Chapter 6

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.

6.1 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".

6.1.1 The Notion of a Proof

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

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.

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.

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 6.2 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.

6.1.2 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.

6.2 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 8.

6.2.1 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.

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 6.1 (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 6.2 (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 6.2.3, 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).

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 6.3 (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.

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 6.4 (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 $x \in L$*

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

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

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

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 8).

Definition 6.5 (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 6.6 (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 $x \in L$*

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

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

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

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 6.7 *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).

6.2.2 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 6.8 (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 6.9 *The language GNI is in the class $\mathcal{IP}$. Furthermore, the programs specified in Construction 6.8 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 6.9.1: If the graphs G_1 and G_2 are isomorphic then for every graph G' it holds that

$$\text{Prob}\,(\xi=1|\Pi(G_\xi)=G') = \text{Prob}\,(\xi=2|\Pi(G_\xi)=G') = \frac{1}{2}$$

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

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

Using Bayes Rule, the claim follows.□

Using Claim 6.9.1, 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

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

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

Remarks concerning Construction 6.8

In the proof system of Construction 6.8, 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 8.

6.2.3 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 6.10 (interactive proof systems - revisited):

- *An* interactive machine *is defined as in Definition 6.1, 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* $z \in \{0,1\}^*$
    $$\text{Prob}\left(\langle P(y), V(z)\rangle(x) = 1\right) \geq \frac{2}{3}$$
  - Soundness: *For every* $x \notin L$*, every interactive machine* B*, and every* $y, z \in \{0,1\}^*$
    $$\text{Prob}\left(\langle B(y), V(z)\rangle(x) = 1\right) \leq \frac{1}{3}$$

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.

6.3 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").

6.3.1 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 6.11 (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., $\mathrm{Prob}(M^*(x)=\perp) \leq \frac{1}{2}$).*
2. *Let $m^*(x)$ be a random variable describing the distribution of $M^*(x)$ conditioned on $M^*(x) \neq \perp$ (i.e., $\mathrm{Prob}(m^*(x)=\alpha) = \mathrm{Prob}(M^*(x)=\alpha | M^*(x) \neq \perp)$, 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 4, 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

$$|\text{Prob}(D(x, R_x) = 1) - \text{Prob}(D(x, S_x) = 1)| < \frac{1}{p(|x|)}$$

Definition 6.12 (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 6.11), does not add to the power of Definition 6.12.

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 6.13 (zero-knowledge – alternative formulation): *Let (P, V), L and V^* be as in Definition 6.12. 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 6.13 is obtained from Definition 6.12 by replacing $\langle P, V^*\rangle(x)$ for $\text{view}_{V^*}^{P}(x)$. The simulator M^* used in Definition 6.13 is related, but not equal, to the simulator used in Definition 6.12 (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 6.13 is equivalent to Definition 6.12 (see Exercise 10). The latter fact justifies the use of Definition 6.13, which is more convenient to work with, although it seems less natural than Definition 6.12. An alternative formulation of perfect zero-knowledge is straightforward, and clearly it is equivalent to Definition 6.11.

* Complexity classes based on Zero-Knowledge

Definition 6.14 (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 6.24 and Theorems 3.29 and 6.30.

* Expected polynomial-time simulators

The formulation of perfect zero-knowledge presented in Definition 6.11 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 6.15 (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 6.11 implies Definition 6.15 – 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 6.11. We prefer to adopt Definition 6.11, rather than Definition 6.15, 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 6.12 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.

6.3.2 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 6.2).

Construction 6.16 (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 6.17 *The language GI has a perfect zero-knowledge interactive proof system. Furthermore, the programs specified in Construction 6.16 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 6.3.4. 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 6.5.

We stress that it is not known whether $GI \in \mathcal{BPP}$. Hence, Proposition 6.17 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 6.16 (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 6.16, except that at termination it output the entire transcript of its interaction with P_{GI} – see Exercise 11. 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
$$F \stackrel{\mathrm{def}}{=} \{(\psi(u),\psi(v)) : (u,v) \in E_\tau\}$$
Set $G'' \stackrel{\mathrm{def}}{=} (V_\tau, F)$.

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 6.17.1: 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 6.9.1). Then, for every graph G'', it holds that

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

Claim 6.17.1 is identical to Claim 6.9.1 (used to demonstrate that Construction 6.8 constitutes an interactive proof for GNI). As in the rest of the proof of Proposition 6.9, 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 6.17.2: Let $x = (G_1, G_2) \in GI$. Then, for every string r, graph H, and permutation ψ, it holds that

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

proof: Let $m^*(x)$ describe $M^*(x)$ conditioned on its not being $\perp$. We first observe that both $m^*(x)$ and $\text{view}_{V^*}^{P_{GI}}(x)$ 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 $\text{view}_{V^*}^{P_{GI}}(x)$ 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

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

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 $\mathrm{aut}(G_2)/(|V_2|!)$. Furthermore, for each H (which is isomorphic to G_2),

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

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 $\mathrm{aut}(G_2)/(|V_2|!)$. Furthermore, for each H (which is isomorphic to G_2),

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

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

The claim follows. □

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

6.3.3 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 6.16 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 6.10.

Definition 6.18 (zero-knowledge – revisited): *Let (P,V) be an interactive proof for a language L (as in Definition 6.10). Denote by $P_L(x)$ the set of strings y satisfying the completeness condition with respect to $x \in L$ (i.e., for every $z \in \{0,1\}^*$* Prob $(\langle P(y), V(z)\rangle(x)=1) \geq \frac{2}{3}$*). 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

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

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 6.10, 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. 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 6.18 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 6.16, 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 6.17 (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.

* Implicit non-uniformity in Definition 6.18

The non-uniform nature of Definition 6.18 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 6.18 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 $z \in \{0,1\}^*$,

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

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.

6.3.4 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 6.18) satisfies this requirement. Interestingly, zero-knowledge proofs as defined in Definition 6.12 are not closed under sequential composition, and this fact is indeed another indication to the necessity of augmenting this definition (as done in Definition 6.18).

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 6.16 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 6.16 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 6.19 (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 6.19.1: 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. $\Box$

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

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

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 6.19.2: 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

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

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

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

Using Claim 6.19.1, the $Q(|x|)^{\rm th}$ 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 6.19.2: The proof of Claim 6.19.2 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 6.19.2. 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

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

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

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

Denote $\epsilon(n) \stackrel{\mathrm{def}}{=} 1/(Q(n) \cdot p(n))$. Combining the definition of the i^{th} and $i-1^{\mathrm{st}}$ 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

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

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 6.3.3, 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 6.5.

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 6.9 and 6.6.

6.4 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.

6.4.1 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.

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 6.20 (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 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 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 6.13, 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 6.8.2.

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.

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 6.21 (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 6.22 *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 6.21 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 $b(s) \oplus v = \sigma$. ■

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 12). On the other hand, by the results in Chapter 3 (specifically, Theorems 3.11 and 3.29), 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 6.23 (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 6.24 *If G is a pseudorandom generator, then the protocol presented in Construction 6.23 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. ■

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 2.6 in Section 2.2) commitment schemes with nonuniform secrecy can be constructed, following the same constructions used in the uniform case.

6.4.2 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 $(u,v) \in E$.

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 $\phi(v) \stackrel{\text{def}}{=} \pi(\psi(v))$, 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.)

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 6.21 (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 6.25 (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 6.19), it follows that also polynomially many sequential applications of this protocol preserve the zero-knowledge property.

Proposition 6.26 *Suppose that the commitment scheme used in Construction 6.25 satisfies the (nonuniform) secrecy and the unambiguity requirements. Then Construction 6.25 constitutes an auxiliary input zero-knowledge (generalized) interactive proof for* $G3C$.

For further discussion of Construction 6.25 see remarks at the end of the current subsection.

Proof of Proposition 6.26

We first prove that Construction 6.25 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 6.25, 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 6.13), 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 6.16) 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 6.26.1: 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

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

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

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

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
>
> $$\text{Prob}(A_n(C_{U_{n^2}}(\gamma)) = 1) = p_{u_n,v_n}(G_n, r_n, \gamma)$$
>
> 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 6.26.1, 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 $(e_1, ..., e_n) \in \{1, 2, 3\}^n$,

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

(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

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

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 6.26.2: 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 $G=(V,E)$,

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

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 6.26.2 follows easily from the secrecy property of the commitment scheme. Indeed, Claim 6.26.2 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

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

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 6.26.1. 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 $1^n 2^n 3^n$. 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

$$\begin{aligned}\text{Prob}(A_n(C(1^n 2^n 3^n)) = 1) &= (1 - q_{u_n,v_n}(G_n)) \\ &\quad + q_{u_n,v_n}(G_n) \cdot \text{Prob}(A(\nu_{u_n,v_n}(G_n)) = 1)\end{aligned}$$

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

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

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

$$|\text{Prob}(A_n(C(1^n 2^n 3^n)) = 1) - \text{Prob}(A_n(C(T_{3n}) = 1)|$$

$$\begin{aligned}
&\geq q_{u_n,v_n} \cdot |\mathrm{Prob}(A(\nu_{u_n,v_n}) = 1) - \mathrm{Prob}(A(\mu_{u_n,v_n}) = 1)| - 2 \cdot |p_{u_n,v_n} - q_{u_n,v_n}| \\
&> \frac{1}{p(n)} \cdot \frac{1}{p(n)} - 2 \cdot \frac{1}{3 \cdot p(n)^2} \\
&= \frac{1}{3 \cdot p(n)^2}
\end{aligned}$$

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 6.26.2. □

Combining Claims 6.26.1 and 6.26.2, the zero-knowledge property of P_{G3C} follows. This completes the proof of the proposition. ■

Concluding remarks

Construction 6.25 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 6.26 indeed took advantage on this property. We remark that Construction 6.23 also possesses this simultaneous secrecy property, and hence the proof of Proposition 6.26 can be carried out also if the commitment scheme in used is the one of Construction 6.23 (see Exercise 14). We recall that this latter construction constitutes a commitment scheme if and only if such schemes exist at all (since Construction 6.23 is based on any one-way function and the existence of one-way functions is implied by the existence of commitment schemes).

Proposition 6.26 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 6.25 can be shown to be *uniformly* zero-knowledge.

By itself, Construction 6.25 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 6.19), 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 6.5.

An important property of Construction 6.25 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 6.25 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*.

6.4.3 The General Result and Some Applications

The theoretical and practical importance of a zero-knowledge proof for Graph 3-Coloring (e.g., Construction 6.25) 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 6.25) 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 16.) 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 6.27 (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 6.28 *Suppose that the subprotocol used in the last step of Construction 6.27 is indeed an auxiliary input zero-knowledge proof for $G3C$. Then Construction 6.27 constitutes an auxiliary input zero-knowledge proof for L.*

Proof: The fact that Construction 6.27 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 6.27 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 6.27, 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 6.27, 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 17). In any case, we immediately get

Theorem 6.29 *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 3.29 (asserting that one-way functions imply pseudorandom generators), Proposition 6.24 (asserting that pseudorandom generators imply commitment schemes), and Exercise 12 (asserting that commitment schemes imply one-way functions).

An Example: Proving properties of secrets

A typical application of Theorem 6.29 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 6.29 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 6.29 can be applied. Additional examples are presented in Exercise 18.

Zero-Knowledge for any language in IP

Interestingly, the result of Theorem 6.29 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 6.30 *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 6.30 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).

6.4.4 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).

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 6.25. Recall that this proof system has very moderate acceptance gap, specifically $1/|E|$, on common input graph $G = (V, E)$. So Construction 6.25 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 15, 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 6.25 and in Exercise 15 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 6.9.

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 6.31 (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 6.12) 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 6.16 has knowledge tightness 2, whereas Construction 6.25 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.

6.5 * 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 6.29 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?