done by extending the scope of the validity condition also to x's not in L_R.)

Definition 4.7.10 (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 $\mu : \mathbb{N} \mapsto [0,1]$ 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 $p(x)$ and $P_{x,y,r}$ be as in Definition 4.7.2. If $p(x) > \mu(|x|)$ then, on input x and access to oracle $P_{x,y,r}$, with probability at least $1 - \mu(|x|)$, machine K outputs a solution $s \in R(x)$.

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

An interactive pair (P, V) so 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.*

Sequentially repeating the (zero-knowledge) proof systems for Graph Isomorphism (i.e., Construction 4.3.6), 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 4.3.6), results in one of two possible situations depending on whether the verifier asks to see an isomorphism to the first or second graph. In case the prover answers correctly in both cases, we can retreive an isomorphism between the input graphs (by composing the isomorphisms provided in the two cases). In case the prover fails in both cases, the verifier will reject regradless of what the prover does from this point on. Specifically, the above 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 procced in n iterantions, starting with $i = 1$. Initially, T (the trascript so far), is empty.

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

4. In case ψ_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, halt with no output.

It can be easily verified that if the extractor halts with no output in 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 2^{-n}. Thus, whenever $p(G_1, G_2) > 2^{-n}$, the extractor succeeds in recovering an isomporphism between the two input graphs. A similar argument may be applied to some zero-knowledge proof systems for NP. In particular, consider n sequential repetitions of the following basic proof system for the *Hamiltonian Cycle* (HC) problem which is NP-complete (and thus get proof systems for any language in $\mathcal{NP}$). We consider directed graphs (and the existence of directed Hamiltonian cycles).

Construction 4.7.11 (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,* π*, 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 so that the* $(\pi(i), \pi(j))^{\text{th}}$ *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.*
- Prover's second step (P1): *If* $\sigma = 0$ *then the prover sends* π *to the verifier along with the revealing (i.e., preimages) 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 (V1): *If* $\sigma = 0$ *then the verifier checks that the revealed graph is indeed isomorphic, via* π*, to* G*. Otherwise, the verifier just 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 mention that the known (zero-knowledge) strong proofs of knowledge are all costy in round-complexity. Still, we have

Theorem 4.7.12 *Assuming the existence of (nonuniformly) one-way function, every 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 is *impossible* to fool the verifier into accepting false statement (with probability greater than some bound), we only require that it is *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 $\mathcal{NP}$. We remark that *perfect* zero-knowledge proof systems are unlikely to exists for all languages in $\mathcal{NP}$ (see section 4.5). We recall that *computational* zero-knowledge proof systems do exist for all languages in $\mathcal{NP}$, provided that one-way functions exist. Hence, the above 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 this is not a real trade-off since the perfect zero-knowledge computationally sound proofs for $\mathcal{NP}$ are constructed under stronger complexity theoretic assumption than the ones used for the computationally 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 (and in particular assuming only the existence of one-way functions).

We remark 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 above discussion. The only issue to consider is that merely replacing the soundness condition of Definition 4.2.4 by the following *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.

> *Computational Soundness:* For every polynomial-time interactive machine B, and for all sufficiently long $x \notin L$
>
> $$\mathsf{Pr}\left(\langle B, V\rangle(x) = 1\right) \leq \frac{1}{3}$$

Hence, it is natural to restrict the prover in both (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 an* computationally sound proof system 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*

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

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

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

As usual, the error probability in the completeness condition can be reduced (from $\frac{1}{3}$) up to $2^{-\mathrm{poly}(|x|)}$, by repeating the protocol sufficiently many times. By *sequentially* repeating a computationally sound proof system sufficiently many time (i.e., superlogarithmically many times) we get a new verifier V' for which it holds that

> For every polynomial $p(\cdot)$, every polynomial-time interactive machine B, and for all sufficiently long $x \notin L$ and every y and z
>
> $$\mathsf{Pr}\left(\langle B(y), V'(z)\rangle(x)=1\right) \leq \frac{1}{p(|x|)}$$

See Exercise 21. We note that error reduction via *parallel* repetitions is **not** known to work *in general.*

4.8.2 Perfect Commitment Schemes

The thrust of the current section is in a method for constructing *perfect* zero-knowledge *arguments* for every language in $\mathcal{NP}$. This method makes essential use of the concept of commitment schemes with a *perfect* (or "information theoretic") secrecy property. Hence, we start with an exposition of "perfect" commitment schemes. We remark that such schemes may be useful also in other settings (e.g., in settings in which the receiver of the commitment is computationally unbounded, see for example Section 4.9).

The difference between commitment scheme (as defined in Subsection 4.4.1) and perfect commitment schemes (defined below) consists of a switching in 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 perfect commitment schemes (see definition below), the secrecy requirement is information theoretic, whereas the unambiguity requirement is computational (i.e., refers only to probabilistic polynomial-time adversaries). Hence, in some sense calling one of these schemes "perfect" is somewhat unfair to the other (yet, we do so in order to avoid cumbersome terms as a "perfectly-secret/computationally-nonambiguous commitment scheme"). We remark that it is impossible to have a commitment scheme in which both the secrecy and unambiguity requirements are information theoretic (see Exercise 22).

4.8.2.1 Definition

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

1. *Secrecy*: At the end of the *commit phase* the *receiver* does not gain any *information* of the sender's value.
2. *Unambiguity*: It is infeasible for the sender to interact with the receiver so that 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 analogous conventions to the ones used in Subsection 4.4.1, we make the following definition.

Definition 4.8.2 (perfect bit commitment scheme)*: A* perfect bit commitment scheme *is a pair of probabilistic polynomial-time interactive machines, denoted (S, R) (for* sender *and* receiver*), satisfying:*

- 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 v.*
- Secrecy: *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 statistically close.*
- Unambiguity:
  Preliminaries. For simplicity $v \in \{0,1\}$ and $n \in \mathbb{N}$ are implicit in all notations. Fix any probabilistic polynomial-time algorithm F^*.
  - *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 (r) and the sequence of messages received from the sender ($\overline{m}$). A* sender's view *of the same interaction, denoted $(s, \tilde{m})$, consists of the random coins used by the sender (s) and the sequence of messages received from the receiver ($\tilde{m}$). A* joint view *of the interaction is a pair consisting of corresponding receiver and sender views of the same interaction.*

- *Let $\sigma \in \{0,1\}$. We say that a joint view (of an interaction), $((r,\overline{m}),(s,\tilde{m}))$, has* a feasible σ-opening (with respect to F^*) *if on input $(\overline{m}, s, \tilde{m}, \sigma)$, algorithm F^* outputs (say, with probability $> 1/2$) a string s' such that $\overline{m}$ describes the messages received by R when R uses local coins r and interacts with machine S which uses local coins s' and input $(\sigma, 1^n)$.*
  (Remark: We stress that s' may, but need not, equal s. The output of algorithm F^* has to satisfy a relation which depends only on the receiver's view part of the input; the sender's view is supplied to algorithm F^* as additional help.)
- *We say that a joint view is* ambiguous (with respect to F^*) *if it has both a feasible 0-opening and a feasible 1-opening* (w.r.t. F^*).

The unambiguity requirement *asserts that, for all but a negligible fraction of the coin tosses of the receiver, it is infeasible for the sender to interact with the receiver so that the resulting joint view is ambiguous with respect to some probabilistic polynomial-time algorithm F^*. Namely, for every probabilistic polynomial time interactive machine S^*, probabilistic polynomial-time algorithm F^*, polynomial $p(\cdot)$, and all sufficiently large n, the probability that the joint view of the interaction between R and with S^*, on common input 1^n, is ambiguous with respect to F^*, is at most $1/p(n)$.*

In the formulation of the unambiguity requirement, S^* describes the (cheating) sender strategy in the commit phase, whereas F^* describes its strategy in the reveal phase. Hence, it is justified (and in fact necessary) to pass the sender's view of the interaction (between S^* and R) to algorithm F^*. The unambiguity requirement asserts that any efficient strategy S^* will fail to produce a joint view of interaction, which can be latter (efficiently) opened in two different ways supporting two different values. As usual, events occurring with negligible probability are ignored.

As in Definition 4.4.1, the secrecy requirement refers explicitly to the situation at the end of the commit phase, whereas 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).

4.8.2.2 Construction based on one-way permutations

Perfect commitment schemes can be constructed using any one-way permutation. The known scheme, however, involve a linear (in the security parameter) number of rounds. Hence, it can be used for the purposes of the current section, but not for the construction in Section 4.9.

Construction 4.8.3 (perfect bit commitment): *Let f be a permutation, and $b(x,y)$ denote the inner-product mod 2 of x and y (i.e., $b(x,y) = \sum_{i=1}^{n} x_i y_i \bmod 2$).*

1. commit phase *(using security parameter n):*
   - *The receiver randomly selects $n-1$ linearly indepndent vectors $r^1, ..., r^{n-1} \in \{0,1\}^n$. The sender uniformly selects $s \in \{0,1\}^n$ and computes $y = f(s)$. (So far no message is exchanged between the parties.)*
   - *The parties proceed in $n-1$ rounds. In the i^{th} round ($i = 1, ..., n-1$), the receiver sends r^i to the sender, which replies by computing and sending $c^i \stackrel{\text{def}}{=} b(y, r^i)$.*
   - *At this point there are exactly two solutions to the equations $b(y, r^i) = c^i$, $1 \le i \le n-1$. Define $j = 0$ if y is the lexicographically first solution (among the two), and $j = 1$ otherwise. To commit to a value $v \in \{0,1\}$, the sender sends $c^n \stackrel{\text{def}}{=} j \oplus v$ to the receiver.*
2. reveal phase: *In the reveal phase, the sender reveals the string s used in the commit phase. The receiver accepts the value v if $f(s) = y$, $b(y, r^i) = c^i$ for all $1 \le i \le n-1$, and y is the lexicographically first solution to these $n-1$ equations iff $c^n = v$.*

Proposition 4.8.4 *Suppose that f is a one-way permutation. Then, the protocol presented in Construction 4.8.3 constitutes a perfect bit commitment scheme.*

It is quite easy to see that Construction 4.8.3 satisfies the secrecy condition. The proof that the unambiguity requirement is satisfied is quite complex and is omitted for space considerations.

4.8.2.3 Construction based on clawfree collections

Perfect commitment schemes (of constant number of rounds) can be constructed using a strong intractability assumption; specifically, the existence of clawfree collections (see Subsection 2.4.5). This assumption implies the existence of one-way functions, but it is not known whether the converse is true. Nevertheless, clawfree collections can be constructed under widely believed assumptions such as the intractability of factoring and DLP. Actually, the construction of perfect commitment schemes, presented below, uses a clawfree collection with an additional property; specifically, it is assume that the set of indices of the collection (i.e., the range of algorithm I) can be efficiently recognized (i.e., is in $\mathcal{BPP}$). We remark that such collections do exist under the assumption that DLP is intractable (see Subsection 2.4.5).

Construction 4.8.5 (perfect bit commitment): *Let (I, D, F) be a triplet of efficient algorithms.*

1. commit phase: *To receive a commitment to a bit (using security parameter n), the receiver randomly generates $i = I(1^n)$ and sends it to the sender. To commit to value*

$v \in \{0,1\}$ *(upon receiving the message i from the receiver), the sender checks if indeed i is in the range of $I(1^n)$, and if so the sender randomly generates $s = D(i)$, computes $c = F(v,i,s)$, and sends c to the receiver. (In case i is not in the range of $I(1^n)$ the sender aborts the protocol announcing that the receiver is cheating.)*

2. reveal phase: *In the reveal phase, the sender reveals the string s used in the commit phase. The receiver accepts the value v if $F(v,i,s) = c$, where (i,c) is the receiver's (partial) view of the commit phase.*

Proposition 4.8.6 *Let (I,D,F) be a clawfree collection with a probabilistic polynomial-time recognizable set of indices (i.e., range of algorithm I). Then, the protocol presented in Construction 4.8.5 constitutes a perfect bit commitment scheme.*

Proof: The secrecy requirement follows directly from Property (2) of a clawfree collection (combined with the test $i \in I(1^n)$ conducted by the sender). The unambiguity requirement follows from Property (3) of a clawfree collection, using a standard reducibility argument. ■

We remark that the Factoring Clawfree Collection, presented in Subsection 2.4.5, can be used to construct a perfect commitment scheme although this collection *is not known* to have an efficiently recognizable index set. Hence, perfect commitment schemes exists also under the assumption that factoring Blum integers is intractable. Loosely speaking, this is done by letting the receiver prove to the sender (in zero-knowledge) that the selected index, N, satisfies the secrecy requirement. What is actually being proven is that half of the square roots, of each quadratic residue mod N, have Jacobi symbol 1 (relative to N). A zero-knowledge proof system of this claim does exist (without assuming anything). We remark that the idea just presented can be described as replacing the requirement that the index set is efficiently recognizable by a zero-knowledge proof that a string is indeed a legitimate index.

4.8.2.4 Commitment Schemes with a posteriori secrecy

We conclude the discussion of perfect commitment schemes by introducing a relaxation of the secrecy requirement. The resulting scheme cannot be used for the purposes of the current section, yet it is useful in different settings. The advantage in the relaxation is that it allows to construct commitment schemes using any clawfree collection, thus waiving the additional requirement that the index set is efficiently recognizable.

Loosely speaking, we relax the secrecy requirement of perfect commitment schemes by requiring that it only holds whenever the receiver follows it prescribed program (denoted R). This seems strange since we don't really want to assume that the real receiver follows the prescribed program (but rather allow it to behave arbitrarily). The point is that a real receiver may disclose the coin tosses used by it in the commit phase in a later stage, say even after the reveal phase, and by doing so a posteriori prove that (at least in some weak sense) it was following the prescribed program. Actually, the receiver only proves that he behaved in a manner which is consistent with its program.

Definition 4.8.7 (commitment scheme with perfect a posteriori secrecy)*: A* bit commitment scheme with perfect a posteriori secrecy *is defined as in Definition 4.8.2, except that the secrecy requirement is replaced by the following* a posteriori secrecy requirement*: For every string $r \in \{0,1\}^{\mathrm{poly}(n)}$ it holds that $\langle S(0), R_r\rangle(1^n)$ and $\langle S(1), R_r\rangle(1^n)$ are statistically close, where R_r denotes the execution of the interactive machine R when using internal coin tosses r.*

Proposition 4.8.8 *Let (I, D, F) be a clawfree collection. Consider a modification of Construction 4.8.5, in which the sender's check, of whether i is in the range of $I(1^n)$, is omitted (from the commit phase). Then the resulting protocol constitutes a bit commitment scheme with perfect a posteriori secrecy.*

In contrast to Proposition 4.8.6, here the clawfree collection may not have an efficiently recognizable index set. Hence, the verifier's check must have been omitted. Yet, the receiver can later prove that the message sent by it during the commit phase (i.e., i) is indeed a valid index by disclosing the random coins it has used in order to generate i (using algorithm I).

Proof: The a posteriori secrecy requirement follows directly from Property (2) of a clawfree collection (combined with the assumption that i in indeed a valid index). The unambiguity requirement follows as in Proposition 4.8.6. ■

A typical application of commitment scheme with perfect a posteriori secrecy is presented in Section 4.9. In that setting the commitment scheme is used inside an interactive proof with the verifier playing the role of the sender (and the prover playing the role of the receiver). If the verifier a posteriori learns that the prover has been cheating then the verifier rejects the input. Hence, no damage is caused, in this case, by the fact that the secrecy of the verifier's commitments might have been breached.

4.8.2.5 Nonuniform computational unambiguity

Actually, for the applications to proof/argument systems, both the one below and the one in Section 4.9, we need commitment schemes with perfect secrecy and *nonuniform* computational unambiguity. (The reasons for this need are analogous to the case of the zero-knowledge proof for $\mathcal{NP}$ presented in Section 4.4.) By nonuniform computational unambiguity we mean that the unambiguity condition should hold also for (nonuniform) families of polynomial-size circuits. We stress that all the constructions of perfect commitment schemes possess the nonuniform computational unambiguity, provided that the underlying clawfree collections foil also nonuniform polynomial-size claw-forming circuits.

In order to prevent the terms of becoming too cumbersome we omit the phrase "nonuniform" when referring to the perfect commitment schemes in the description of the two applications.

4.8.3 Perfect Zero-Knowledge Arguments for NP

Having perfect commitment scheme at our disposal, we can construct perfect zero-knowledge arguments for $\mathcal{NP}$, by modifying the construction of (computational) zero-knowledge proofs (for $\mathcal{NP}$) in a totally syntactic manner. We recall that in these proof systems (e.g., Construction 4.4.6 for Graph 3-Colorability) the prover uses a commitment scheme in order to commit itself to many values, part of them it later reveals upon the verifier's request. All that is needed is to replace the commitment scheme used by the prover by a perfect commitment scheme. We claim that the resulting protocol is a perfect zero-knowledge argument (computationally sound proof) for the original language. For sake of concreteness we prove

Proposition 4.8.9 *Consider a modification of Construction 4.4.6 so that the commitment scheme used by the prover is replaced by a perfect commitment scheme. Then the resulting protocol is a perfect zero-knowledge weak argument for Graph 3-Colorability.*

By a weak argument we mean a protocol in which the gap between the completeness and the computational soundness condition is non-negligible. In our case the verifier always accepts inputs in $G3C$, whereas no efficient prover can fool him into accepting graphs $G=(V,E)$ not in $G3C$ with probability greater than $1-\frac{1}{2|E|}$. We remind the reader that by polynomially many repetitions the error probability can be made negligible.

Proof Sketch: We start by proving that the resulting protocol is perfect zero-knowledge for $G3C$. We use the same simulator as in the proof of Proposition 4.4.7. However, this time analyzing the properties of the simulator is much easier since the commitments are distributed independently of the committed values, and consequently the verifier acts in total oblivion of the values. It follows that the simulator outputs a transcript with probability exactly $\frac{2}{3}$, and for similar reasons this transcript is distributed identically to the real interaction. The perfect zero-knowledge property follows.

The completeness condition is obvious as in the proof of Proposition 4.4.7. It is left to prove that the protocol satisfies the computational soundness requirement. This is indeed the more subtle part of the current proof (in contrast to the proof of Proposition 4.4.7 in which proving soundness is quite easy). We use a reducibility argument to show that a prover's ability to cheat with too high probability on inputs not in $G3C$ translates to an algorithm contradicting the unambiguity of the commitment scheme. Details follows.

We assume, to the contradiction, that there exists a (polynomial-time) cheating prover P^*, and an infinite sequence integers, so that for each integer n there exists graphs $G_n = (V_n, E_n) \notin G3C$ and a string y_n so that $P^*(y_n)$ leads the verifier to accept G_n with probability $> 1-\frac{1}{2|E_n|}$. Let $k \stackrel{\rm def}{=} |V_n|$. Let $c_1, ..., c_k$ be the sequence of commitments (to the vertices colors) sent by the prover in step (P1). Recall that in the next step, the verifier sends a uniformly chosen edge (of E_n) and the prover must answer by revealing different colors for its endpoint, otherwise the verifier rejects. A straightforward calculation shows that, since G_n is not 3-colorable, there must exist a vertex for which the prover is able to reveal at least two different colors. Hence, we can construct a polynomial-size circuit, incorporating

P^*, G_n and y_n, that violates the (nonuniform) unambiguity condition. Contradiction to the hypothesis of the proposition follows, and this completes the proof. ■

Combining Propositions 4.8.4 and 4.8.9, we get

Corollary 4.8.10 *If non-uniformly one-way permutations exist then every language in $\mathcal{NP}$ has a perfect zero-knowledge argument.*

Concluding Remarks: Propositions 4.4.7 and 4.8.9 exhibit a kind of a trade-off between the strength of the soundness and zero-knowledge properties. The protocol of Proposition 4.4.7 offers computational zero-knowledge and "perfect" soundness, whereas the protocol of Proposition 4.8.9 offers perfect zero-knowledge and computational soundness. However, one should note that *the two results are not obtained under the same assumptions*. The conclusion of Proposition 4.4.7 is valid as long as any one-way functions exist, whereas the conclusion of Proposition 4.8.9 requires a (probably much) stronger assumption. Yet, one may ask which of the two protocols should we prefer, *assuming that they are both valid*. The answer depends on the setting (i.e., application) in which the protocol is to be used. In particular, one should consider the following issues

- The relative importance attributed to soundness and zero-knowledge in the specific application. In case of clear priori to one of the two properties a choice should be made accordingly.
- The computational resources of the various users in the application. One of the users may be known to be in possession of much more substantial computing resources, and it may be reasonable to require that he/she should not be able to cheat even not in an information theoretic sense.
- The soundness requirement refers only to the duration of the execution, whereas in many applications zero-knowledge may be of concern also for a long time afterwards. If this is the case then perfect zero-knowledge arguments do offer a clear advantage (over zero-knowledge proofs).

4.8.4 Zero-Knowledge Arguments of Polylogarithmic Efficiency

A dramatic improvement in the efficiency of zero-knowledge arguments for $\mathcal{NP}$, can be obtained by combining ideas from Chapter `[missing(sign.sec)]` and a result described in Section `[missing(eff-pcp.sec)]`. In particular, assuming the existence of very strong collision-free hashing functions one can construct a computationally-sound (zero-knowledge) proof, for any language in $\mathcal{NP}$, which uses only polylogarithmic amount of communication and randomness. The interesting point in the above statement is the mere existence of such extremely efficient argument, let alone their zero-knowledge property. Hence, we refrain ourselves to describing the ideas involved in constructing such arguments, and do not address the issue of making them zero-knowledge.

By Theorem [missing(np-pcp.thm)], every $\mathcal{NP}$ language, L, can be reduced to $3SAT$ so that non-members of L are mapped into 3CNF formulae for which every truth assignment satisfies at most an $1-\epsilon$ fraction of the clauses, where $\epsilon > 0$ is a universal constant. Let us denote this reduction by f. Now, in order to prove that $x \in L$ it suffices to prove that the formula $f(x)$ is satisfiable. This can be done by supplying a satisfying assignment for $f(x)$. The interesting point is that the verifier need not check that all clauses of $f(x)$ are satisfied by the given assignment. Instead, it may uniformly select only polylogarithmically many clauses and check that the assignment satisfies all of them. If $x \in L$ (and the prover supplies a satisfying assignment to $f(x)$) then the verifier will always accept. Yet, if $x \notin L$ then no assignment satisfies more than a $1-\epsilon$ fraction of the clauses, and consequently a uniformly chosen clause is not satisfied with probability at least ϵ. Hence, checking superlogarithmically many clauses will do.

The above paragraph explains why the randomness complexity is polylogarithmic, but it does not explain why the same holds for the communication complexity. For this end we need an additional idea. The idea is to use a special commitment scheme which allows to commit to a string of length n so that the commitment phase takes polylogarithmic communication and individual bits of this string can be revealed (and verified correct) at polylogarithmic communication cost. For constructing such a commitment scheme we use a *collision-free* hashing function. The function maps strings of some length to strings of half the length so that it is "hard" to find two strings which are mapped by the function to the same image. (The following description is slightly inaccurate. What we need is a family of hash function such that no small non-uniform circuit, given the description of a function in the family can form collisions with respect to it.)

Let n denote the length of the input string to which the sender wishes to commit itself, and let k be a parameter (which is later set to be polylogarithmic in n). Denote by H a collision-free hashing function mapping strings of length $2k$ into strings of length k. The sender partitions its input string into $m \stackrel{\text{def}}{=} \frac{n}{k}$ consequtive blocks, each of length k. Next, the sender constructs a binary tree of depth $\log_2 m$, placing the m blocks in the corresponding leaves of the tree. In each internal node, the sender places the hash value obtained by applying the function H to the contents of the children of this node. The only message sent in the commit phase is the contents of the root (sent by the sender to the receiver). By doing so, *unless the sender can form collisions under H*, the sender has "committed" itself to some n-bit long string. When the receiver wishes to get the value of a specific bit in the string, the sender reveals to the receiver the contents of *both children* of each node along the path from the root to the corresponding leaf. The receiver checks that the values supplied for each node (along the path) match the value obtained by applying H to the values supplied for the two children.

The protocol for arguing that $x \in L$ consists of the prover committing itself to a satisfying assignment for $f(x)$, using the above scheme, and the verifier checking individual clauses by asking the prover to reveal the values assigned to the variables in these clauses. The protocol can be shown to be computationally-sound provided that it is infeasible to find a pair $\alpha, \beta \in \{0,1\}^{2k}$ so that $H(\alpha) = H(\beta)$. Specifically, we need to assume that forming collisions under H is not possible in subexponential time; namely, that for some $\delta > 0$, forming collisions with probability greater than 2^{-k^δ} must take at least 2^{k^δ} time. In such a case, we set $k = (\log n)^{1+\frac{1}{\delta}}$ and get a computationally-sound proof of communication complexity $O(\frac{\log n}{o(1)} \cdot m \cdot k) = \text{polylog}(n)$. (Weaker lower bounds for the collision-forming task may still yield meaningful results by an appropriate setting of the parameter k.) We stress that collisions can always be formed in time 2^{2k} and hence the entire approach fails if the prover is not computationally bounded (and consequently we cannot get (perfectly-sound) proof systems this way). Furthermore, by a simulation argument one may show that, only languages in Dtime(2^{polylog}) have proof systems with polylogarithmic communication and randomness complexity.

4.9 * Constant Round Zero-Knowledge Proofs

In this section we consider the problem of constructing *constant-round* zero-knowledge proof systems *with negligible error probability* for all languages in $\mathcal{NP}$. To make the rest of the discussion less cumbersome we define a proof system to be *round-efficient* if it is both constant-round and with negligible error probability.

We present two approaches to the construction of round-efficient zero-knowledge proofs for $\mathcal{NP}$.

1. Basing the construction of round-efficient zero-knowledge proof systems on commitment schemes with perfect secrecy (see Subsection 4.8.2).

2. Constructing (round-efficient zero-knowledge) *computationally-sound* proof systems (see Section 4.8) instead of (round-efficient zero-knowledge) proof systems.

The advantage of the second approach is that round-efficient zero-knowledge computationally-sound proof systems for $\mathcal{NP}$ can be constructed using any one-way function, whereas it is not known whether round-efficient zero-knowledge proof systems for $\mathcal{NP}$ can be constructed under the same general assumption. In particular, we only know how to construct perfect commitment schemes by using much stronger assumptions (e.g., the existence of clawfree permutations).

Both approaches have one fundamental idea in common. We start with an abstract exposition of this common idea. Recall that the *basic* zero-knowledge proof for Graph 3-Colorability, presented in Construction 4.4.6, consists of a constant number of rounds. However, this proof system has a non-negligible error probability (in fact the error probability is very close to 1). In Section 4.4, it was suggested to reduce the error probability to a negligible one by sequentially applying the proof system sufficiently many times. The problem is that this yields a proof system with a non-constant number of rounds. A natural suggestion is to perform the repetitions of the basic proof in parallel, instead of sequentially.

The problem with this "solution" is that it is not known whether that the resulting proof system is zero-knowledge.

> Furthermore, it is known that it is not possible to present, as done in the proof of Proposition 4.4.7, a single simulator which uses every possible verifier as a black box (see Section 4.5). The source of trouble is that, when playing many versions of Construction 4.4.6 in parallel, a cheating verifier may select the edge to be inspected (i.e., step (V1)) in each version depending on the commitments sent in all versions (i.e., in step (P1)). Such behaviour of the verifier defeats a simulator analogous to the one presented in the proof of Proposition 4.4.7.

The way to overcome this difficulty is to "switch" the order of steps (P1) and (V1). But switching the order of these steps enables the prover to cheat (by sending commitments in which only the "query" edges are colored correctly). Hence, a more refined approach is required. The verifier starts by committing itself to one edge-query for each version (of Construction 4.4.6), then the prover commits itself to the coloring in each version, and only then the verifier reveals its queries and the rest of the proof proceeds as before. The commitment scheme used by the verifier should prevent the prover from predicting the sequence of edges committed to by the verifier. This is the point were the two approaches differ.

1. The first approach utilizes for this purpose a commitment scheme with perfect secrecy. The problem with this approach is that such schemes are known to exists only under stronger assumption than merely the existence of one-way function. Yet, such schemes do exists under assumptions such as the intractability of factoring integers of special form or the intractability of the discrete logarithm problem.

2. The second approach bounds the computational resources of prospective cheating provers. Consequently, it suffices to utilize, "against" these provers (as commitment receivers), commitment schemes with computational security. We remark that this approach utilizes (for the commitments done by the prover) a commitment scheme with an extra property. Yet, such schemes can be constructed using any one-way function.

We remark that both approaches lead to protocols that are zero-knowledge in a liberal sense (i.e., using expected polynomial-time simulators).

4.9.1 Using commitment schemes with perfect secrecy

For sake of clarity, let us start by presenting a detailed description of the constant-round interactive proof (for Graph 3-Colorability (i.e., $G3C$)) sketched above. This interactive proof employs two different commitment schemes. The first scheme is the simple commitment scheme (with "computational" secrecy) presented in Construction 4.4.2. We denote by $C_s(\sigma)$ the commitment of the sender, using coins s, to the (ternary) value σ. The second commitment scheme is a commitment scheme with perfect secrecy (see Section 4.8.2). For simplicity, we assume that this scheme has a commit phase in which the receiver sends one message to the sender which then replies with a single message (e.g., the schemes presented in Section 4.8.2). Let us denote by $P_{m,s}(\alpha)$ the commitment of the sender to string α, upon receiving message m (from the receiver) and when using coins s.

Construction 4.9.1 (A round-efficient zero-knowledge proof for G3C):

- Common Input: *A simple (3-colorable) graph* $G=(V,E)$. *Let* $n \stackrel{\text{def}}{=} |V|$, $t \stackrel{\text{def}}{=} n \cdot |E|$ *and* $V = \{1,...,n\}$.
- Auxiliary Input to the Prover: *A 3-coloring of* G, *denoted* ψ.
- Prover's preliminary step (P0): *The prover invokes the commit phase of the perfect commit scheme, which results in sending to the verifier a message* m.
- Verifier's preliminary step (V0): *The verifier uniformly and independently selects a sequence of* t *edges,* $\overline{E} \stackrel{\text{def}}{=} ((u_1,v_1),...,(u_t,v_t)) \in E^t$, *and sends the prover a random commitment to these edges. Namely, the verifier uniformly selects* $\overline{s} \in \{0,1\}^n$ *and sends* $P_{m,\overline{s}}(\overline{E})$ *to the prover;*
- Motivating Remark: *At this point the verifier is committed to a sequence of* t *edges. This commitment is of perfect secrecy;*
- Prover's step (P1): *The prover uniformly and independently selects* t *permutations,* $\pi_1,...,\pi_t$, *over* $\{1,2,3\}$, *and sets* $\phi_j(v) \stackrel{\text{def}}{=} \pi_j(\psi(v))$, *for each* $v \in V$ *and* $1 \le j \le t$. *The prover uses the computational commitment scheme to commit itself to colors of each of the vertices according to each 3-coloring. Namely, the prover uniformly and independently selects* $s_{1,1},...,s_{n,t} \in \{0,1\}^n$, *computes* $c_{i,j} = C_{s_{i,j}}(\phi_j(i))$, *for each* $i \in V$ *and* $1 \le j \le t$, *and sends* $c_{1,1},...,c_{n,t}$ *to the verifier;*
- Verifier's step (V1): *The verifier reveals the sequence* $\overline{E} = ((u_1,v_1),...,(u_t,v_t))$ *to the prover. Namely, the verifier send* $(\overline{s},\overline{E})$ *to the prover;*
- Motivating Remark: *At this point the entire commitment of the verifier is revealed. The verifier now expects to receive, for each* j, *the colors assigned by the* j^{th} *coloring to vertices* u_j *and* v_j *(the endpoints of the* j^{th} *edge in* $\overline{E}$*);*
- Prover's step (P2): *The prover checks that the message just received from the verifier is indeed a valid revealing of the commitment made by the verifier at step (V0). Otherwise the prover halts immediately. Let us denote the sequence of* t *edges, just revealed, by* $(u_1,v_1),...,(u_t,v_t)$. *The prover uses the reveal phase of the computational*

commitment scheme in order to reveal, for each j, the j^{th} coloring of vertices u_j and v_j to the verifier. Namely, the prover sends to the verifier the sequence of quadruples

$$(s_{u_1,1}, \phi_1(u_1), s_{v_1,1}, \phi_1(v_1)), ..., (s_{u_t,t}, \phi_t(u_t), s_{v_t,t}, \phi_t(v_t))$$

- Verifier's step (V2): *The verifier checks whether, for each j, the values in the j^{th} quadruple constitute a correct revealing of the commitments $c_{u_j,j}$ and $c_{v_j,j}$, and whether the corresponding values are different. Namely, upon receiving $(s_1, \sigma_1, s'_1, \tau_1)$ through $(s_t, \sigma_t, s'_t, \tau_t)$, the verifier checks whether for each j, it holds that $c_{u_j,j} = C_{s_j}(\sigma_j)$, $c_{v_j,j} = C_{s'_j}(\tau_j)$, and $\sigma_j \neq \tau_j$ (and both are in $\{1, 2, 3\}$). If all conditions hold then the verifier accepts. Otherwise it rejects.*

We first assert that Construction 4.9.1 is indeed an interactive proof for $G3C$. Clearly, the verifier always accepts a common input in $G3C$. Suppose that the common input graph, $G = (V, E)$, is not in $G3C$. Clearly, each of the "committed colorings" sent by the prover in step (P1) contains at least one illegally-colored edge. Using the perfect secrecy of the commitments sent by the verifier in step (V0), we deduce that at step (P1) the prover has "no idea" which edges the verifier asks to see (i.e., as far as the information available to the prover is concerned, each possibility is equally likely). Hence, although the prover sends the "coloring commitment" after receiving the "edge commitment", the probability that all the "committed edges" have legally "committed coloring" is at most

$$\left(1 - \frac{1}{|E|}\right)^t \approx e^{-n} < 2^{-n}$$

We now turn to show that Construction 4.9.1 is indeed zero-knowledge (in the liberal sense allowing *expected* polynomial-time simulators). For every probabilistic (expected) polynomial-time interactive machine, V^*, we introduce an expected polynomial-time simulator, denoted M^*. The simulator starts by selecting and fixing a random tape, r, for V^*. Given the input graph G and the random tape r, the commitment message of the verifier V^* is determined. Hence, M^* invokes V^*, on input G and random tape r, and gets the corresponding commitment message, denoted CM. The simulator proceeds in two steps.

S1) *Extracting the query edges*: M^* generates a sequence of $n \cdot t$ random commitments to dummy values (e.g., all values equal 1), and feeds it to V^*. In case V^* replies by revealing correctly a sequence of t edges, denoted $(u_1, v_1), ..., (u_t, v_t)$, the simulator records these edges and proceed to the next step. In case the reply of V^* is not a valid revealing of the commitment message CM, the simulator halts outputting the current view of V^* (e.g., G, r and the commitments to dummy values).

S2) *Generating an interaction that satisfies the query edges* (oversimplified exposition): Let $(u_1, v_1), ..., (u_t, v_t)$ denote the sequence of edges recorded in step (S1). M^* generates a sequence of $n \cdot t$ commitments, $c_{1,1}, ..., c_{n,t}$, so that for each $j = 1, ..., t$, it holds that

$c_{u_j,j}$ and $c_{v_j,j}$ are random commitments to two different random values in $\{1, 2, 3\}$ and all the other $c_{i,j}$'s are random commitments to dummy values (e.g., all values equal 1). The underlying values are called a *pseudo-colorings*. The simulator feeds this sequence of commitments to V^*. If V^* replies by revealing correctly the (above recorded) sequence of edges, then M^* can complete the simulation of a "real" interaction of V^* (by revealing the colors of the endpoints of these recorded edges). Otherwise, the entire step is repeated (until success occurs).

In the rest of the description we ignore the possibility that, when invoked in steps (S1) and (S2), the verifier reveals two different edge commitments. Loosely speaking, this practice is justified by the fact that during expected polynomial-time computations such event can occur only with negligible probability (since otherwise it contradicts the computational unambiguity of the commitment scheme used by the verifier).

To illustrate the behaviour of the simulator assume that the program V^* always reveals correctly the commitment done in step (V0). In such a case, the simulator will find out the query edges in step (S1), and using them in step (S2) it will simulate the interaction of V^* with the real prover. Using ideas as in Section 4.4 one can show that the simulation is computational indistinguishable from the real interaction. Note that in this case, step (S2) of the simulator is performed only once.

Consider now a more complex case in which, on each possible sequence of internal coin tosses r, program V^* correctly reveals the commitment done in step (V0) only with probability $\frac{1}{3}$. The probability in this statement is taken over all possible commitments generated to the dummy values (in the simulator step (S1)). We first observe that the probability that V^* correctly reveals the commitment done in step (V0), after receiving a random commitment to a sequence of pseudo-colorings (generated by the simulator in step (S2)), is approximately $\frac{1}{3}$. (Otherwise, we derive a contradiction to the computational secrecy of the commitment scheme used by the prover.) Hence, the simulator reaches step (S2) with probability $\frac{1}{3}$, and each execution of step (S2) is completed successfully with probability $p \approx \frac{1}{3}$. It follows that the expected number of times that step (S2) is invoked when running the simulator is $\frac{1}{3} \cdot \frac{1}{p} \approx 1$.

Let us now consider the general case. Let $q(G, r)$ denote the probability that, on input graph G and random tape r, *after receiving random commitments to dummy values* (generated in step (S1)), program V^* correctly reveals the commitment done in step (V0). Likewise, we denote by $p(G, r)$ the probability that, (on input graph G and random tape r) *after receiving a random commitment to a sequence of pseudo-colorings* (generated by the simulator in step (S2)), program V^* correctly reveals the commitment done in step (V0). As before the difference between $q(G, r)$ and $p(G, r)$ is negligible (in terms of the size of the graph G), otherwise one derives contradiction to the computational secrecy of the prover's commitment scheme. We conclude that the simulator reaches step (S2) with probability $q \stackrel{\text{def}}{=} q(G, r)$, and each execution of step (S2) is completed successfully with probability $p \stackrel{\text{def}}{=} p(G, r)$. It follows that the expected number of times that step (S2) is invoked when running the simulator is $q \cdot \frac{1}{p}$. Here are the bad news: we cannot guarantee that $\frac{q}{p}$ is approximately 1 or even bounded by a polynomial in the input size (e.g., let $p = 2^{-n}$ and $q = 2^{-n/2}$, then the difference between them is negligible and yet $\frac{q}{p}$ is not bounded by $\text{poly}(n)$). This is why the above description of the simulator is oversimplified and a modification is indeed required.

We make the simulator expected polynomial-time by modifying step (S2) as follows. We add an intermediate step (S1.5), to be performed only if the simulator did not halt in step (S1). The purpose of step (S1.5) is to provide a good estimate of $q(G,r)$. The estimate is computed by repeating step (S1) until a fixed (polynomial in $|G|$) number of correct V^*-reveals are encountered (i.e., the estimate will be the ratio of the number of successes divided by the number of trial). By fixing a sufficiently large polynomial, we can guarantee that with overwhelmingly high probability (i.e., $1 - 2^{-\text{poly}(|G|)}$) the estimate is within a constant factor of $q(G,r)$. It is easily verified that the estimate can be computed within expected time $\text{poly}(|G|)/q(G,r)$. Step (S2) of the simulator is modified by adding a bound on the number of times it is performed, and if none of these executions yield a correct V^*-reveal then the simulator outputs a *special empty interaction*. Specifically, step (S2) will be performed at most $\text{poly}(|G|)/\overline{q}$, where $\overline{q}$ is the estimate to $q(G,r)$ computed in step (S1.5). It follows that the modified simulator has expected running time bounded by $q(G,r) \cdot \frac{\text{poly}(|G|)}{q(G,r)} = \text{poly}(|G|)$.

It is left to analyze the output distribution of the modified simulator. We refrain ourselves to reducing this analysis to the analysis of the output of the original simulator, by bounding the probability that the modified simulator outputs a special empty interaction. This probability is bounded by

$$\begin{aligned}
\Delta(G,r) &\stackrel{\text{def}}{=} q(G,r) - q(G,r) \cdot \left(1 - (1 - p(G,r))^{\text{poly}(|G|)/q(G,r)}\right) \\
&= q(G,r) \cdot (1 - p(G,r))^{\text{poly}(|G|)/q(G,r)}
\end{aligned}$$

We claim that $\Delta(G,r)$ is a negligible function of $|G|$. Assume, to the contrary, that there exists a polynomial $P(\cdot)$, an infinite sequence of graphs $\{G_n\}$, and an infinite sequence of random tapes $\{r_n\}$, such that $\Delta(G_n,r_n) > 1/P(n)$. It follows that for each such n we have $q(G_n,r_n) > 1/P(n)$. We consider two cases.

Case 1: For infinitely many n's, it holds that $p(G_n,r_n) \geq q(G_n,r_n)/2$. In such a case we get for these n's

$$\begin{aligned}
\Delta(G_n,r_n) &\leq (1 - p(G_n,r_n))^{\text{poly}(|G_n|)/q(G_n,r_n)} \\
&\leq \left(1 - \frac{q(G_n,r_n)}{2}\right)^{\text{poly}(|G_n|)/q(G_n,r_n)} \\
&< 2^{-\text{poly}(|G_n|)/2}
\end{aligned}$$

which contradicts our hypothesis that $\Delta(G_n,r_n) > 1/\text{poly}(n)$.

Case 2: For infinitely many n's, it holds that $p(G_n,r_n) < q(G_n,r_n)/2$. It follows that for these n's we have $|q(G_n,r_n) - p(G_n,r_n)| > P(n)/2$, which leads to contradiction of the computational secrecy of the commitment scheme (used by the prover).

Hence, contradiction follows in both cases.

We remark that one can modify Construction 4.9.1 so that weaker forms of perfect commitment schemes can be used. We refer specifically to commitment schemes with perfect a posteriori secrecy (see Subsection 4.8.2). In such schemes the secrecy is only established a posteriori by the receiver which discloses the coin tosses it has used in the commit phase. In our case, the prover plays the role of the receiver, and the verifier plays the role of the sender. It suffices to establish the secrecy property a posteriori, since in case secrecy is not establish the verifier may reject. In such a case no harm has been caused since the secrecy of the perfect commitment scheme is used only to establish the soundness of the interactive proof.

4.9.2 Bounding the power of cheating provers

Construction 4.9.1 can be modified to yield a zero-knowledge computationally sound proof, under the (more general) assumption that one-way functions exist. In the modified protocol, we let the verifier use a commitment scheme with computational secrecy, instead of the commitment scheme with perfect secrecy used in Construction 4.9.1. (Hence, both users commit to their messages using commitment scheme with computational secrecy.) Furthermore, the commitment scheme used by the prover must have the extra property that it is infeasible to construct a commitment without "knowing" to what value it commits. Such a commitment scheme is called *non-oblivious*. We start by defining and constructing non-oblivious commitment schemes.

4.9.2.1 Non-oblivious commitment schemes

The non-obliviousness of a commitment scheme is intimately related to the definition of proof of knowledge (see Section 4.7).

Definition 4.9.2 (non-oblivious commitment schemes)*: Let (S,R) be a commitment scheme as in Definition 4.4.1. We say that the commitment scheme is* non-oblivious *if the prescribed receiver, R, constitutes a knowledge-verifier, that is always convinced by S, for the relation*

$$\{((1^n,r,\overline{m}),(\sigma,s)) : \overline{m} = \mathrm{view}_{R(1^n,r)}^{S(\sigma,1^n,s)}\}$$

where, as in Definition 4.4.1, $\mathrm{view}_{R(1^n,r)}^{S(\sigma,1^n,s)}$ *denotes the messages received by the interactive machine R on input 1^n and local-coins r, when interactive with machine S (that has input $(\sigma,1^n)$ and uses coins s).*

It follows that the receiver prescribed program, R, may accept or rejects at the end of the commit phase, and that this decision is supposed to reflect the sender's ability to later come up with a legal opening of the commitment (i.e., successfully complete the reveal phase). We stress that non-obliviousness relates mainly to cheating senders, since the prescribed sender has no difficulty to later successfully complete the reveal phase (and in fact during the commit phase S always convinces the receiver of this ability). Hence, *any* sender program (not merely the prescribed S) can be modified so that at the end of the commit phase it (locally) outputs information enabling the reveal phase (i.e., σ and s). The modified sender runs in expected time that is inversely proportional to the probability that the commit phase is completed successfully.

We remark that in an ordinary commitment scheme, at the end of the commit phase, the receiver does not necessarily "know" whether the sender can later successfully conduct the reveal phase. For example, a cheating sender in Construction 4.4.2 can (undetectedly) perform the commit phase without ability to later successfully perform the reveal phase (e.g., the sender may just send a uniformly chosen string). It is only guaranteed that if the sender follows the prescribed program then the sender can always succeed in the reveal phase. Furthermore, with respect to the scheme presented in Construction 4.4.4, a cheating sender can (undetectedly) perform the commit phase in a way that it generates a receiver view which does not have any corresponding legal opening (and hence the reveal phase is doomed to fail). See Exercise 14.

Nevertheless

Theorem 4.9.3 *If one-way functions exist then there exist non-oblivious commitment schemes with constant number of communication rounds.*

We recall that (ordinary) commitment schemes can be constructed assuming the existence of one-way functions (see Proposition 4.4.5 and Theorem 3.5.12). Consider the relation corresponding to such a scheme. Using zero-knowledge proofs of knowledge (see Section 4.7) for the above relation, we get a non-oblivious commitment scheme. (We remark that such proofs do exist under the same assumptions.) However, the resulting commitment scheme has unbounded number of rounds (due to the round complexity of the zero-knowledge proof). We seem to have reached a vicious circle, yet there is a way out. We can use constant-round witness indistinguishable proofs (see Section 4.6), instead of the zero-knowledge proofs. The resulting commitment scheme has the additional property that when applied (polynomially) many times in parallel the secrecy property holds simultaneously in all copies. This fact follows from the Parallel Composition Lemma for witness indistinguishable proofs (see Section 4.6). The simultaneous secrecy of many copies is crucial to the following application.

4.9.2.2 Modifying Construction 4.9.1

We recall that we are referring to a modification of Construction 4.9.1 in which the verifier uses a commitment scheme (with computational secrecy), instead of the commitment scheme with perfect secrecy used in Construction 4.9.1. In addition, the commitment scheme used by the prover is non-oblivious.

We conclude this section by remarking on how to adopt the argument of the first approach (i.e., of Subsection 4.9.1) to suit our current needs. We start with the claim that the modified protocol is a computationally-sound proof for $G3C$. Verifying that the modified protocol satisfies the completeness condition is easy as usual. We remark that the modified protocol does not satisfy the (usual) soundness condition (e.g., a "prover" of exponential computing power can break the verifier's commitment and generate pseudo-colorings that will later fool the verifier into accepting). Nevertheless, we can show that the modified protocol does satisfy the computational soundness (of Definition 4.8.1). Namely, we show that for every polynomial $p(\cdot)$, every polynomial-time interactive machine B, and for all sufficiently large graph $G \notin G3C$ and every y and z

$$\mathsf{Pr}\left(\langle B(y), V_{G3C}(z)\rangle(x)=1\right) \leq \frac{1}{p(|x|)}$$

where V_{G3C} is the verifier program in the modified protocol.

Using the information theoretic unambiguity of the commitment scheme employed by the prover, we can talk of a unique color assignment which is induced by the prover's commitments. Using the fact that this commitment scheme is non-oblivious, it follows that the prover can be modified so that, in step (P1), it outputs the values to which it commits itself at this step. We can now use the computational secrecy of the verifier's commitment scheme to show that the color assignment generated by the prover is almost independent of the verifier's commitment. Hence, the probability that the prover can fool the verifier to accept an input not in the language is non-negligibly greater than what it would have been if the verifier asked random queries after the prover makes its (color) commitments. The computational soundness of the (modified) protocol follows. We remark that we do not know whether the protocol is computationally sound in case the prover uses a commitment scheme that is not guaranteed to be non-oblivious.

Showing that the (modified) protocol is zero-knowledge is even easier than it was in the first approach (i.e., in Subsection 4.9.1). The reason being that when demonstrating zero-knowledge of such protocols we use the secrecy of the prover's commitment scheme and the unambiguity of the verifier's commitment scheme. Hence, only these properties of the commitment schemes are relevant to the zero-knowledge property of the protocols. Yet, the current (modified) protocol uses commitment schemes with relevant properties which are not weaker than the ones of the corresponding commitment schemes used in Construction 4.9.1. Specifically, the prover's commitment scheme in the modified protocol possess computationally secrecy just like the prover's commitment scheme in Construction 4.9.1. We stress that this commitment, like the simpler commitment used for the prover in Construction 4.9.1, has the simultaneous secrecy (of many copies) property. Furthermore, the verifier's commitment scheme in the modified protocol possess "information theoretic" unambiguity, whereas the verifier's commitment scheme in Construction 4.9.1 is merely computationally unambiguous.

4.10 * Non-Interactive Zero-Knowledge Proofs

In this section we consider 'non-interactive' zero-knowledge proof systems. Actually, the term non-interactive is somewhat misleading. Indeed, in the model which we will discuss the interaction between the prover and the verifier is minimal; it consists of the prover sending a single message to the verifier (as in the case of an NP-proof). Yet, both the prover and the verifier have access to a (trusted) random string, which can be thought of as a restricted trusted third party. Non-interactive zero-knowledge proof systems have various applications (e.g., to Encryption Schemes secure against Chosen Message Attacks and to Signature Schemes).

We start with basic definitions and constructions allowing to prove a single assertion of a-priori bounded length. Next we extend the treatment to proof systems in which many assertions of various lengths can be proven, as long as the total length of all assertions is a polynomial in a security parameter but the polynomial is NOT a-priori known. Jumping ahead, we note that, unlike the basic treatment, the extended treatment allows to prove assertions of total length much bigger than the length of the trusted random string. The relation between the total length of the provable assertions and the length of the trusted random string is analogous to the relation between the total length of messages that can be encrypted (resp., documents that can be signed) and the length of the encryption-key (resp., signing-key). We stress, however, that even handling the basic case is very challenging in the case of non-interactive zero-knowledge proofs.

4.10.1 Basic Definitions

In the setting of non-interactive proof systems, both the prover and verifier are ordinary probabilistic machines which, in addition to the common-input, also get a *common random-string*. We stress that both the prover and verifier may toss coins and get auxiliary inputs, in addition to the above common input and random-string. However, for sake of simplicity we present a definition for the case in which none of these machines gets an auxiliary input. The verifier also gets as input the output produced by the prover.

Definition 4.10.1 (non-interactive proof system): *A pair of probabilistic machines, (P, V), is called a* non-interactive proof system for a language *L if V is polynomial-time and the following two conditions hold*

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

$$\mathsf{Pr}\left(V(x, R, P(x, R))=1\right) \geq \frac{2}{3}$$

  where R is a random variable uniformly distributed in $\{0,1\}^{\mathrm{poly}(|x|)}$.

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

$$\mathsf{Pr}\left(V(x, R, B(x, R))=1\right) \leq \frac{1}{3}$$

where R is a random variable uniformly distributed in $\{0,1\}^{\mathrm{poly}(|x|)}$.

The uniformly chosen string R is called the common random-string.

As usual, the error probability in both conditions can be reduced (from $\frac{1}{3}$) up to $2^{-\mathrm{poly}(|x|)}$, by repeating the "protocol" sufficiently many times (using a sequence of many independently chosen random-strings). In stating the soundness condition, we have deviated from common formulations which allows $x \notin L$ to be selected after R. Namely, in many sources the soundness condition is stated as

for every n and every pair of functions $\chi : \{0,1\}^{\mathrm{poly}(n)} \mapsto (\{0,1\}^n - L)$ *and* $\pi : \{0,1\}^{\mathrm{poly}(n)} \mapsto \{0,1\}^{\mathrm{poly}(n)}$

$$\mathsf{Pr}\left(V(\chi(R), R, \pi(R)) = 1\right) \leq \frac{1}{3}$$

where R is a random variable uniformly distributed in $\{0,1\}^{\mathrm{poly}(n)}$.

Clearly, the two formulations are equivalent; starting from the weaker soundness condition, one may first reduce the soundness error to $\frac{1}{3} \cdot 2^{-n}$ (by repetitions), and next apply a standard counting argument. Every language in $\mathcal{NP}$ has a non-interactive proof system (in which no randomness is used). However, this NP-proof system is unlikely to be zero-knowledge.

The definition of zero-knowledge for the non-interactive model gets simplified since we need only consider one verifier. Actually, we can avoid considering the verifier at all.

Definition 4.10.2 (non-interactive zero-knowledge): *A non-interactive proof system, (P,V), for a language L is* zero-knowledge *if there exists a probabilistic polynomial-time algorithm M such that the ensembles* $\{(x, R_{|x|}, P(x, R_{|x|}))\}_{x \in L}$ *and* $\{M(x)\}_{x \in L}$ *are computationally indistinguishable, where R_n is a random variable uniformly distributed in* $\{0,1\}^{\mathrm{poly}(n)}$.

4.10.2 Constructions

A fictitious abstraction which is nevertheless very helpful for constructing non-interactive zero-knowledge proof systems is the *hidden bits model*. In this model the common random-string is uniformly selected as before, but only the prover can see all of it. The 'proof' he sends the verifier consists of two parts; a 'certificate' and the specification of some bit positions in the common random-string. The verifier may only inspect the bits of the common random-string residing in the locations which have been specified by the prover. Certainly, in addition, the verifier inspects the common input and the 'certificate'.

Definition 4.10.3 (proof systems in the Hidden Bits Model): *A pair of probabilistic machines, (P,V), is called a* hidden-bits proof system for *L if V is polynomial-time and the following two conditions hold*

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

$$\mathsf{Pr}\left(V(x, R_I, I, \pi) = 1\right) \geq \frac{2}{3}$$

where $(I, \pi) \stackrel{\text{def}}{=} P(x, R)$*,* R *is a random variable uniformly distributed in* $\{0,1\}^{\text{poly}(|x|)}$ *and* R_I *is the sequence of bits at positions* $I \subseteq \{1, 2, ..., \text{poly}(|x|)\}$*. That is,* $R_I = r_{i_1} \cdots r_{i_t}$*, where* $R = r_1 \cdots r_t$ *and* $I = (i_1, ..., i_t)$*.*

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

$$\mathsf{Pr}\left(V(x, R_I, I, \pi) = 1\right) \leq \frac{1}{3}$$

where $(I, \pi) \stackrel{\text{def}}{=} B(x, R)$*,* R *is a random variable uniformly distributed in* $\{0,1\}^{\text{poly}(|x|)}$ *and* R_I *is the sequence of bits at positions* $I \subseteq \{1, 2, ..., \text{poly}(|x|)\}$*.*

In both cases, I *is called the set of* revealed bits *and* π *is called the* certificate*. Zero-knowledge is defined as before, with the exception that we need to simulate* $(x, R_I, P(x, R))$ *rather than* $(x, R, P(x, R))$*, where* $(I, \pi) = P(x, R)$*.*

As hinted above, we do not suggest the Hidden-Bits Model as a realistic model. The importance of the model stems from two facts. Firstly, it is a 'clean' model which facilitates the design of proof systems (in it), and secondly that proof systems in the Hidden-Bits Model can be easily transformed into non-interactive proof systems (i.e., the realistic model). The transformation follows.

Construction 4.10.4 (from Hidden Bits proof systems to non-interactive ones): *Let* (P, V) *be a hidden-bits proof system for* L*,* $f : \{0,1\}^* \mapsto \{0,1\}^*$ *and* $b : \{0,1\}^* \mapsto \{0,1\}$*. Furthermore, let* $m = \text{poly}(n)$ *denote the length of the common random-string for common inputs of length* n *and suppose that* f *is 1-1 and length preserving. Following is a specification of a non-interactive system,* (P', V')*:*

- Common Input: $x \in \{0,1\}^n$.
- Common Random-String: $s = (s_1, ..., s_m)$*, where each* s_i *is in* $\{0,1\}^n$*.*
- Prover (denoted P'):
  - *computes* $r_i = b(f^{-1}(s_i))$*, for* $i = 1, 2, ..., m$*.*
  - *invokes* P *to get* $(I, \pi) = P(x, r_1 \cdots r_m)$*.*
  - *outputs* (I, π, p_I)*, where* $p_I \stackrel{\text{def}}{=} (f^{-1}(s_{i_1}) \cdots f^{-1}(s_{i_t}))$ *for* $I = (i_1, ..., i_t)$*.*
- Verifier (denoted V') *given prover's output* $(I, \pi, (p_1 \cdots p_t))$*:*
  - *checks that* $s_{i_j} = f(p_j)$*, for each* $i_j \in I$*. In case a mismatch is found,* V' *rejects.*

- *computes* $r_i = b(p_i)$, *for* $i = 1, ..., t$. *Let* $r = r_1, ..., r_t$.
- *invokes* V *on* (x, r, I, π) *and accepts if and only if* V *accepts.*

Proposition 4.10.5 *Let* (P, V), L, f, b *and* (P', V') *be as in Construction 4.10.4. Then,* (P', V') *is a non-interactive proof system for* L, *provided that* $\mathsf{Pr}(b(U_n) = 1) = \frac{1}{2}$. *Furthermore, if* P *is zero-knowledge and* b *is a hard-core of* f *then* P' *is zero-knowledge too.*

We remark that P' is not perfect zero-knowledge even in case P is. Also, P' may not be implemented in polynomial-time (even with help of auxiliary inputs) even if P is (see Remark 4.10.6 below).

proof: To see that (P', V') is a non-interactive proof system for L we note that uniformly chosen $s_i \in \{0,1\}^n$ induce uniformly distributed bits $r_i \in \{0,1\}$. (This follows by $r_i = b(f^{-1}(s_i))$, the fact that f is one-to-one, and the fact that $b(f^{-1}(U_n)) = b(U_n)$ is unbiased.) Note that in case b is a hard-core of f, it is almost unbiased (i.e., $\mathsf{Pr}(b(U_n) = 1) = \frac{1}{2} \pm \frac{1}{\mu(n)}$, where μ is a negligible function). Thus, saying that b is a hard-core for f essentially suffices.

To see that P' is zero-knowledge note that we can convert an efficient simulator for P into an efficient simulator for P'. Specifically, for each revealed bit of value σ we uniformly select a string $r \in \{0,1\}^n$ so that $b(r) = \sigma$ and put $f(r)$ in the corresponding position in the common random-string. For each *un*revealed bit we uniformly select a string $s \in \{0,1\}^n$ and put it in the corresponding position in the common random-string. Using the fact that b is a hard-core of f, it follows that the simulator's output is computationally indistinguishable from the verifier's view. ■

Remark 4.10.6 (efficient implementation of P'): As stated above, in general P' cannot be efficiently implemented given a black-box access to P. What is needed is ability (of P') to invert f, however for P' to be zero-knowledge f must be one-way. The oblivious solution is to use a family of trapdoor permutations and let the prover know the trapdoor. Furthermore, the family should have the property that its members can be efficiently recognized (i.e., given a description of a function one can efficiently decide whether it is in the family). In other words, P' starts by selecting a permutation f over $\{0,1\}^n$ so that it knows its trapdoor, and proceeds as in Construction 4.10.4, except that it also appends f to the 'proof'. The verifier acts as in Construction 4.10.4 with respect to the function f specified in the proof. In addition it also checks that f is indeed in the family. Both the completeness and the zero-knowledge conditions follow exactly as in the proof of Proposition 4.10.5. For the soundness condition we need to consider all possible members of the family (w.l.o.g., there are at most 2^n such permutation). For each such permutation, the argument is as before and our claim thus follows by a counting argument. (Actually, we need also to repeat the (P, V) system for $O(n)$ times to make the counting argument work.)

We now turn to the construction of proof systems in the Hidden Bits model. Specifically, we are going to construct a proof system for the *Hamiltonian Cycle* (HC) problem which is

NP-complete (and thus get proof systems for any language in $\mathcal{NP}$). We consider directed graphs (and the existence of directed Hamiltonian cycles). Below, we present a basic zero-knowledge system in which Hamiltonian graphs are accepted with probability 1 whereas non-Hamiltonian graphs on n vertices are rejected with probability $\Omega(n^{-3/2})$. (This system is inspired by the one presented in Construction 4.7.11.)

Construction 4.10.7 (Hidden Bits systems for HC):

- Common Input: *a directed graph* $G=(V,E)$ *with* $n \stackrel{\text{def}}{=} |V|$.
- Common Random-String: *viewed as an* n^3*-by-*n^3 *Boolean matrix* M, *with each entry being 1 with probability* n^{-5}.
  (This is implemented by breaking the common random-string to blocks of length $5\log_2 n$ and setting a matrix entry to 1 iff the corresponding block is all 1's.)
- Definitions: *A* permutation matrix *is a matrix in which each row* (resp., column) *contains a single entry of value 1. A* Hamiltonian matrix *is a permutation matrix which corresponds to a single directed cycle* (it follows that the corresponding directed graph consists of a single Hamiltonian cycle). *An* n^3*-by-*n^3 *matrix* M *is called* useful *if it contains an n-by-n Hamiltonian submatrix and all other entries in* M *are 0.*
- Prover: *Let* C *be a Hamiltonian cycle in* G, *in case such exists.*

  case 1: *M is useful. Let H denote its Hamiltonian n-by-n submatrix.*
  - *the prover reveals all entries in M which are not in H.*
  - *the prover finds a 1-1 mapping, π_1, of V to the rows of H and a 1-1 mapping, π_2, of V to the columns of H so that the edges of C are mapped to the 1-entries of H.*
    (Directed pairs of vertices of G, being edges or not, are mapped in the natural manner; that is (u,v) is mapped to the matrix entry $(\pi_1(u),\pi_2(v))$. The mapping-pair (π_1,π_2) is an isomorphism of C to H. Actually, we should specify one isomorphism among the n possible ones.)
  - *the prover reveals the entries corresponding to non-edges of G.*
    (The correspondence is by the above mappings.)
  - *the prover outputs the mapping pair* (π_1,π_2) (as a certificate).

  case 2: *M is not useful. In this case the prover reveals all entries of M.*
- Verifier:

  case 1: *The prover has not revealed all entries in M. Let (π_1,π_2) be the certificate sent/output by the prover. The verifier checks that all entries in M which do not have an image under (π_1,π_2) in E are revealed and are indeed zero. That is, the verifier accepts if all matrix entries, except for the entries in $\{(\pi_1(u),\pi_2(v)) : (u,v)\in E\}$, are revealed and all revealed bits are 0.*

case 2: *The prover has revealed all of M. In this case the verifier accepts iff M is* NOT *useful.*

The following fact is instrumental for the analysis of Construction 4.10.7.

Fact 4.10: $\mathsf{Pr}(M \text{ is useful}) = \Omega(n^{-3/2})$.

proof: With probability $\Omega(1/\sqrt{n})$, the matrix M contains exactly n entries of value 1. Considering any row of M, observe that with probability at most $\binom{n^3}{2} \cdot (n^{-5})^2 < n^{-4}$ this row contains more than a single 1-entry. Thus, with probability $\Omega(1/\sqrt{n})$, the matrix M contains an n-by-n permutation matrix and all its other entries are 0. The fact follows by observing that there are $n!$ (n-by-n) permutation matrices and $(n-1)!$ of them are Hamiltonian matrices. ■

Proposition 4.10.8 *There exists a* (perfect) *zero-knowledge Hidden Bits proof system for Graph Hamiltonicity. Furthermore, the prover may be implemented by a polynomial-time machine which gets an Hamiltonian cycle as auxiliary input.*

proof: We start by demonstrating a gap in the acceptance probability of the verifier of Construction 4.10.7. Firstly, we claim that in case G is Hamiltonian and the prover follows the program then the verifier accepts no matter which matrix M appears as common random-string. The claim follows easily by observing that in Case 1 the mapping-pair maps the Hamiltonian cycle of G to the Hamiltonian cycle of H and, since the latter contains the only 1-entries in M, all non-edges of G are mapped to 0-entries of M. (In Case 2 the claim is trivial.) We remark that the prover's actions can be implemented in polynomial-time when given an Hamiltonian cycle of G as auxiliary input. Specifically, all that the prover needs to do is check if M is useful and find an isomorphism between two given n-vertex cycles.

Next, suppose that G is non-Hamiltonian. By Fact 4.10, with probability at least $\Omega(n^{-3/2})$, the matrix M is useful and let H denote its n-by-n Hamiltonian submatrix. In this case the prover must reveal all entries not in the submatrix H since mapping $V \times V$ to any other n-by-n submatrix of M will reveal 1-entries (in the rest of M). Thus, the prover must output a mapping pair (π_1, π_2) so that $\pi_1(V) \times \pi_2(V) = H$. Also, each non-edge of G must be mapped to a 0-entry of H (or else the verifier will reject). It follows that the preimage of each 1-entry in H must be an edge in G, which implies that G has a Hamiltonian cycle (in contradiction to our hypothesis). We conclude that in case G is non-Hamiltonian, it is rejected with probability $\Omega(n^{-3/2})$.

Finally, we show that the above prover is zero-knowledge. This is done by constructing a simulator that on input a graph G randomly selects an n^3-by-n^3 matrix, denoted M, with distribution as in the common random-string (i.e., each entry being 1 with probability n^{-5}). If M is not useful then the simulator outputs $(G, M, \{1, ..., n^3\}^2)$ (i.e., all bits are revealed with values as in M and no certificate is given). Otherwise, the prover selects uniformly a pair of 1-1 mappings (π_1, π_2) so that $\pi_i : V \mapsto \{1, ..., n^3\}$, for $i = 1, 2$. The prover outputs $(G, 0^{n^6-|E|}, I, (\pi_1, \pi_2))$, where $I \stackrel{\rm def}{=} \{1, ..., n^3\}^2 - \{(\pi_1(u), \pi_2(v)) : (u, v) \in E\}$. The reader can easily verify that the output distribution of the simulator is identical to the distribution seen by the verifier. ∎

Using Propositions 4.10.8 and 4.10.5 and Remark 4.10.6, we conclude

Theorem 4.10.9 *Assuming the existence of one-way permutations, each language in $\mathcal{NP}$ has a zero-knowledge non-interactive proof system. Furthermore, assuming the existence of families of trapdoor permutations for which membership in the family can be decided in $\mathcal{BPP}$, each language in $\mathcal{NP}$ has a zero-knowledge non-interactive proof system in which the prover can be implemented by a probabilistic polynomial-time machine which gets an NP-witness as auxiliary input.*

4.10.3 Extensions: many assertions of varying length

The definitions presented in Section 4.10.1 are restricted in two ways. Firstly, they consider only the proving of one assertion relative to the common random-string, and furthermore the common random-string is allowed to be longer than the assertion (though polynomial in length of the assertion). A stronger definition provided below allows proving poly(n)-many assertions, each of poly(n)-length, using the same n-bit long common random-string.

We first note that it suffices to treat the case in which the number of assertions is unbounded but the length of each assertion is a-priori bounded. Specifically, for any $\varepsilon > 0$, it suffices to consider the case where poly(n)-many assertions, each of length n^ε, need to be proven relative to the same n-bit long common random-string. The reason for this is that we can reduce, in a "zero-knowledge manner", any NP-assertion of length poly(n) into a sequence of poly(n)-many NP-assertions, each of length n^ε. For example, first we reduce the original NP-assertion to an assertion regarding the 3-colorability of a poly(n)-vertex graph. Next, we use a commitment scheme with commitments of length n^ε, in order to commit to the coloring of each vertex. Finally, for each edge, we (invoke the proof system to) prove that the corresponding two commitments are to two different values in $\{1, 2, 3\}$.

We now turn to the actual definitions. First we note that nothing needs to be changed regarding the definition of non-interactive proof systems (Definition 4.10.1). We still require ability to be convinced by valid assertions and "protection" from false assertions. Alas a minor technical difference is that, while in Definition 4.10.1 we have denoted by n the length of the assertion and considered a common random-string of length poly(n), here we let n denote the length of the common random-string used for assertions of length n^ε. We call ε the *fundamental constant* of the proof system. In contrast, the definition of zero-knowledge has to be extended to handle a sequence of proofs.

Definition 4.10.10 (non-interactive zero-knowledge – extended): *A non-interactive proof system, (P, V), with fundamental constant ε, for a language L is* strongly zero-knowledge *if* *there exists a probabilistic polynomial-time algorithm M such that the ensembles*

$$\{((x_1, ..., x_m), U_n, (P(x_1, U_n), ..., P(x_m, U_n)))\}_{x_1,...,x_m \in L_{n^\epsilon}} \quad \textit{and} \quad \{M(x_1, ..., x_m)\}_{x_1,...,x_m \in L_{n^\epsilon}}$$

are computationally indistinguishable, where $m = \text{poly}(n)$ and $L_\ell \stackrel{\text{def}}{=} L \cap \{0,1\}^\ell$.

We now turn to the construction of strong zero-knowledge (non-interactive) proof systems. The underlying idea is to facilitate the simulation by potentially proving a fictitious assertion regarding a portion of the common random-string. The assertion that will be potentially proven about this portion will have the following properties

1. The assertion holds for a negligible fraction of the strings of the same length. Thus, adding this potential ability does not significantly effect the soundness condition.

2. Strings satisfying the assertion are computationally indistinguishable from uniformly distributed strings of the same length. Thus, it will be OK for the simulator to use such strings rather than uniformly chosen ones (used in the real proof system).

3. The decision problem for the assertion is in $\mathcal{NP}$. This will allow a reduction to an NP-complete problem.

An immediate assertion, concerning strings, which comes to mind is being produced by a pseudorandom generator.

Construction 4.10.11 (strong zero-knowledge non-interactive proof systems): *Let $G : \{0,1\}^\ell \mapsto \{0,1\}^{2\ell}$, L_1 be an NP-complete language, and (P, V) be a non-interactive proof system for L_1. Furthermore, suppose that (P, V) uses a common random-string of length $n - 2\ell$ for assertions of length* poly(ℓ) *and that P takes as auxiliary input an NP-witness for membership in L_1. Following is a specification of a non-interactive system for $L \in \mathcal{NP}$:*

- Common Input: $x \in \{0,1\}^\ell$.
- Common Random-String: $r = (p, s)$, *where* $p \in \{0,1\}^{2\ell}$ *and* $s \in \{0,1\}^{n-2\ell}$.
- Prover:
  - *Using a standard reduction of L_2 to L_1, reduces (x, p) to $y \in \{0,1\}^{\text{poly}(\ell)}$, where*

    $$L_2 \stackrel{\text{def}}{=} \{(x, p) : x \in L \bigvee \exists w \in \{0,1\}^{|x|} \ \textit{s.t.} \ \ G(w) = p\}$$

    In case the prover is given a witness u for $x \in L$, it reduces u to a witness, denoted w, for $y \in L_1$.
  - *Invokes P with common input y, auxiliary input w and common random-string s, getting output π which it outputs/sends.*

- Verifier:
  - *Reduces (x, p) into y using the same standard reduction of L_2 to L_1.*
  - *Invokes V with common input y, common random-string s and prover's output π, and decide as V does.*

Proposition 4.10.12 *Let (P, V) be as above, G be a pseudorandom generator, and* $\text{poly}(\ell) = n^{\varepsilon}$. *Furthermore, suppose that P is zero-knowledge and that when given an NP-witness as auxiliary input it can be implemented in probabilistic polynomial-time. Then, Construction 4.10.11 constitutes a zero-knowledge non-interactive proof system for L, with fundamental constant ε. Furthermore, the prover may be implemented by a probabilistic polynomial-time machine which gets an NP-witness as auxiliary input.*

proof sketch: The completeness and efficiency claims for the new prover follow immediately from the hypothesis concerning (P, V). The soundness condition follows by observing that the probability that p is in the range of G is at most $2^{-\ell}$. To prove the zero-knowledge property, we construct a simulator as follows. The simulator uniformly selects $u' \in \{0,1\}^{\ell}$ and $s \in \{0,1\}^{n-2\ell}$, sets $p = G(u')$, and follows the prover's program except that it uses u' as the NP-witness for $(x, p) \in L_2$. Namely, the simulator reduces $(x, p) \in L_1$ to $y \in L_1$ along with reducing the NP-witness u' to a witness w' (for y). Next, the simulator invokes P with common input y, auxiliary input w' and common random-string s. Note that the efficiency of the simulator relies on the efficient implementation of P. To prove that the simulator's output is computationally indistinguishable from the verifier's view we combine the following two observations:

1. The distribution of the common random-string is very different in the two cases. Yet, by the pseudorandomness of G this difference is computationally indistinguishable. Thus, we may consider the verifier's view in case the common random-string is selected exactly as in the simulation (but the prover acts as in Construction 4.10.11).

2. The zero-knowledge property of P implies that P is witness-indistinguishable (see Section 4.6). Thus, one cannot distinguish the case P uses a witness for $x \in L$ (as in Construction 4.10.11) from the case P uses as witness a seed for the pseudorandom sequence p (as done in the simulator). The same holds when repeating the process polynomially-many times.

■

Using Theorem 4.10.9 and Proposition 4.10.12, we obtain

Theorem 4.10.13 *Assuming the existence of families of trapdoor permutations for which membership in the family can be decided in $\mathcal{BPP}$, each language in $\mathcal{NP}$ has a* strong *zero-knowledge non-interactive proof system. Furthermore, the prover can be implemented by a probabilistic polynomial-time machine which gets an NP-witness as auxiliary input.*

4.11 * Multi-Prover Zero-Knowledge Proofs

In this section we consider an extension of the notion of an interactive proof system. Specifically, we consider the interaction of a verifier with several (say, two) provers. The provers may share an a-priori selected strategy, but it is assumed that they cannot interact with each other during the time period in which they interact with the verifier. Intuitively, the provers can coordinate their strategies prior to, but not during, their interrogation by the verifier.

The notion of multi-prover interactive proof plays a fundamental role in complexity theory. This aspect is not addressed here (but rather postponed to Section [missing(eff-pcp.sec)]). In the current section we merely address the zero-knowledge aspects of multi-party interactive proofs. Most importantly, the multi-prover model enables the construction of (perfect) zero-knowledge proof systems for $\mathcal{NP}$, *independent of any* complexity theoretic (or other) *assumptions*. Furthermore, these proof systems can be extremely efficient. Specifically, the on-line computations of all parties can be performed in polylogarithmic time (on a RAM).

4.11.1 Definitions

For sake of simplicity we consider the two-prover model. We remark that more provers do not offer any essential advantages (and specifically, none that interest us in this section). Loosely speaking, a two-prover interactive proof system is a three party protocol, where two parties are provers and the additional party is a verifier. The only interaction allowed in this model is between the verifier and each of the provers. In particular, a prover does not "know" the contents of the messages sent by the verifier to the other prover. The provers do however share a random input tape, which is (as in the one-prover case) "beyond the reach" of the verifier. The two-prover setting is a special case of the *two-partner model* described below.

4.11.1.1 The two-partner model

The two-party model consists of two *partners* interacting with a third party, called *solitary*. The two partners can agree on their strategies beforehand, and in particular agree on a common uniformly chosen string. Yet, once the interaction with the solitary begins, the partners can no longer exchange information. The following definition of such an interaction extends Definitions 4.2.1 and 4.2.2.

Definition 4.11.1 (two-partner model): *The* two-partner model *consists of three interactive machines, two are called* partners *and the third is called* solitary, *which are linked and interact as hereby specified.*

- *The input-tapes of all three parties coincide, and its contents is called the* common input.

- *The random-tapes of the two partners coincide, and is called the* partners' random-tape. *(The solitary has a separate random-tape.)*
- *The solitary has two pairs of communication-tapes and two switch-tapes; instead of a single pair of communication-tapes and a single switch-tape (as in Definition 4.2.1).*
- *Both partners have the same identity and the solitary has an opposite identity (see Definitions 4.2.1 and 4.2.2).*
- *The first (resp., second) switch-tape of the solitary coincides with the switch-tape of the first (resp., second) partner, the first (resp., second) read-only communication-tape of the solitary coincides with the write-only communication-tape of the first (resp., second) partner and vice versa.*
- *The* joint computation *of the three parties, on a common input x, is a sequence of triplets. Each triplet consists of the local configuration of each of the three machines. The behaviour of each partner-solitary pair is as in the definition of the joint computation of a pair of interactive machines.*
- **Notation:** *We denote by $\langle P_1, P_2, S\rangle(x)$ the output of the solitary S after interacting with the partners P_1 and P_2, on common input x.*

4.11.1.2 Two-prover interactive proofs

A two-prover interactive proof system is now defined analogously to the one-prover case (see Definitions 4.2.4 and 4.2.6).

Definition 4.11.2 (two-prover interactive proof system): *A triplet of interactive machines, (P_1, P_2, V), in the two-partner model is called an* proof system for a language *L if the machine V* (called verifier) *is probabilistic polynomial-time and the following two conditions hold*

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

$$\mathsf{Pr}\left(\langle P_1, P_2, V\rangle(x)=1\right) \geq \frac{2}{3}$$

- Soundness: *For every $x \notin L$ and every pair of partners (B_1, B_2),*

$$\mathsf{Pr}\left(\langle B_1, B_2, V\rangle(x)=1\right) \leq \frac{1}{3}$$

As usual, the error probability in the completeness condition can be reduced (from $\frac{1}{3}$) up to $2^{-\mathrm{poly}(|x|)}$, by *sequentially* repeating the protocol sufficiently many times. We stress that error reduction via *parallel* repetitions is **not** known to work *in general*.

The notion of zero-knowledge (for multi-prove systems) remains exactly as in the one-prover case. Actually, the definition of perfect zero-knowledge may even be made more strict by requiring that the simulator never fails (i.e., never outputs the special symbol $\perp$). Namely,

Definition 4.11.3 *We say that a* (two-prover) *proof system* (P_1, P_2, V) *for a language* L *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 random variables* $\langle P_1, P_2, V^* \rangle(x)$ *and* $M^*(x)$ *are identically distributed.*

Extension to the auxiliary-input (zero-knowledge) model is straightforward.

4.11.2 Two-Senders Commitment Schemes

The thrust of the current section is in a method for constructing perfect zero-knowledge two-prover proof systems for every language in $\mathcal{NP}$. This method makes essential use of a commitment scheme *for two senders and one receiver* that posses "information theoretic" secrecy and unambiguity properties. We stress that it is impossible to simultaneously achieve "information theoretic" secrecy and unambiguity properties in the single sender model.

4.11.2.1 A Definition

Loosely speaking, a two-sender commitment scheme is an efficient *two-phase* protocol for the two-partner model, through which the partners, called *senders*, can commit themselves to a *value* so that the following two conflicting requirements are satisfied.

1. *Secrecy*: At the end of the *commit phase* the solitary, called *receiver*, does not gain any *information* of the senders' value.
2. *Unambiguity*: Suppose that the commit phase is successfully terminated. Then if later the senders can perform the *reveal phase* so that the receiver accepts the value 0 with probability p then they cannot perform the *reveal phase* so that the receiver accepts the value 1 with probability substantially bigger than $1-p$. (Due to the secrecy requirement and the fact that the senders are computationally unbounded, for every p, the senders can always conduct the commit phase so that they can later reveal the value 0 with probability p and the value 1 with probability $1-p$.)

Instead of presenting a general definition, we restrict our attention to the special case of two-sender commitment schemes in which only the first sender (and the receiver) takes part in the commit phase, whereas only the second sender takes part in the reveal phase. Furthermore, we assume, without loss of generality, that in the reveal phase the (second) sender sends the contents of the joint random-tape (used by the first sender in the commit phase) to the receiver.

Definition 4.11.4 (two-sender bit commitment)*: A* two-sender bit commitment scheme *is a triplet of probabilistic polynomial-time interactive machines, denoted (S_1, S_2, R), for the two-partner model satisfying:*

- Input Specification: *The common input is an integer n presented in unary, called the* security parameter*. The two partners, called the* senders*, have an auxiliary private input $v \in \{0,1\}$.*
- Secrecy: *The 0-commitment and the 1-commitment are identically distributed. Namely, for every probabilistic (not necessarily polynomial-time) machine R^* interacting with the first sender (i.e., S_1), the random variables $\langle S_1(0), R^* \rangle(1^n)$ and $\langle S_1(1), R^* \rangle(1^n)$ are identically distributed.*
- Unambiguity: *Preliminaries. For simplicity $v \in \{0,1\}$ and $n \in \mathbb{N}$ are implicit in all notations.*
  - *As in Definition 4.4.1, a* receiver's view of an interaction *with the (first) sender, denoted $(r, \overline{m})$, consists of the random coins used by the receiver, denoted r, and the sequence of messages received from the (first) sender, denoted $\overline{m}$.*
  - *Let $\sigma \in \{0,1\}$. We say that the string s is a* possible σ-opening *of the receiver's view $(r, \overline{m})$ if $\overline{m}$ describes the messages received by R when R uses local coins r and interacts with machine S_1 which uses local coins s and input $(\sigma, 1^n)$.*
  - *Let S_1^* be an arbitrary program for the first sender. Let p be a real, and $\sigma \in \{0,1\}$. We say that p is an* upper bound on the probability of a σ-opening of the receiver's view of the interaction with S_1^* *if for every random variable X, which is statistically independent of the receiver's coin tosses, the probability that X is a possible σ-opening of the receiver's view of an interaction with S_1^* is at most p.*
  (The probability is taken over the coin tosses of the receiver, the strategy S_1^* and the random variable X.)
  - *Let S_1^* be as above, and, for each $\sigma \in \{0,1\}$, let p_σ be an upper bound on the probability of a σ-opening of the interaction with S_1^*. We say that* the receiver's view of the interaction with S_1^* is unambiguous *if $p_0 + p_1 \leq 1 + 2^{-n}$.*

  The unambiguity requirement *asserts that, for every program for the first sender, S_1^*, the receiver's interaction with S_1^* is unambiguous.*

In the formulation of the unambiguity requirement, the random variables X represent possible strategies of the second sender. These strategies may depend on the random input that is shared by the two senders, but is independent of the receiver's random coins (since information on these coins, if at all, is only sent to the first sender). Actually, the highest possible value of $p_0 + p_1$ is attainable by deterministic strategies for both senders. Thus, it suffices to consider an arbitrary deterministic strategy S_1^* for the first sender and a fixed

σ-opening, denoted s^σ, for the second sender (for each $\sigma \in \{0,1\}$). In this case, the probability is taken only over the receiver coin tosses and we can strengthen the unambiguity condition as follows:

> (strong unambiguity condition) *for every deterministic strategy S_1^*, and every pair of strings (s^0, s^1), the probability that* **for both** $\sigma = 0, 1$ *the string s^σ is a σ-opening of the receiver's view of the interaction with S_1^* is bounded above by 2^{-n}.*

In general, in case the sender employ randomized strategies, they determine for each possible coin-tossing of the receiver a pair of probabilities corresponding to their success in a 0-opening and a 1-opening. The unambiguity condition asserts that the average of these pairs, taken over all possible receiver's coin tosses is a pair which sums-up to at most $1 + 2^{-n}$. Intuitively, this means that the senders cannot do more harm than deciding at random (possibly based also on the receiver's message to the first sender) whether to commit to 0 or to 1. Both secrecy and unambiguity requirements are information theoretic (in the sense that no computational restrictions are placed on the adversarial strategies). We stress that we have implicitly assumed that the reveal phase takes the following form:

1. the second sender sends to the receiver the initial private input, v, and the random coins, s, used by the first sender in the commit phase;
2. the receiver verifies that v and s (together with the private 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_1 and R).

4.11.2.2 A Construction

By the above conventions, it suffices to explicitly describe the commit phase (in which only the first sender takes part).

Construction 4.11.5 (two-sender bit commitment):

- Preliminaries: *Let π_0, π_1 denote two permutations over $\{0,1,2\}$ so that π_0 is the identity permutation and π_1 is a permutation consisting of a single transposition, say $(1,2)$. Namely, $\pi_1(1)=2$, $\pi_1(2)=1$ and $\pi_1(0)=0$.*
- Common input: *the security parameter n (in unary).*
- A convention: *Suppose that the contents of the senders' random-tape encodes a uniformly selected $\overline{s}=s_1 \cdots s_n \in \{0,1,2\}^n$.*
- Commit Phase:
  1. *The receiver uniformly selects $\overline{r} = r_1 \cdots r_n \in \{0,1\}^n$ and sends $\overline{r}$ to the first sender.*

2. *To commit to a bit σ, the first sender computes $c_i \stackrel{\text{def}}{=} \pi_{r_i}(s_i) + \sigma \bmod 3$, for each i, and sends $c_1 \cdots c_n$ to the receiver.*

We remark that the *second* sender could have opened the commitment either way if he had known $\overline{r}$ (sent by the receiver to the *first* sender). The point is that the second sender does not "know" $\overline{r}$, and this fact drastically limits its ability to cheat.

Proposition 4.11.6 *Construction 4.11.5 constitutes a two-sender bit commitment scheme.*

Proof: The security property follows by observing that for every choice of $\overline{r} \in \{0,1\}^n$, the message sent by the first sender is uniformly distributed over $\{0,1,2\}^n$.

The unambiguity property is proven by contradiction. As a motivation, we first consider the execution of the above protocol when n equals 1 and show that it is impossible for the two senders to be *always* able to open the commitments both ways. Consider two messages, $(0, s^0)$ and $(1, s^1)$, sent by the second sender in the reveal phase so that s^0 is a possible 0-opening and s^1 is a possible 1-opening, both with respect to the receiver's view. We stress that these messages are sent obliviously of the random coins of the receiver, and hence must match all possible receiver's views (or else the opening does not always succeed). It follows that for each $r \in \{0,1\}$, both $\pi_r(s^0)$ and $\pi_r(s^1) + 1 \bmod 3$ must fit the message received by the receiver (in the commit phase) in response to message r sent by it. Hence, $\pi_r(s^0) \equiv \pi_r(s^1) + 1 \pmod 3$ holds, for each $r \in \{0,1\}$. Contradiction follows since no two $s^0, s^1 \in \{0,1,2\}$ can satisfy both $\pi_0(s^0) \equiv \pi_0(s^1) + 1 \pmod 3$ and $\pi_1(s^0) \equiv \pi_1(s^1) + 1 \pmod 3$. (The reason being that the first equality implies $s^0 \equiv s^1 + 1 \pmod 3$ which combined with the second equality yields $\pi_1(s_1 + 1 \bmod 3) \equiv \pi_1(s_1) + 1 \pmod 3$, whereas for every $s \in \{0,1,2\}$ it holds that $\pi_1(s + 1 \bmod 3) \not\equiv \pi_1(s) + 1 \pmod 3$.)

We now turn to the actual proof of the unambiguity property. We first observe that if there exists a program S_1^* so that the receiver's interaction with S_1^* is ambiguous, then there exists also such a deterministic program. Actually, the program is merely a function, denoted f, mapping n-bit long strings into sequences in $\{0,1,2\}^n$. Likewise, the (0-opening and 1-opening) strategies for the second sender can be assumed, without loss of generality, to be deterministic. Consequently, both strategies consist of constant sequences, denoted $\overline{s}^0$ and $\overline{s}^1$, and both can be assumed (with no loss of generality) to be in $\{0,1,2\}^n$.

For each $\sigma \in \{0,1\}$, let p_σ denote the probability that the sequence $\overline{s}^\sigma$ is a possible σ-opening of the receiver's view $(U_n, f(U_n))$, where U_n denotes a random variable uniformly distributed over $\{0,1\}^n$. The contradiction hypothesis implies that $p_0 + p_1 > 1 + 2^{-n}$. Put in other words, $|R^0| + |R^1| \geq 2^n + 2$, where R^σ denotes the set of all strings $\overline{r} \in \{0,1\}^n$ for which the sequence $\overline{s}^\sigma$ is a possible σ-opening of the receiver's view $(\overline{r}, f(\overline{r}))$. Namely,

$$R^\sigma = \{\overline{r} : (\forall i)\ f_i(\overline{r}) \equiv \pi_{r_i}(s_i^\sigma) + \sigma \pmod 3\}$$

where $\overline{r} = r_1 \cdots r_n$, $\overline{s}^\sigma = s_1^\sigma \cdots s_n^\sigma$, and $f(\overline{r}) = f_1(\overline{r}) \cdots f_n(\overline{r})$. We are going to refute the contradiction hypothesis by showing that the intersection of the sets R^0 and R^1 cannot contain more than a single element.

Claim 4.11.6.1: Let R^0 and R^1 as defined above. Then $|R^0 \cap R^1| \leq 1$.

proof: Suppose, on the contrary, that $\overline{\alpha}, \overline{\beta} \in R^0 \cap R^1$ (and $\overline{\alpha} \neq \overline{\beta}$). Then, there exist an i such that $\alpha_i \neq \beta_i$, and without loss of generality $\alpha_i = 0$ (and $\beta_i = 1$). By the definition of R^σ it follows that

$$\begin{array}{rcl} f_i(\overline{\alpha}) & \equiv & \pi_0(s_i^0) \pmod 3 \\ f_i(\overline{\beta}) & \equiv & \pi_1(s_i^0) \pmod 3 \\ f_i(\overline{\alpha}) & \equiv & \pi_0(s_i^1) + 1 \pmod 3 \\ f_i(\overline{\beta}) & \equiv & \pi_1(s_i^1) + 1 \pmod 3 \end{array}$$

Contradiction follows as in the motivating discussion. □

This completes the proof of the proposition. ■

We remark that Claim 4.11.6.1 actually yields the strong unambiguity condition (presented in the discussion following Definition 4.11.4). More importantly, we remark that the proof extends easily to the case in which many instances of the protocol are executed in parallel; namely, the parallel protocol constitutes a two-sender multi-value (i.e., string) commitment scheme.

> Author's Note: *The last remark should be elaborated significantly. In addition, it should be stressed that the claim holds also when the second sender is asked to reveal only some of the commitments, as long as this request is indepdendent of the coin tosses used by the receiver during the commit phase.*

4.11.3 Perfect Zero-Knowledge for NP

Two-prover perfect zero-knowledge proof systems for any language in $\mathcal{NP}$ follow easily by modifying Construction 4.4.6. The modification consists of replacing the bit commitment scheme, used in Construction 4.4.6, by the two-sender bit commitment scheme of Construction 4.11.5. Specifically, the modified proof system for Graph Coloring proceeds as follows.

Two-prover atomic proof of Graph Coloring:

- The first prover uses the prover's random tape to determine a permutation of the coloring. In order to commit to each of the resulting colors, the first prover invokes (the commit phase of) a two-sender bit commitment, setting the security parameter to be the number of vertices in the graph. (The first prover plays the role of the first sender whereas the verifier plays the role of the receiver.)

- The verifier uniformly selects an edge and sends it to the second prover.

- The second prover reveals the colors of the endpoints of the required edge, by sending the portions of the prover's random-tape used in the corresponding instance of the commit phase.

We now remark on the properties of the above protocol. As usual, one can see that the provers can always convince the verifier of valid claims (i.e., the completeness condition hold). Using the unambiguity property of the two-sender commitment scheme we can think of the first prover as selecting at random, with arbitrary probability distribution, a color assignment to the vertices of the graph. We stress that this claim holds although many instances of the commit protocol are performed concurrently (see remark above). If the graph is not 3-colored than each of the possible color assignments chosen by the first prover is illegal, and a weak soundness property follows. Yet, by executing the above protocol polynomially many times, even in parallel, we derive a protocol satisfying the soundness requirement. We stress that the fact that parallelism is effective here (as means for decreasing error probability) follows from the unambiguity property of two-sender commitment scheme and not from a general "parallel composition lemma" (which is not valid in the two-prover setting).

Author's Note: *The last sentence refers to a false claim by which the error probability of a protocol in which a basic protocol is repeated t times in parallel is at most p^t, where p is the error probability of the basic protocol. Interestingly, Ran Raz has recently proven a general "parallel composition lemma" of slightly weaker form: the error probability indeed decreases exponentially in t (but the base is indeed bigger than p).*

We now turn to the zero-knowledge aspects of the above protocol. It turns out that this part is much easier to handle than in all previous cases we have seen. In the construction of the simulator we take advantage on the fact that it is playing the role of both provers and hence the unambiguity of the commitment scheme does not apply. Specifically, the simulator, playing the role of both senders, can easily open each commitment any way it wants. (Here we take advantage on the specific structure of the commitment scheme of Construction 4.11.5.) Details follow.

Simulation of the atomic proof of Graph Coloring

- The simulator generates random "commitments to nothing". Namely, the simulator invokes the verifier and answers its messages by uniformly chosen strings.

- Upon receiving the query-egde (u, v) from the verifier, the simulator uniformly selects two different colours, ϕ_u and ϕ_v, and opens the corresponding commitments so that to reveal this values. The simulator has no difficulty to do so since, unlike the second prover, it knows the messages sent by the verifier in the commit phase. (Given the receiver's view, $(r_1 \cdots r_n, c_1 \cdots c_n)$, of the commit phase, a 0-opening is computed by setting $s_i = \pi_{r_i}^{-1}(c_i)$ whereas a 1-opening is computed by setting $s_i = \pi_{r_i}^{-1}(c_i - 1)$, for all i.)

We now remark that the entire argument extends trivially to the case in which polynomially many instances of the protocol are performed concurrently.

Efficiency improvement. A dramatic improvement in the efficiency of two-prover (perfect) zero-knowledge proofs for $\mathcal{NP}$, can be obtained by using the techniques described in Section [missing(eff-pcp.sec)]. In particular, such a proof system with constant error probability, can be implemented in probabilistic polynomial-time, so that the number of bits exchanged in the interaction is logarithmic. Furthermore, the verifier is only required to use logarithmically many coin tosses. The error can be reduced to 2^{-k} by repeating the protocol sequentially for k times. In particular negligible error probability is achieved in polylogarithmic communication complexity. We stress again that error reduction via parallel repetitions is not known to work in general, and in particular is not known to work in this specific case.

> Author's Note: *Again, the last statement is out of date and recent results do allow to reduce the error probability without increasing the number of rounds.*

4.11.4 Applications

Multi-prover interactive proofs are useful only in settings in which the "proving entity" can be separated and its parts kept ignorant of one another during the proving process. In such cases we get perfect zero-knowledge proofs without having to rely on complexity theoretic assumptions. In other words, general widely believed mathematical assumptions are replaced by physical assumptions concerning the specific setting.

A natural application is to the problem of identification, and specifically the identification of a *user* at some *station*. In Section 4.7 we discuss how to reduce identification to a zero-knowledge proof of knowledge (for some NP relation). The idea is to supply each user with two smart-cards, implementing the two provers in a two-prover zero-knowledge proof of knowledge. These two smart-cards have to be inserted in two different slots of the station, and this guarantees that the smart-cards cannot communicate one with another. The station will play the role of the verifier in the zero-knowledge proof of knowledge. This way the station is protected against impersonation, whereas the users are protected against pirate stations which may try to extract knowledge from the smart-cards (so to enable impersonation by its agents).

4.12 Miscellaneous

4.12.1 Historical Notes

Interactive proof systems were introduced by Goldwasser, Micali and Rackoff [GMR85]. (Earlier versions of this paper date to early 1983. Yet, the paper, being rejected three times from major conferences, has first appeared in public only in 1985, concurrently to the paper of Babai [B85].) A restricted form of interactive proofs, known by the name *Arthur Merlin Games*, was introduced by Babai [B85]. (The restricted form turned out to be equivalent in power – see Section [missing(eff-ip.sec)].) The interactive proof for Graph Non-Isomorphism is due to Goldreich, Micali and Wigderson [GMW86].

The concept of zero-knowledge has been introduced by Goldwasser, Micali and Rackoff, in the same paper quoted above [GMR85]. Their paper contained also a perfect zero-knowledge proof for Quadratic Non Residuousity. The perfect zero-knowledge proof system for Graph Isomorphism is due to Goldreich, Micali and Wigderson [GMW86]. The latter paper is also the source to the zero-knowledge proof systems for all languages in $\mathcal{NP}$, using any (nonunifomly) one-way function. (Brassard and Crépeau have *later* constructed alternative zero-knowledge proof systems for $\mathcal{NP}$, using a *stronger* intractability assumption, specifically the intractability of the Quadratic Residuousity Problem.)

The cryptographic applications of zero-knowledge proofs were the very motivation for their presentation in [GMR85]. Zero-knowledge proofs were applied to solve cryptographic problems in [FMRW85] and [CF85]. However, many more applications were possible once it was shown how to construct zero-knowledge proof systems for every language in $\mathcal{NP}$. In particular, general methodologies for the construction of cryptographic protocols have appeared in [GMW86,GMW87].

Credits for the advanced sections

The results providing upper bounds on the complexity of languages with perfect zero-knowledge proofs (i.e., Theorem 4.5.5) are from Fortnow [For87] and Aiello and Hastad [AH87]. The results indicating that one-way functions are necessary for non-trivial zero-knowledge are from Ostrovsky and Wigderson [OWistcs93]. The negative results concerning parallel composition of zero-knowledge proof systems (i.e., Proposition 4.5.6 and Theorem 4.5.8) are from [GKr89b].

The notions of witness indistinguishability and witness hiding, were introduced and developed by Feige and Shamir [FSwitness].

Author's Note: *FSwitness has appeared in STOC90.*

The concept of proofs of knowledge originates from the paper of Goldwasser, Micali and Rackoff [GMR85]. First attempts to provide a definition to this concept appear in Fiat, Feige and Shamir [FFS87] and Tompa and Woll [TW87]. However, the definitions provided in both [FFS87,TW87] are not satisfactory. The issue of defining proofs of knowledge has been extensively investigated by Bellare and Goldreich [BGknow], and we follow their suggestions. The application of zero-knowledge proofs of knowledge to identification schemes was discovered by Feige, Fiat and Shamir [FFS87]. The Fiat-Shamir Identification scheme is based on the zero-knowledge proof for Quadratic Residousity of Goldwasser, Micali and Rackoff [GMR85].

Author's Note: *Reference Fiat and Shamir in Crypto86(?)*

Computationally sound proof systems (i.e., arguments) were introduced by Brassard, Chaum, and Crépeau [BCC87]. Their paper also presents perfect zero-knowledge arguments for $\mathcal{NP}$ based on the intractability of factoring. Naor et. al. [NOVY92] showed how to construct perfect zero-knowledge arguments for $\mathcal{NP}$ based on any one-way permutation, and Construction 4.8.3 is taken from their paper. The polylogarithmic-communication argument system for $\mathcal{NP}$ (of Subsection 4.8.4) is due to Kilian [K92].

Author's Note: *NOVY92 has appeared in Crypto92, and K92 in STOC92.*

The round-efficient zero-knowledge proof systems for $\mathcal{NP}$, based on any clawfree collection, is taken from Goldreich and Kahan [GKa89]. The round-efficient zero-knowledge arguments for $\mathcal{NP}$, based on any one-way function, uses ideas of Feige and Shamir [FSconst] (yet, their original construction is different).

Non-interactive zero-knowledge proof systems were introduced by Blum, Feldman and Micali. The constructions presented in Section 4.10 are due to Feige, Lapidot and Shamir [FLSnizk].

Author's Note: *BFM has appeared in STOC88, and FLSnizk in FOCS90*

Multi-prover interactive proofs were introduced by Ben-Or, Goldwasser, Kilian and Wigderson [BGKW88]. Their paper also presents a perfect zero-knowledge two-prover proof system for $\mathcal{NP}$. The perfect zero-knowledge two-prover proof for $\mathcal{NP}$, presented in Section 4.11, follows their ideas but explicitly states the properties of the two-sender commitment scheme in use. Consequently, we observe that (sufficiently many) parallel repetitions of this *specific* proof system *does* decease the error probability to a negligible one. This observation escaped Feige, Lapidot and Shamir, who, being aware of the problematics of parallel repetitions (of general multi-prover systems), suggested an alternative construction [FLSmip].

Author's Note: *add FLSmip reference*

4.12.2 Suggestion for Further Reading

For further details on interactive proof systems see Section [missing(eff-ip.sec)]. The interested reader may also want to look at Section [missing(eff-pcp.sec)], which is devoted to probabilistically checkable proof systems, which in turn are an off-spring of interactive proof systems.

A uniform-complexity treatment of zero-knowledge was given by Goldreich [Guniform]. In particular, it is shown how to use (uniformly) one-way functions to construct interactive proof systems for $\mathcal{NP}$ so that it is infeasible to find instances on which the prover leaks knowledge.

Zero-knowledge proof systems for any language in $\mathcal{IP}$, based on (nonuniformly) one-way functions, were constructed by Impagliazzo and Yung [IY87] (yet, their paper contains no details). An alternative construction is presented by Ben-Or et. al. [Betal88].

Further reading related to the advanced sections

Additional negative results concerning zero-knowledge proofs of restricted types appear in Goldreich and Oren [GO87]. The interested reader is also directed to Boppana, Hastad and Zachos [BHZ87] for a proof that if every language in co$\mathcal{NP}$ has a constant-round interactive proof system then the Polynomial-Time Hierarchy collapses to its second level.

Round-efficient *perfect* zero-knowledge arguments for $\mathcal{NP}$, based on the intractability of the Discrete Logarithm Problem, appears in a paper by Brassard, Crépeau and Yung [BCY]. A round-efficient *perfect* zero-knowledge proof system for Graph Isomorphism appears in a paper by Bellare, Micali and Ostrovsky [BMO89].

A much more efficient construction of non-interactive proof systems for $\mathcal{NP}$, based on the same assumptions as [FLSnizk], has appeared in a paper of Kilian and Petrnak [KP96].

Author's Note: *KP96 has appeared in ECCC and will appear in Jour of Cryptology.*

An extremely efficient perfect zero-knowledge two-prover proof system for $\mathcal{NP}$, appears in a paper by Dwork et. al. [DFKNS]. Specifically, only logarithmic randomness and communication complexities are required to get a constant error probability. This result uses the characterization of $\mathcal{NP}$ in terms of low complexity multi-prover interactive proof systems, which is further discussed in Section [missing(eff-pcp.sec)].

Parallel repetition is problematic both in the multi-prover setting (cf., [Feige,Raz]) and in the computational-sound setting (cf., [BIN]).

The paper by Goldwasser, Micali and Rackoff [GMR85] contains also a suggestion for a general measure of "knowledge" revealed by a prover, of which zero-knowledge is merely a special case. For further details see Goldreich and Petrank [GPkc].

Author's Note: *GPkc has appeared in FOCS91. See also a recent work by Petrank and Tardos (in FOCS96).*

Author's Note: *The discussion of knowledge complexity may be better fit into the missing chapter on complexity.*

4.12.3 Open Problems

Our formulations of zero-knowledge (e.g., perfect zero-knowledge as defined in Definition 4.3.1) is different from the standard definition used in the literature (e.g., Definition 4.3.5). The standard definition refers to *expected* polynomial-time machines rather to strictly (probabilistic) polynomial-time machines. Clearly, Definition 4.3.1 implies Definition 4.3.5 (see Exercise 8), but it is open whether the converse hold.

Author's Note: *Base nizk and arguments on (more) general assumptions.*

4.12.4 Exercises

Exercise 1: *Decreasing the error probability in interactive proofs*:
Prove Proposition 4.2.7.

Guideline: Execute the weaker interactive proof sufficiently many times, using independently chosen coin tosses for each execution, and rule by an appropriate threshold. Observe that the bounds on completeness and soundness need to be efficiently computable. Be careful when demonstrating the soundness of the resulting verifier. The statement remains valid regardless of whether these repetitions are executed sequentially or "in parallel", yet demonstrating that the soundness condition is satisfied is much easier in the first case.

Exercise 2: *the role of randomization in interactive proofs – part 1*: Prove that if L has an interactive proof system in which the verifier is deterministic then $L \in \mathcal{NP}$. (Hint: Note that if the verifier is deterministic then the entire interaction between the prover and the verifier is determined by the prover.)

Exercise 3: *the role of randomization in interactive proofs – part 2*: Prove that if L has an interactive proof system then it has one in which the prover is deterministic. Furthermore, prove that for every (probabilistic) interactive machine V there exists a deterministic interactive machine P so that for every x the probability $\mathsf{Pr}(\langle P, V\rangle(x)=1)$ equals the supremum of $\mathsf{Pr}(\langle B, V\rangle(x)=1)$ taken over all interactive machines B.

Guideline: For each possible prefix of interaction, the prover can determine a message which maximizes the accepting probability of the verifier V.

Exercise 4: *the role of randomization in interactive proofs – part 3*: Consider a modification, to the definition of an interactive machine, in which the random-tapes of the prover and verifier coincide (i.e., intuitively, both use the same sequence of coin tosses which is known to both of them). Prove that every language having such a modified interactive proof system has also an interactive proof system (of the original kind) in which the prover sends a single message.

Exercise 5: *the role of error in interactive proofs*: Prove that if L has an interactive proof system in which the verifier never (not even with negligible probability) accepts a string not in the language L then $L \in \mathcal{NP}$.

Guideline: Define a relation R_L such that $(x, y) \in R_L$ if y is a full transcript of an interaction leading the verifier to accept the input x. We stress that y contains the verifier's coin tosses and all the messages received from the prover.

Exercise 6: *error in perfect zero-knowledge simulators - part 1*: Consider modifications of Definition 4.3.1 in which condition 1 is replaced by requiring, for some function $q(\cdot)$, that $\mathsf{Pr}(M^*(x)=\perp) < q(|x|)$. Assume that $q(\cdot)$ is polynomial-time computable. Show that if for some polynomials, $p_1(\cdot)$ and $p_2(\cdot)$, and all sufficiently large n's, $q(n) >$

$1/p_1(n)$ and $q(n) < 1-2^{-p_2(n)}$ then the modified definition is equivalent to the original one. Justify the bounds placed on the function $q(\cdot)$. (Hint: Invoke the simulator sufficiently many times.)

Exercise 7: *error in perfect zero-knowledge simulators - part 2*: Consider the following alternative to Definition 4.3.1, by which 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^* so that the following two ensembles are statistically close (i.e., their statistical difference is negligible as a function of $|x|$)

- $\{\langle P, V^* \rangle(x)\}_{x\in L}$
- $\{M^*(x)\}_{x\in L}$

Prove that Definition 4.3.1 implies the new definition.

Exercise 8: (E) *error in perfect zero-knowledge simulators - part 3*: Prove that Definition 4.3.1 implies Definition 4.3.5.

Exercise 9: *error in computational zero-knowledge simulators*: Consider an alternative to Definition 4.3.2, by which the simulator is allowed to output the symbol $\perp$ (with probability bounded above by, say, $\frac{1}{2}$) and its output distribution is considered conditioned on its not being $\perp$ (as done in Definition 4.3.1). Prove that this alternative definition is equivalent to the original one (i.e., to Definition 4.3.2).

Exercise 10: *alternative formulation of zero-knowledge - simulating the interaction*: Prove the equivalence of Definitions 4.3.2 and 4.3.3.

Exercise 11: Prove that Definition 4.3.8 is equivalent to a version where the auxiliary input to the verifier is rexplicitly bounded in length. That is, the alternative zero-knowledge claus reads

> for every polynomial ℓ, and for every probabilistic polynomial-time interactive machine V^*, there exists a probabilistic ploynomial-time algorithm M^*, so that the following two ensembles are computationally indistinguishable
>
> - $\{\langle P(y), V^*(z)\rangle(x)\}_{x\in L, y\in P_L(x), z\in\{0,1\}^{\ell(|x|)}}$
> - $\{M^*(x, z)\}_{x\in L, z\in\{0,1\}^{\ell(|x|)}}$

Note that it is immaterial here whether the running-time of M^* (as well as the distinguishing gap) is considered as a function of $|x|$ or as a function of $|(x, z)|$.

Exercise 12: Present a simple probabilistic polynomial-time algorithm which simulates the view of the interaction of the verifier described in Construction 4.3.6 with the prover defined there. The simulator, on input $x \in GI$, should have output which is distributed identically to $\text{view}_{V_{GI}}^{P_{GI}}(x)$.

Exercise 13: Prove that the existence of bit commitment schemes implies the existence of one-way functions.

> **Guideline:** Following the notations of Definition 4.4.1, consider the mapping of (v, s, r) to the receiver's view $(r, \overline{m})$. Observe that by the unambiguity requirement range elements are very unlikely to have inverses with both possible values of v. The mapping is polynomial-time computable and an algorithm that inverts it, even with success probability that is not negligible, can be used to contradict the secrecy requirement.

Exercise 14: Considering the commitment scheme of Construction 4.4.4, suggest a cheating sender that induces a receiver-view (of the commit phase) being both

- indistinguishable from the receiver-view in interactions with the prescribed sender;
- with very high probability, neither a possible 0-commitment nor a possible 1-commitment.

(Hint: the sender just replies with a uniformly chosen string.)

Exercise 15: *using Construction 4.4.4 as a commitment scheme in Construction 4.4.6*: Prove that when the commitment scheme of Construction 4.4.4 is used in the $G3C$ protocol then resulting scheme remains zero-knowledge. Consider the modifications required to prove Claim 4.4.7.2.

Exercise 16: *more efficient zero-knowledge proofs for $\mathcal{NP}$*: Following is an outline for a constant-round zero-knowledge proof for the Hamiltonian Circuit Problem (HCP), with acceptance gap $\frac{1}{2}$ (between inputs inside and outside of the language).

- *Common Input*: a graph $G=(V, E)$;
- *Auxiliary Input* (to the prover): a permutation ψ, over V, representing the order of vertices along a Hamiltonian Circuit;
- Prover's first step: Generates a random isomorphic copy of G, denoted $G' = (V, E')$. (Let π denote the permutation between G and G'). For each pair $(i, j) \in V^2$, the prover sets $e_{i,j} = 1$ if $(i, j) \in E'$ and $e_{i,j} = 0$ otherwise. The prover computes a random commitment to each $e_{i,j}$. Namely, it uniformly chooses $s_{i,j} \in \{0,1\}^n$ and computes $c_{i,j} = C_{s_{i,j}}(e_{i,j})$. The prover sends all the $c_{i,j}$'s to the verifier;
- Verifier's first step: Uniformly selects $\sigma \in \{0, 1\}$ and sends it to the prover;
- Prover's second step: Let σ be the message received from the verifier. If $\sigma = 1$ then the prover reveals all the $|V|^2$ commitments to the verifier (by revealing all $s_{i,j}$'s), and sends along also the permutation π. If $\sigma = 0$ then the prover reveals only $|V|$ commitments to the verifier, specifically those corresponding to the Hamiltonian circuit in G' (i.e., the prover sends $s_{\pi(1),\pi(\psi(2))}$, $s_{\pi(2),\pi(\psi(3))}$,...,$s_{\pi(n-1),\pi(\psi(n))}$, $s_{\pi(n),\pi(\psi(1))}$).

Complete the description of the above interactive proof, evaluate its acceptance probabilities, and provide a sketch of the proof of the zero-knowledge property (i.e., describe the simulator). If you are really serious provide a full proof of the zero-knowledge property.

Exercise 17: *strong reductions*: Let L_1 and L_2 be two languages in $\mathcal{NP}$, and let R_1 and R_2 be binary relations characterizing L_1 and L_2, respectively. We say that the relation R_1 is *Levin-reducible* to the relation R_2 if there exist two polynomial-time computable functions f and g such that the following two conditions hold.

1. *standard requirement*: $x \in L_1$ if and only if $f(x) \in L_2$.
2. *additional requirement*: For every $(x, w) \in R_1$, it holds that $(f(x), g(w)) \in R_2$.

We call the above reduction after Levin, who upon discovering, independently of Cook and Karp, the existence of $\mathcal{NP}$-complete problem, made a stronger definition of a reduction which implies the above. Prove the following statements

1. Let $L \in \mathcal{NP}$ and let L_R be the generic relation characterizing L (i.e., fix a non-deterministic machine M_L and let $(x, w) \in R_L$ if w is an accepting computation of M_L on input x). Let R_{SAT} be the standard relation characterizing SAT (i.e., $(x, w) \in R_{SAT}$ if w is a truth assignment satisfying the CNF formula x). Prove that R_L is Levin-reducible to R_{SAT}.
2. Let R_{SAT} be as above, and let R_{3SAT} be defined analogously for $3SAT$. Prove that R_{SAT} is Levin-reducible to R_{3SAT}.
3. Let R_{3SAT} be as above, and let R_{G3C} be the standard relation characterizing $G3C$ (i.e., $(x, w) \in R_{G3C}$ if w is a 3-coloring of the graph x). Prove that R_{3SAT} is Levin-reducible to R_{G3C}.
4. Levin-reductions are transitive.

Exercise 18: Prove the existence of a Karp-reduction of L to SAT that, when considered as a function, can be inverted in polynomial-time. Same for the reduction of SAT to $3SAT$ and the reduction of $3SAT$ to $G3C$. (In fact, the standard Karp-reductions have this property.)

Exercise 19: *applications of Theorem 4.4.10*: Assuming the existence of non-uniformly one-way functions, present solutions to the following cryptographic problems:

1. Suppose that party R received over a public channel a message encrypted using its own public-key encryption. Suppose that the message consists of two parts and party R wishes to reveal to everybody the first part of the message but not the second. Further suppose that the other parties want a proof that R indeed revealed the correct contents of the first part of its message.

2. Suppose that party S wishes to send party R a signature to a publicly known document so that only R gets the signature but everybody else can verify that such a signature was indeed sent by S. (We assume that all parties share a public channel.)
3. Suppose that party S wishes to send party R a commitment to a partially specified statement so that R remains oblivious of the unspecified part. For example, S may wish to commit itself to some standard offer while keeping the amount offered secret.

Exercise 20: *on knowledge tightness*: Evaluate the knowledge tightness of Construction 4.4.6, when applied logarithmically many times *in parallel*.

Exercise 21: *error reduction in computationally sound proofs – part 1*: Given a computationally sound proof (with error probability $\frac{1}{3}$) for a language L construct a computationally sound proof with negligible error probability (for L).

> **Guideline:** Use sequentil repetitions. Parallel repetitions may fail to reduce computational-soundness in some cases (see [BIN]).

Exercise 22: *commitment schemes – an impossibility result*: Prove that there exists no two-party protocol which simultaneously satisfies the perfect secrecy requirement of Definition 4.8.2 and the (information theoretic) unambiguity requirement of Definition 4.4.1.

Exercise 23: *alternative formulation of black-box zero-knowledge*: 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 (not necessarily uniform) polynomial-size circuit family $\{B_n\}_{n\in\mathbb{N}}$, the ensembles $\{\langle P, B_{|x|}\rangle(x)\}_{x\in L}$ and $\{M^{B_{|x|}}(x)\}_{x\in L}$ are indistinguishable by (non-uniform) polynomial-size circuits. Namely, for every polynomial-size circuit family $\{D_n\}_{n\in\mathbb{N}}$, every polynomial $p(\cdot)$, all sufficiently large n and $x \in \{0,1\}^n \cap L$,

$$|\mathsf{Pr}\left(D_n(\langle P, B_n\rangle(x))=1\right) - \mathsf{Pr}\left(D_n(M^{B_n}(x))=1\right)| < \frac{1}{p(n)}$$

Prove that the current formulation is equivalent to the one presented in Definition 4.5.7.

Exercise 24: Prove that the protocol presented in Construction 4.4.6 is indeed a black-box zero-knowledge proof system for $G3C$.

> **Guideline:** Use the formulation presented above.

Author's Note: *First draft (execpt Section 4.10) written mainly in 1992. Section 4.10 written in February 1996.*