2. *Let* $\sigma \in \{0, 1\}$. *We say that a joint view (of an interaction),* $((r, \overline{m}), (s, \tilde{m}))$*, has* **a feasible σ-opening** (with respect to F^* and $p(\cdot)$) *if on input* $(\overline{m}, (s, \tilde{m}), \sigma)$*, algorithm* F^* *outputs, with probability at least* $1/p(n)$*, a string* s' *such that* $\overline{m}$ *describes the messages received by* R *when* R *uses local coins* r *and interacts with machine* S *that uses local coins* s' *and input* $(\sigma, 1^n)$.

   (Remark: We stress that s' *may, but need not, equal* s*. The output of algorithm* F^* *has to satisfy a relation that depends on only part of the input (i.e., the receiver's view* $(r, \overline{m})$*); the sender's view (i.e.,* $(s, \tilde{m})$*) is supplied to algorithm* F^* *as additional help.)*

3. *We say that a joint view is* **ambiguous with respect to** F^* **and** $p(\cdot)$ *if it has both a feasible* 0*-opening and a feasible* 1*-opening* (with respect to F^* and $p(\cdot)$).

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

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

One can consider a relaxation of the secrecy condition in which the probability ensembles $\{\langle S(0), R^*\rangle(1^n)\}_{n\in\mathbb{N}}$ and $\{\langle S(1), R^*\rangle(1^n)\}_{n\in\mathbb{N}}$ are required to be statistically close, rather than identically distributed. We choose not to do so because the currently known constructions achieve the more stringent condition. Furthermore, use of the weaker notion of a perfectly hiding commitment scheme (in Section 4.8.3) yields almost-perfect zero-knowledge arguments rather than perfect zero-knowledge ones.

As in Definition 4.4.1, the secrecy requirement refers explicitly to the situation at the end of the commit phase, whereas the unambiguity requirement implicitly assumes that the reveal phase takes the following canonical 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 (i.e., r) used by R in the commit phase) indeed yield the messages that R has received in the commit phase. Verification is done in polynomial time (by running the programs S and R).

4.8.2.2. Construction Based on One-Way Permutations

Perfectly hiding commitment schemes can be constructed using any one-way permutation. The known scheme, however, involves a linear (in the security parameter) number of rounds. Hence, it can be used for the purposes of the current section, but not for the construction in Section 4.9.1.

Construction 4.8.3 (A Perfectly Hiding Bit Commitment): *Let f be a permutation, and let $b(x, y)$ denote the inner product* mod 2 *of x and y (i.e., $b(x, y) = \sum_{i=1}^{n} x_i y_i \bmod 2$, where $x = x_1 \cdots x_n \in \{0, 1\}^n$ and $y = y_1 \cdots y_n \in \{0, 1\}^n$).*

1. Commit phase *(using security parameter n):*

(a) (Local computations): *The receiver randomly selects $n - 1$ linearly independent vectors $r^1, \ldots, r^{n-1} \in \{0, 1\}^n$. The sender uniformly selects $s \in \{0, 1\}^n$ and computes $y = f(s)$.*
(Thus far, no message has been exchanged between the parties.)

(b) (Iterative hashing): *The parties proceed in $n - 1$ rounds. In the i round ($i = 1, \ldots, n - 1$), the receiver sends r^i to the sender, which replies by computing and sending $c^i \stackrel{\text{def}}{=} b(y, r^i)$.*

(c) (The "actual" commitment): *At this point there are exactly two solutions to the system of equations $\{b(y, r^i) = c^i : 1 \le i \le n - 1\}$. (Both parties can easily determine both solutions.)*

- *The sender sets $\pi = 1$ if y is the lexicographically first solution (of the two), and $\pi = 0$ otherwise.*
- *To commit to a value $v \in \{0, 1\}$, the sender sends $c^n \stackrel{\text{def}}{=} \pi \oplus v$ to the receiver.*

2. Canonical reveal phase: *In the reveal phase, the sender reveals v along with the string s randomly selected by it in the commit phase. The receiver accepts the value v if the following two conditions hold, where $((r^1, \ldots, r^{n-1}), (c^1, \ldots, c^n))$ denote the receiver's view of the commit phase:*

- *$b(f(s), r^i) = c^i$, for all $1 \le i \le n - 1$.*
- *If there exists $y' < f(s)$ (resp., $y' > f(s)$) such that $b(y', r^i) = c^i$ for all $1 \le i \le n - 1$, then $v = c^n$ (resp., $v = c^n \oplus 1$) must hold.*

That is, the receiver solves the linear system $\{b(y^j, r^i) = c^i\}_{i=1}^{n-1}$, obtaining solutions $y^1 < y^2$, so that $b(y^j, r^i) = c^i$ for $j = 1, 2$ and $i = 1, \ldots, n - 1$. Next, it checks whether or not $f(s) \in \{y^1, y^2\}$ (if the answer is negative, it rejects immediately) and sets π accordingly (i.e., so that $f(s) = y^\pi$). It accepts the value v if and only if $v \equiv c^n + \pi \pmod 2$.

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

It is quite easy to see that Construction 4.8.3 satisfies the secrecy condition. The proof that the unambiguity requirement is satisfied is quite complex andis omitted. The intuition underlying the proof is that it is infeasible to play the iterative hashing so as to reach a situation in which one can invert f on *both* the resulting solutions y^1 and y^2. (We mention that this reasoning fails if one replaces the iterative hashing by an ordinary one; see Exercise 33.)

4.8.2.3. Construction Based on Claw-Free Collections

A perfectly hiding commitment scheme of *constant number of rounds* can be constructed using a seemingly stronger intractability assumption, specifically, the existence of claw-free collections (see Section 2.4.5). This assumption implies the existence of one-way functions, but it is not known if the converse is true. Nevertheless, claw-free collections can be constructed under widely believed assumptions such as the intractability of factoring and DLP. Actually, the construction of perfectly hiding commitment schemes, presented next, uses a claw-free collection with an additional property; specifically, it is assumed that the set of indices of the collection (i.e., the range of algorithm I) can be efficiently recognized (i.e., is in $\mathcal{BPP}$). Such a collection exists under the assumption that DLP is intractable (see Section 2.4.5).

Construction 4.8.5 (A Constant-Round Perfectly Hiding Bit Commitment): *Let (I, D, F) be a triplet of probabilistic polynomial-time algorithms. (Think of I as the index generating algorithm of a claw-free collection $\{(f_i^0, f_i^1) : i \in \overline{I}\}$ and S and F as the corresponding sampling and evaluating algorithms.)*

1. Commit phase: *To receive a commitment to a bit (using security parameter n), the receiver randomly generates $i = I(1^n)$ and sends it to the sender. To commit to value $v \in \{0, 1\}$ (upon receiving the message i from the receiver), the sender checks to see if indeed i is in the range of $I(1^n)$, and if so the sender randomly generates $s = D(v, i)$, computes $c = F(v, i, s)$, and sends c to the receiver. (In case i is not in the range of $I(1^n)$, the sender aborts the protocol, announcing that the receiver is cheating.)*
2. (Almost-canonical) reveal phase: *In the reveal phase, it suffices for the sender to reveal the string s generated by it in the commit phase. The receiver accepts the value v if $F(v, i, s) = c$, where (i, c) is the receiver's (partial) view of the commit phase.*

Proposition 4.8.6: *Let (I, D, F) be a claw-free collection with a probabilistic polynomial-time-recognizable set of indices. Then the protocol presented in Construction 4.8.5 constitutes a perfectly hiding bit-commitment scheme.*

Proof Sketch: The secrecy requirement follows directly from Property 2 of a claw-free collection (as in Definition 2.4.6) combined with the test $i \in I(1^n)$ conducted by the sender. The unambiguity requirement follows from Property 3 of a claw-free collection (Definition 2.4.6), using a standard reducibility argument. (Note that $F(0, i, s_0) = F(1, i, s_1)$ means that (s_0, s_1) constitute a claw for the permutation pair (f_i^0, f_i^1).) ■

The rationale for having the sender check to see if the index i indeed belongs to the legitimate index set $\overline{I}$ is that only permutation pairs (f_i^0, f_i^1) with $i \in \overline{I}$ are guaranteed to have identical range distributions. Thus, it actually is not necessary for the sender to check whether or not $i \in \overline{I}$; it suffices for it to check (or be otherwise convinced) that the permutation pair (f_i^0, f_i^1) satisfies the requirement of identical range distributions. Consider, for example, the factoring claw-free collection (presented in Section 2.4.5). This collection *is not known* to have an efficiently recognizable index set. Still, having sent an index N, the receiver can prove in zero-knowledge to the sender that the permutation pair (f_N^0, f_N^1) satisfies the requirement of identical range distributions. What is actually being proved is that half of the square roots of each quadratic residue mod N have Jacobi symbol 1 (relative to N). A (perfect) zero-knowledge proof system for this claim does exist (without assuming anything). In fact, it suffices to use a witness-independent proof system, and such a system having a constant number of rounds does exist (again, without assuming anything). Hence, the factoring claw-free collection can be used to construct a constant-round perfectly hiding commitment scheme, and thus such commitment schemes also exist under the assumption that the factoring of Blum integers is intractable.

4.8.2.4. Non-Uniform Computational Unambiguity

Actually, for the applications to proof/argument systems, both the one following and the one in Section 4.9.1, we need commitment schemes with perfect secrecy and *non-uniform* computational unambiguity. (The reason for this need is analogous to one discussed in the case of the zero-knowledge proof for $\mathcal{NP}$ presented in Section 4.4.) By non-uniform computational unambiguity we mean that the unambiguity condition should also hold for (non-uniform) families of polynomial-size circuits. We stress that the foregoing constructions of perfect commitment schemes possess the non-uniform computational unambiguity, provided that the underlying intractability assumption also holds with respect to non-uniform polynomial-size circuits (e.g., the one-way permutation is hard to invert even by such circuits, and the claw-free collections also foil non-uniform polynomial-size claw-forming circuits).

In order to prevent the terminology from becoming too cumbersome, we omit the attribute "non-uniform" when referring to the perfectly hiding commitment schemes in the description of the two applications mentioned earlier.

4.8.2.5. Commitment Schemes with A Posteriori Secrecy

We conclude the discussion of perfectly hiding commitment schemes by introducing a relaxation of the secrecy requirement. The resulting scheme cannot be used for the purposes of the current section, yet it is useful in different settings discussed later. The advantage of the relaxation is that it allows us to construct such (constant-round perfectly hiding) commitment schemes using any claw-free collection, thus waiving the additional requirement that the index set be efficiently recognizable.

Loosely speaking, we relax the secrecy requirement of perfectly hiding commitment schemes by requiring that it hold only when the receiver follows its prescribed program (denoted R). This seems strange, because we do not really want to assume that the real receiver follows the prescribed program (but rather protect against arbitrary behavior). The point is that a real receiver may disclose its commit-phase coin tosses at a later stage, say even after the reveal phase, and by doing so prove *a posteriori* that (at least in some weak sense) it was following the prescribed program. Actually, the receiver proves only that it has behaved in a manner that is consistent with its program.

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

Proposition 4.8.8: *Let (I, D, F) be a claw-free collection. Consider a modification of Construction 4.8.5 in which the sender's check of whether or not i is in the range of $I(1^n)$ is omitted (from the commit phase). Then the resulting protocol constitutes a bit-commitment scheme with perfect a posteriori secrecy.*

We stress that in contrast to Proposition 4.8.6, here the claw-free collection need not have an efficiently recognizable index set. Hence, we had to omit the sender's check. Yet the receiver can later prove that the message it sent during the commit phase (i.e., i) is indeed a valid index simply by disclosing the random coins it used in order to generate i (using algorithm I).

Proof Sketch: The a posteriori secrecy requirement follows directly from Property 2 of a claw-free collection (combined with the fact that i is indeed a valid index, since it is generated by invoking I). The unambiguity requirement follows as in Proposition 4.8.6. ■

A typical application of a commitment scheme with perfect a posteriori secrecy is presented in Section 4.9.1. In that setting the commitment scheme is used inside an interactive proof, with the verifier playing the role of the sender (and the prover playing the role of the receiver). If the verifier a posteriori learns that the prover has been cheating, then the verifier rejects the input. Hence, no damage is caused, in this case, by the fact that the secrecy of the verifier's commitments may have been breached.

4.8.3. Perfect Zero-Knowledge Arguments for $\mathcal{NP}$

Having a perfectly hiding commitment scheme at our disposal, we can construct perfect zero-knowledge arguments for $\mathcal{NP}$ by modifying the construction of (computational) zero-knowledge proofs (for $\mathcal{NP}$) in a totally syntactic manner. We recall that in these proof systems (e.g., Construction 4.4.7 for Graph 3-Colorability) the prover uses a perfectly binding commitment scheme in order to commit itself to many values, some of which it later reveals upon the verifier's request. All that is needed is to replace the perfectly binding commitment scheme used by the prover with a perfectly hiding commitment scheme. We claim that the resulting protocol is a perfect zero-knowledge argument (i.e., computationally sound proof) for the original language.

Proposition 4.8.9: *Consider a modification of Construction 4.4.7 such that the commitment scheme used by the prover is replaced by a perfectly hiding commitment scheme. Then the resulting protocol is a perfect zero-knowledge weak argument for Graph 3-Colorability.*

By a *weak argument* we mean a protocol in which the gap between the completeness and the computational-soundness conditions is noticeable. In our case, the verifier always accepts inputs in $G3C$, whereas no efficient prover can fool him into accepting graphs $G = (V, E)$ not in $G3C$ with probability that is non-negligibly greater than $1 - \frac{1}{|E|}$. Specifically, we shall show that no efficient prover can fool him into accepting graphs $G = (V, E)$ not in $G3C$ with probability greater than $1 - \frac{1}{2|E|}$. Recall that by (sequentially) repeating this protocol polynomially many times the (computational-soundness) error probability can be made negligible.

Proof Sketch: We start by proving that the resulting protocol is perfect zero-knowledge for $G3C$. We use the same simulator as in the proof of Proposition 4.4.8. However, this time analyzing the properties of the simulator is much easier and yields stronger results, the reason being that here the prover's commitment is perfectly hiding, whereas there it is only computationally hiding. Thus, here the prover's commitments are distributed independently of the committed values, and consequently the verifier acts in total oblivion of the values. It follows that the simulator outputs a transcript with probability exactly $\frac{2}{3}$, and for similar reasons this transcript is distributed identically to the real interaction. The perfect zero-knowledge property follows.

The completeness condition is obvious, as in the proof of Proposition 4.4.8. It is left to prove that the protocol satisfies the (weak) computational-soundness requirement. This is indeed the more subtle part of the current proof (in contrast to the proof of Proposition 4.4.8, in which proving soundness is quite easy). The reason is that here the prover's commitment is only computationally binding, whereas there it is perfectly binding. Thus, here we use a reducibility argument to show that a prover's ability to cheat, with too high a probability, on inputs not in $G3C$ translates to an algorithm contradicting the unambiguity of the commitment scheme. Details follow.

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

Combining Propositions 4.8.4 and 4.8.9, we get the following:

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

ZK Proofs versus Perfect ZK Arguments: Which to Prefer?

Propositions 4.4.8 and 4.8.9 exhibit a kind of trade-off between the strength of the soundness and zero-knowledge properties. The protocol of Proposition 4.4.8 offers computational zero-knowledge and "perfect" soundness, whereas the protocol of Proposition 4.8.9 offers perfect zero-knowledge and only computational soundness. We remark that the two results are *not* obtained under the same assumptions: The conclusion of Proposition 4.4.8 is valid as long as one-way functions exist, whereas the conclusion of Proposition 4.8.9 seems to require a (probably) stronger assumption. Yet one may ask which of the two protocols we should prefer, *assuming that they are both valid* (i.e., assuming that the underlying complexity assumptions hold). The answer depends on the setting (i.e., application) in which the protocol is to be used. In particular, one should consider the following issues:

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

4.8.4. Arguments of Poly-Logarithmic Efficiency

A dramatic improvement in the efficiency of zero-knowledge arguments for $\mathcal{NP}$ can be obtained by combining the idea of an authentication tree with results regarding probabilistically checkable proofs (PCPs). In particular, assuming the existence of very strong collision-free hashing functions, one can construct a computationally sound (zero-knowledge) proof for any language in $\mathcal{NP}$, using only poly-logarithmic amounts of communication and randomness. The interesting point in that statement is the mere existence of such extremely efficient arguments, let alone their zero-knowledge property. Hence, we confine ourselves to describing the ideas involved in constructing such arguments and do not address the issue of making them zero-knowledge. (We stress that the argument system presented next is *not* zero-knowledge, unless $\mathcal{NP} \subseteq \mathcal{BPP}$.)

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

The preceding paragraph shows that the randomness complexity can be made poly-logarithmic and that the verifier need only inspect a poly-logarithmic number of randomly selected values. Specifically, the prover commits to each of the values of the variables in the formula $f(x)$ but is asked to reveal only a few of them. To obtain (total) poly-logarithmic communication complexity, we use a special commitment scheme that allows us to commit to a string of length n such that the commitment phase takes poly-logarithmic communication and individual bits of this string can be revealed (and verified as correct) at poly-logarithmic communication cost. For constructing such a commitment scheme, we use a *collision-free* hashing function. The function maps strings of some length to strings of half that length, so that it is "hard" to find two strings that are mapped by the function to the same image. (The following description is slightly inaccurate. What we need is a family of hashing functions such that no small non-uniform circuit, given the description of a function in the family, can form collisions with respect to it.)

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

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

4.9.* Constant-Round Zero-Knowledge Proofs

In this section we consider the problem of constructing *constant-round* zero-knowledge proof systems *with negligible error probability* for all languages in $\mathcal{NP}$. To make the rest of the discussion less cumbersome, we define a proof system to be **round-efficient** if it is *both constant-round and has negligible error probability*. We stress that none of the zero-knowledge proof systems for $\mathcal{NP}$ presented and discussed thus far have been round-efficient (i.e., they either had non-constant numbers of rounds or had non-negligible error probability).

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

1. basing the construction of round-efficient zero-knowledge proof systems on constant-round perfectly hiding commitment schemes (as defined in Section 4.8.2)
2. constructing (round-efficient zero-knowledge) *computationally sound* proof systems (as defined in Section 4.8) instead of (round-efficient zero-knowledge) proof systems

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

The two approaches have a fundamental idea in common. We start with an abstract exposition of this common idea. Recall that the *basic* zero-knowledge proof for Graph 3-Colorability, presented in Construction 4.4.7, consists of a constant number of rounds. However, this proof system has a non-negligible error probability (in fact, the error probability is very close to 1). In Section 4.4 it was suggested that the error probability be reduced to a negligible value by sequentially applying the proof system sufficiently many times. The problem is that this yields a proof system with a non-constant number of rounds. A natural suggestion is to perform the repetitions of the basic proof in parallel, instead of sequentially. The problem with this "solution" is that it is not known if the resulting proof system is zero-knowledge. Furthermore, it is known that it is not possible to present, as done in the proof of Proposition 4.4.8, a single simulator that uses any possible verifier as a black box (see Section 4.5.4). The source of trouble is that when playing many copies of Construction 4.4.7 in parallel, a cheating verifier can select the edge to be inspected (i.e., Step V1) in each copy, depending on the commitments sent in all copies (i.e., in Step P1). Such behavior of the verifier defeats a simulator analogous to the one presented in the proof of Proposition 4.4.8.

One way to overcome this difficulty is to "switch" the order of Steps P1 and V1. But switching the order of these steps enables the prover to cheat (by sending commitments in which only the "query edges" are colored correctly). Hence, a more refined approach is required. The verifier starts by committing itself to one edge query per each copy (of Construction 4.4.7), then the prover commits itself to the coloring in each copy, and only then does the verifier reveal its queries, after which the rest of the proof proceeds as before. The commitment scheme used by the verifier should prevent the prover from predicting the sequence of edges committed to by the verifier. This is the point where the two approaches differ.

1. The first approach uses a perfectly hiding commitment scheme. The problem with this approach is that such (constant-round) schemes are known to exist only under seemingly stronger assumptions than merely the existence of one-way functions. Yet such schemes do exist under assumptions such as the intractability of factoring integers or the intractability of the discrete-logarithm problem.
2. The second approach bounds the computational resources of prospective cheating provers. Consequently, it suffices to utilize, "against" these provers (as commitment receivers), commitment schemes with computational security. We remark that this approach uses (for the commitments by the prover) a commitment scheme with an extra property. Yet such schemes can be constructed using any one-way function.

Caveat. Both approaches lead to protocols that are zero-knowledge in a liberal sense (i.e., using *expected* polynomial-time simulators as defined in Section 4.3.1.6). It is not known if these protocols (or other round-efficient protocols for $\mathcal{NP}$) can be shown to be zero-knowledge in the strict sense (i.e., using strict probabilistic polynomial-time simulators).

4.9.1. Using Commitment Schemes with Perfect Secrecy

For the sake of clarity, let us start by presenting a detailed description of the constant-round interactive proof (for Graph 3-Colorability, $G3C$) sketched earlier. This interactive proof employs two different commitment schemes. The first scheme is the simple (perfectly binding) commitment scheme presented in Construction 4.4.2. We denote by $C_s(\sigma)$ the commitment of the sender, using coins s, to the (ternary) value

$\sigma \in \{1, 2, 3\}$. The second commitment scheme is a perfectly hiding commitment scheme (see Section 4.8.2). For simplicity, we assume that this scheme has a commit phase in which the receiver sends one message to the sender, which then replies with a single message (e.g., Construction 4.8.5). Let us denote by $P_{m,s}(\alpha)$ the (perfectly hiding) commitment of the sender to the string α, upon receiving message m (from the receiver) and when using coins s.

Construction 4.9.1 (A Round-Efficient Zero-Knowledge Proof for $G3C$):

- Common input: *A simple (3-colorable) graph* $G = (V, E)$. *Let* $n \overset{\text{def}}{=} |V|$, $t \overset{\text{def}}{=} n \cdot |E|$, *and* $V = \{1, \ldots, n\}$.
- Auxiliary input to the prover: *A 3-coloring of* G, *denoted* ψ.
- Prover's preliminary step (P0): *The prover invokes the commit phase of the perfectly hiding commitment scheme, which results in sending to the verifier a message* m.
- Verifier's preliminary step (V0): *The verifier uniformly and independently selects a sequence of* t *edges,* $\overline{E} \overset{\text{def}}{=} ((u_1, v_1), \ldots, (u_t, v_t)) \in E^t$, *and sends the prover a random commitment to these edges. Namely, the verifier uniformly selects* $\overline{s} \in \{0, 1\}^{\text{poly}(n)}$ *and sends* $P_{m,\overline{s}}(\overline{E})$ *to the prover.*
- Motivating remark: *At this point the verifier is committed (in a computational sense) to a sequence of* t *edges. Because this commitment is of perfect secrecy, the prover obtains no information about the edge sequence.*
- Prover's step (P1): *The prover uniformly and independently selects* t *permutations,* $\pi_1, \ldots, \pi_t$, *over* $\{1, 2, 3\}$ *and sets* $\phi_j(v) \overset{\text{def}}{=} \pi_j(\psi(v))$ *for each* $v \in V$ *and* $1 \leq j \leq t$. *The prover uses the (perfectly binding, computationally hiding) commitment scheme to commit itself to colors of each of the vertices according to each 3-coloring. Namely, the prover uniformly and independently selects* $s_{1,1}, \ldots, s_{n,t} \in \{0, 1\}^n$, *computes* $c_{i,j} = C_{s_{i,j}}(\phi_j(i))$ *for each* $i \in V$ *and* $1 \leq j \leq t$, *and sends* $c_{1,1}, \ldots, c_{n,t}$ *to the verifier.*
- Verifier's step (V1): *The verifier performs the (canonical) reveal phase of its commitment, yielding the sequence* $\overline{E} = ((u_1, v_1), \ldots, (u_t, v_t))$. *Namely, the verifier sends* $(\overline{s}, \overline{E})$ *to the prover.*
- Motivating remark: *At this point the entire commitment of the verifier is revealed. The verifier now expects to receive, for each* j, *the colors assigned by the* jth *coloring to vertices* u_j *and* v_j *(the endpoints of the* jth *edge in the sequence* $\overline{E}$*).*
- Prover's step (P2): *The prover checks that the message just received from the verifier is indeed a valid revealing of the commitment made by the verifier at Step* V0. *Otherwise the prover halts immediately. Let us denote the sequence of* t *edges, just revealed, by* $(u_1, v_1), \ldots, (u_t, v_t)$. *The prover uses the (canonical) reveal phase of the perfectly binding commitment scheme in order to reveal to the verifier, for each* j, *the* jth *coloring of vertices* u_j *and* v_j. *Namely, the prover sends to the verifier the sequence of quadruples*

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

- Verifier's step (V2): *The verifier checks whether or not, for each* j, *the values in the* jth *quadruple constitute a correct revealing of the commitments* $c_{u_j,j}$ *and* $c_{v_j,j}$ *and whether or not the corresponding values are different. Namely, upon receiving*

$(s_1, \sigma_1, s'_1, \tau_1)$ through $(s_t, \sigma_t, s'_t, \tau_t)$, the verifier checks whether or not for each j it holds that $c_{u_j,j} = C_{s_j}(\sigma_j)$, $c_{v_j,j} = C_{s'_j}(\tau_j)$, and $\sigma_j \neq \tau_j$ (and both σ_j and τ_j are in $\{1, 2, 3\}$). If all conditions hold, then the verifier accepts. Otherwise it rejects.

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

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

The (Basic) Simulation Strategy. We now proceed to show that Construction 4.9.1 is indeed zero-knowledge (in the liberal sense allowing *expected* polynomial-time simulators). For every probabilistic polynomial-time interactive machine V^*, we introduce an expected polynomial-time simulator, denoted M^*, that uses V^* as a black box. The simulator starts by selecting and fixing a random tape r for V^* and by emulating the prover's preliminary Step P0, producing a message m. Given the input graph G, the random tape r, and the preliminary (prover) message m, the commitment message of the verifier V^* is determined. Hence, M^* invokes V^* on input G, random tape r, and message m and gets the corresponding commitment message, denoted CM. The simulator proceeds in two steps.

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

S2. *Generating an interaction that satisfies the query edges* (an oversimplified exposition): Let $(u_1, v_1), \ldots, (u_t, v_t)$ denote the sequence of edges recorded in Step S1. Machine M^* generates a sequence of $n \cdot t$ commitments, $c_{1,1}, \ldots, c_{n,t}$, such that for each $j = 1, \ldots, t$ it holds that $c_{u_j,j}$ and $c_{v_j,j}$ are random commitments to two different random values in $\{1, 2, 3\}$, and all the other $c_{i,j}$'s are random commitments to dummy values (e.g., all values equal 1). The underlying values are called *pseudo-colorings*. The simulator feeds this sequence of commitments to V^*. If V^* replies by revealing correctly the (previously recorded) sequence of edges, then M^* can complete the simulation of a "real" interaction of V^* (by revealing the colors of the endpoints of these recorded edges). Otherwise, the entire Step S2 is repeated (until success is achieved).

For the sake of simplicity, we ignore the preliminary message m in the rest of the analysis. Furthermore, in the rest of the analysis we ignore the possibility that when invoked in Steps S1 and S2 the verifier reveals two different edge commitments. Loosely speaking, this is justified by the fact that during an expected polynomial-time computation, such an event can occur with only negligible probability (since otherwise it contradicts the computational unambiguity of the commitment scheme used by the verifier).

The Running Time of the Oversimplified Simulator. To illustrate the behavior of the simulator, assume that the program V^* always correctly reveals the commitment made in Step V0. In such a case, the simulator will find out the query edges in Step S1, and using them in Step S2 it will simulate the interaction of V^* with the real prover. Using ideas such as in Section 4.4 one can show that the simulation is computationally indistinguishable from the real interaction. Note that in this case Step S2 of the simulator is performed only once.

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

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

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

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

$$\Delta(G, r) \stackrel{\text{def}}{=} q(G, r) \cdot (1 - p(G, r))^{\text{poly}(|G|)/q(G,r)}$$

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

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

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

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

Case 2: For infinitely many n's, it holds that $p(G_n, r_n) < q(G_n, r_n)/2$. It follows that for these n's, we have $|q(G_n, r_n) - p(G_n, r_n)| > \frac{q(G_n,r_n)}{2} > \frac{1}{2P(n)}$, which leads to contradiction of the computational secrecy of the commitment scheme (used by the prover).

Hence, contradiction follows in both cases. ■

Conclusion. We remark that one can modify Construction 4.9.1 so that weaker forms of perfect commitment schemes can be used. We refer specifically to commitment schemes with perfect a posteriori secrecy (see Section 4.8.2). In such schemes the secrecy is established only a posteriori by the receiver disclosing the coin tosses it used in the commit phase. In our case, the prover plays the role of the receiver, and the verifier plays the role of the sender. It suffices to establish the secrecy property a posteriori, because if secrecy is not established, then the verifier will reject. In such a case no harm has been done, because the secrecy of the perfect commitment scheme is used only to establish the soundness of the interactive proof. Thus, using Proposition 4.8.8, we obtain the following:

Corollary 4.9.2: *If non-uniformly claw-free collections exist, then every language in $\mathcal{NP}$ has a round-efficient zero-knowledge proof system.*

4.9.2. Bounding the Power of Cheating Provers

Construction 4.9.1 yields round-efficient zero-knowledge proof systems for $\mathcal{NP}$, under the assumption that claw-free collections exist. Using the seemingly more general assumption that one-way functions exist, we can modify Construction 4.9.1 so as to obtain zero-knowledge computationally sound proof systems. In the modified protocol, we let the verifier use a commitment scheme with computational secrecy, instead of the commitment scheme with perfect secrecy used in Construction 4.9.1. (Hence, both users commit to their messages using a perfectly binding commitment scheme, which offers only computational secrecy.) Furthermore, the commitment scheme used by the prover must have the extra property that it is infeasible to construct a commitment without "knowing" to what value it commits. Such a commitment scheme is called *non-oblivious*. We start by defining and constructing non-oblivious commitment schemes.

4.9.2.1. Non-Oblivious Commitment Schemes

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

Definition 4.9.3 (Non-Oblivious Commitment Schemes): *Let (S, R) be a (perfectly binding) commitment scheme as in Definition 4.4.1. We say that the commitment scheme is* **non-oblivious** *if the prescribed receiver R constitutes a knowledge verifier that is always convinced by S for the relation*

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

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

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

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

Theorem 4.9.4: *If one-way functions exist, then there exist non-oblivious commitment schemes with a constant number of communication rounds. Furthermore, the commitment scheme also preserves the secrecy property when applied (polynomially) many times in parallel.*

The simultaneous secrecy of many copies is crucial to the application in Section 4.9.2.2.

Proof Idea: Recall that (ordinary perfectly binding) commitment schemes can be constructed assuming the existence of one-way functions (see Proposition 4.4.5 and Theorem 3.5.12). Combining such an ordinary commitment scheme with a zero-knowledge proof of knowledge of information allowing a proper decommitment, we get a non-oblivious commitment scheme. (We remark that such proofs do exist under the same assumptions; see Section 4.7.) However, the resulting commitment scheme has an unbounded number of rounds (due to the round complexity of the zero-knowledge proof), whereas we need a bounded-round scheme. We seem to have reached a vicious circle, yet there is a way out: We can use constant-round strong witness-indistinguishable proofs of knowledge, instead of the zero-knowledge proofs (of knowledge). Such proofs do exist under the same assumptions; see Section 4.6 and Exercise 28. The resulting commitment scheme has the additional property that when applied (polynomially) many times in parallel, the secrecy property holds simultaneously in all copies. This fact follows from the parallel-composition lemma for (strong) witness-indistinguishable proofs (see Section 4.6). ■

4.9.2.2. Modifying Construction 4.9.1

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

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

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

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

Using the information-theoretic unambiguity of the commitment scheme employed by the prover, we can talk of a unique color assignment that is induced by the prover's commitments. Using the fact that this commitment scheme is non-oblivious, it follows that the prover can be modified so that in Step P1 it will also output (on its private output tape) the values to which it commits itself at this step. Using this output and relying on the computational secrecy of the verifier's commitment scheme, it follows that the color assignment generated by the prover is almost independent of the verifier's commitment. Hence, the probability that the prover can fool the verifier into accepting an input not in the language is at most negligibly greater than what it would have been if the verifier had asked random queries after the prover made its (color) commitments. The computational soundness of the (modified) protocol follows. (We remark that we do not know if the protocol is computationally sound in the case in which the prover uses a commitment scheme that is not guaranteed to be non-oblivious.[22])

Showing that the (modified) protocol is zero-knowledge is even easier than it was in the first approach (i.e., in Section 4.9.1). The reason is that when demonstrating

[22]Specifically, we do not know how to rule out the possibility that after seeing the verifier's commitment of Step V0, the cheating prover could send some strings at Step P1 such that after the verifier revealed its commitments, the prover could open those strings in a suitable way. To illustrate the problem, suppose that two parties wish to toss a coin by using a (perfectly binding) commitment scheme and that the protocol is as follows: First, the first party commits to a bit, then the second party commits to a bit, next the first party reveals its bit, finally the second party reveals its bit, and the result is defined as the XOR of the two revealed bits. Now, by copying the messages of the first party, the second party can force the outcome always to be zero! Note that this problem does *not* arise when the second party uses a non-oblivious commitment scheme. The problem also does *not* arise when the first party commits via a perfectly hiding commitment scheme (and the second party still uses a perfectly binding commitment scheme). (The latter protocol is analogous to the proof system presented in Section 4.9.1.)

zero-knowledge of such protocols, we use the secrecy of the prover's commitment scheme and the unambiguity of the verifier's commitment scheme. Hence, only these properties of the commitment schemes are relevant to the zero-knowledge property of the protocols. Yet the current (modified) protocol uses commitment schemes with relevant properties that are not weaker than the ones of the corresponding commitment schemes used in Construction 4.9.1. Specifically, the prover's commitment scheme in the modified protocol possesses computational secrecy, just like the prover's commitment scheme in Construction 4.9.1. We stress that this commitment, like the simpler commitment used for the prover in Construction 4.9.1, has the simultaneous-secrecy (of many copies) property. Furthermore, the verifier's commitment scheme in the modified protocol possesses "information-theoretic" unambiguity, whereas the verifier's commitment scheme in Construction 4.9.1 is merely computationally unambiguous. Thus, using Theorem 4.9.4, we have the following:

Corollary 4.9.5: *If non-uniformly one-way functions exist, then every language in $\mathcal{NP}$ has a round-efficient zero-knowledge argument.*

4.9.2.3. An Alternative Construction

An alternative way of deriving Corollary 4.9.5 is by modifying Construction 4.4.7 so as to allow easy simulation, and in particular, robustness under parallel composition. A key ingredient in this modification is the notion of commitment schemes with a "trapdoor property." Loosely speaking, the commit phase of such schemes consists of a receiver message followed by a sender message, so that *given the receiver's private coins* one can efficiently generate strings that are computationally indistinguishable from the sender's message and yet later open these strings so as to reveal any value. Note that this does not contradict the computational-binding property, since the latter refers to cheating senders (that do *not* know the receiver's private coins). We refrain from presenting a formal definition and merely sketch how such schemes can be constructed and used.

Constructing a Trapdoor Commitment Scheme Using Any One-Way Function. Let f be a one-way function, and let $R_f \overset{\text{def}}{=} \{(f(w), w) : w \in \{0,1\}^*\}$ be an $\mathcal{NP}$-relation (corresponding to the $\mathcal{NP}$-set Range(f)). On security parameter n, the receiver selects uniformly $r \in \{0,1\}^n$ and reduces the instance $f(r) \in \text{Range}(f)$ to an instance of the Hamiltonian-cycle (HC) problem, using the standard reduction. The resulting graph is sent to the sender that (not knowing a Hamiltonian cycle is in it) is asked to execute Step P1 in Construction 4.7.14 so that it can respond to a Step-V1 message that equals its input bit (to which it wishes to commit). That is, to commit to the bit 0, the sender sends a matrix of commitments to the entries in the adjacency matrix of a random isomorphic copy of the graph, whereas to commit to the bit 1, the sender sends a matrix of commitments to the entries in the adjacency matrix of a random (simple) n-cycle. Hence, the sender behaves analogously to the simulator of Construction 4.7.14. That is repeated, in parallel, for n times, resulting in a constant-round commitment scheme that is computationally hiding (by virtue of the prover's commitments in Step P1 of Construction 4.7.14) and computationally binding (since otherwise the sender recovers r and so inverts f on input $f(r)$). In contrast, knowledge of r allows one to execute the prover's strategy for Step P1 of Construction 4.7.14 and later open the commitment either way. (Note that the standard reduction of Range(f) to HC is augmented by a polynomial-time computable and invertible mapping of pre-images under f to Hamiltonian cycles in the corresponding reduced graphs.)

Using the Trapdoor Commitment Scheme. One way of using the foregoing scheme toward our goals is to use it for the prover's commitment in (Step P1 of) Construction 4.4.7. To this end, we augment the trapdoor commitment scheme so that before the sender sends its actual commitment (i.e., the message corresponding to Step P1 of Construction 4.7.14) we let the receiver prove that it knows a (corresponding) trapdoor (i.e., a sequence of coins that yields the graph it has sent to the sender). This proof of knowledge need only be witness-hiding, and so it can be carried out in a constant number of rounds. The simulator for the foregoing modification of Construction 4.4.7 first uses the corresponding knowledge extractor (to obtain the trapdoor for the prover's commitments) and then takes advantage of the trapdoor feature to generate false commitments that it can later open any way it needs to (so as to answer the verifier's requests).

4.10.* Non-Interactive Zero-Knowledge Proofs

In this section we consider non-interactive zero-knowledge proof systems. The model consists of three entities: a prover, a verifier, and a uniformly selected sequence of bits (which can be thought of as being selected by a trusted third party). Both verifier and prover can read the random sequence, and each can toss additional coins. The interaction consists of a single message sent from the prover to the verifier, who then is left with the decision (whether to accept or not). Non-interactive zero-knowledge proof systems have various applications (e.g., to encryption schemes secure against chosen message attacks and to signature schemes).

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

4.10.1. Basic Definitions

The model of non-interactive proofs seems closer in spirit to the model of $\mathcal{NP}$-proofs than to general interactive proofs. In a sense, the $\mathcal{NP}$-proof model is extended by allowing the prover and verifier to refer to a common random string, as well as toss coins by themselves. Otherwise, as in the case of $\mathcal{NP}$-proofs, the interaction is minimal (i.e., unidirectional: from the prover to the verifier). Thus, in the definition that follows, both the prover and verifier are ordinary probabilistic machines that, in addition to the common input, also get a uniformly distributed (common) *reference string*. We stress that, in addition to the common input and common reference string, both the prover and verifier can toss coins and get auxiliary inputs. However, for the sake of simplicity, we present a definition for the case in which none of these machines gets an auxiliary input (yet they both can toss additional coins). The verifier also gets as input the output produced by the prover.

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

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

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

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

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

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

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

The uniformly chosen string R is called the **common reference string**.

As usual, the error probability in both conditions can be reduced (from $\frac{1}{3}$) down to $2^{-\text{poly}(|x|)}$ by repeating the process sufficiently many times (using a sequence of many independently chosen reference strings). In stating the soundness condition, we have deviated from the standard formulation that allows $x \notin L$ to be adversarially selected after R is fixed; the latter "adaptive" formulation of soundness is used in Section 4.10.3.2, and it is easy to transform a system satisfying the foregoing ("non-adaptive") soundness condition into one satisfying the adaptive soundness condition (see Section 4.10.3.2).

Every language in $\mathcal{NP}$ has a non-interactive proof system (in which no randomness is used). However, this $\mathcal{NP}$-proof system is unlikely to be zero-knowledge (see Definition 4.10.2).

The definition of zero-knowledge for the non-interactive model is simplified by the fact that because the verifier cannot affect the prover's actions it suffices to consider the simulatability of the view of a single verifier (i.e., the prescribed one). Actually, we can avoid considering the verifier at all (since its view can be generated from the common reference string and the message sent by the prover).

Definition 4.10.2 (Non-Interactive Zero-Knowledge): *A non-interactive proof system* (P, V) *for a language* L *is* **zero-knowledge** *if there exists a polynomial* p *and a probabilistic polynomial-time algorithm* M *such that the ensembles* $\{(x, U_{p(|x|)}, P(x, U_{p(|x|)}))\}_{x \in L}$ *and* $\{M(x)\}_{x \in L}$ *are computationally indistinguishable, where* U_m *is a random variable uniformly distributed over* $\{0, 1\}^m$.

This definition, too, is "non-adaptive" (i.e., the common input cannot depend on the common reference string). An adaptive formulation of zero-knowledge is presented and discussed in Section 4.10.3.2.

Non-Interactive Zero-Knowledge versus Constant-Round Zero-Knowledge. We stress that the non-interactive zero-knowledge model postulates the existence of a uniformly selected reference string available to both prover and verifier. A natural suggestion is to replace this postulate with a two-party protocol for generating a uniformly distributed string of specified length. Such a protocol should be resilient to adversarial behavior by each of the two parties: The output should be uniformly distributed even if one of the parties deviates from the protocol (using any probabilistic polynomial-time strategy). Furthermore, it seems that such a protocol should have a strong simulatability feature, allowing the generation of a random-execution transcript for every given outcome. Specifically, in order to obtain a constant-round zero-knowledge proof system from a non-interactive zero-knowledge proof, one seems to need a constant-round (strongly simulatable) protocol for generating uniformly distributed strings. Such a protocol can be constructed using perfectly hiding commitment schemes. In combination with the results that follow, one can derive an alternative construction of a round-efficient zero-knowledge proof for $\mathcal{NP}$.

4.10.2. Constructions

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

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

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

$$\Pr[V(x, R_I, I, \pi) = 1] \geq \frac{2}{3}$$

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

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

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

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

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

As stated earlier, we do not suggest the hidden-bits model as a realistic model. The importance of the model stems from two facts. First, it is a "clean" model that facilitates the design of proof systems (in it); second, proof systems in the hidden-bits model can be easily transformed into non-interactive proof systems (i.e., the realistic model). The transformation (which utilizes a one-way permutation f with hard-core b) follows.

Construction 4.10.4 (From Hidden-Bits Proof Systems to Non-Interactive Ones): *Let (P, V) be a hidden-bits proof system for L, and suppose that* $f : \{0,1\}^* \to \{0,1\}^*$ *and* $b : \{0,1\}^* \to \{0,1\}$ *are polynomial-time-computable. Furthermore, let $m = \text{poly}(n)$ denote the length of the common reference string for common inputs of length n, and suppose that f is* 1-1 *and length-preserving. Following is a specification of a non-interactive system (P', V'):*

- Common input: $x \in \{0,1\}^n$.
- Common reference string: $s = (s_1, \ldots, s_m)$, *where each s_i is in* $\{0,1\}^n$.
- Prover (denoted P'):

  1. *computes* $r_i = b(f^{-1}(s_i))$ *for* $i = 1, 2, \ldots, m$,

  2. *invokes P to obtain* $(I, \pi) = P(x, r_1 \cdots r_m)$,

  3. *outputs* (I, π, p_I), *where* $p_I \overset{\text{def}}{=} (f^{-1}(s_{i_1}) \cdots f^{-1}(s_{i_t}))$ *for* $I = (i_1, \ldots, i_t)$.

- Verifier (denoted V')*: Given the prover's output* $(I, \pi, (p_1 \cdots p_t))$, *the verifier*

  1. *checks that* $s_{i_j} = f(p_j)$ *for each* $i_j \in I$ *(in case a mismatch is found, V'* halts and rejects*),*

  2. *computes* $r_i = b(p_i)$, *for* $i = 1, \ldots, t$, *lets* $r = r_1, \ldots, r_t$,

  3. *invokes V on* (x, r, I, π) *and accepts if and only if V accepts.*

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

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

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

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

Remark 4.10.6 (Efficient Implementation of P'): As stated earlier, in general, P' cannot be efficiently implemented given black-box access to P. What is needed is the ability (of P') to invert f. On the other hand, for P' to be zero-knowledge, f must be one-way. The obvious solution is to use a family of trapdoor permutations and let the prover know the trapdoor. Furthermore, the family should have the property that its members can be efficiently recognized (i.e., given a description of a function, one can efficiently decide whether or not it is in the family). In other words, P' starts by selecting a permutation f over $\{0, 1\}^n$ such that it knows its trapdoor and proceeds as in Construction 4.10.4, except that it also appends the description of f to the "proof." The verifier acts as in Construction 4.10.4 with respect to the function f specified in the proof. In addition, it checks to see that f is indeed in the family. Both the completeness and the zero-knowledge conditions follow exactly as in the proof of Proposition 4.10.5. For the soundness condition, we need to consider all possible members of the family (without loss of generality, there are at most 2^n such permutations). For each such permutation, the argument is as before, and our claim thus follows by a counting argument (as applied in Section 4.10.3.2).[23] The construction can be extended to arbitrary trapdoor permutations; details omitted.

We now turn to the construction of proof systems in the Hidden-Bits model. Specifically, we are going to construct a proof system for the *Hamiltonian-Cycle (HC)* problem that is $\mathcal{NP}$-complete (and thus get proof systems for any language in $\mathcal{NP}$). We consider

[23]Actually, we also need to repeat the (P, V) system $O(n)$ times, so as first to reduce the soundness error to $\frac{1}{3} \cdot 2^{-n}$.

directed graphs (and the existence of directed Hamiltonian cycles). Next, we present a basic zero-knowledge system in which Hamiltonian graphs are accepted with probability 1, whereas non-Hamiltonian graphs on n vertices are rejected with probability $\Omega(n^{-3/2})$. (This system builds on the one presented in Construction 4.7.14.)

Construction 4.10.7 (A Hidden-Bits System for HC):

- Common input: *A directed graph $G = (V, E)$, with $n \stackrel{\text{def}}{=} |V|$.*
- Common reference string: *Viewed as an n^3-by-n^3 Boolean matrix M, with each entry being* 1 *with probability n^{-5}.*

  This is implemented by breaking the common reference string into blocks of length $5 \log_2 n$ and setting a matrix entry to 1 if and only if the corresponding block is all 1's.

- Definitions: *A* **permutation matrix** *is a matrix in which each row (*resp., column*) contains a single entry of value* 1. *A* **Hamiltonian matrix** *is a permutation matrix that corresponds to a simple directed cycle going through all rows and columns.* (That is, the corresponding directed graph consists of a single Hamiltonian cycle.)

  An n^3-by-n^3 matrix M is called useful if it contains a generalized n-by-n Hamiltonian sub-matrix and all other $n^6 - n^2$ entries in M are 0. That is, a useful n^3-by-n^3 matrix contains exactly n 1-entries that form a simple n-cycle, $\{(\phi_1(i), \phi_2((i \bmod n) + 1)) : i = 1, \ldots, n\}$, where ϕ_1 and ϕ_2 are 1-1 mappings of $\{1, \ldots, n\}$ to $\{1, \ldots, n^3\}$.

- Prover: *Let C be a Hamiltonian cycle in G, in case such exists. The prover examines the matrix M and acts according to the following two cases:*

  Case 1: M is useful. *Let H denote its Hamiltonian n-by-n sub-matrix and C_H the corresponding Hamiltonian cycle in H.*

  - *The prover* reveals *all* $(n^6 - n^2)$ *entries in M that are not in H.*
  - *The prover* finds *a* 1-1 *mapping, π_1, of V to the rows of H and a* 1-1 *mapping, π_2, of V to the columns of H, so that the edges of C are mapped to the* 1*-entries of H.*

    (Directed pairs of vertices of G, being edges or not, are mapped in the natural manner; that is, (u, v) is mapped to the matrix entry $(\pi_1(u), \pi_2(v))$. The mapping pair (π_1, π_2) is required to be an "isomorphism" of C to C_H.[24] Actually, we should specify one isomorphism among the n possible ones.)

  - *The prover* reveals *the $(n^2 - |E|)$ entries corresponding to non-edges of G.*

    (The correspondence is by the preceding mappings. That is, entry $(\pi_1(u), \pi_2(v))$ is revealed if and only if $(u, v) \in V \times V \setminus E$.)

[24] The minor technicality that prevents us from freely using the term "isomorphism" is that H is not a graph.

- *The prover* outputs *the mapping pair* (π_1, π_2) (as a certificate).

In total, $n^6 - |E|$ entries are revealed, all being 0-entries, and the certificate is (π_1, π_2).

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

(No certificate is provided in this case.)

- Verifier: *Given the revealed entries and possibly a certificate, the verifier acts according to the following two cases:*

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

Case 2: *The prover has revealed all of* M. *In this case the verifier accepts if and only if M is* not *useful.*

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

Fact 4.10.8: $\Pr[M \textit{ is useful}] = \Omega(n^{-3/2})$.

Proof Sketch: The expected number of 1-entries in M equals $(n^3)^2 \cdot n^{-5} = n$. Furthermore, with probability $\Theta(1/\sqrt{n})$ the matrix M contains exactly n entries of value 1. Considering any row of M, observe that with probability at most $\binom{n^3}{2} \cdot (n^{-5})^2 < n^{-4}$ this row contains more than a single 1-entry. Thus, with probability at least $1 - 2n^3 \cdot n^{-4} = 1 - O(n^{-1})$ the rows and columns of M each contain at most a single 1-entry. Combining these two facts, it follows that with probability $\Omega(1/\sqrt{n})$ the matrix M contains an n-by-n permutation sub-matrix and all the other entries of M are 0. Now observe that there are $n!$ (n-by-n) permutation matrices, and $(n-1)!$ of them are Hamiltonian matrices. Thus, conditioned on M containing an n-by-n permutation sub-matrix (and zeros elsewhere), with probability $1/n$ the matrix M is useful. ■

Proposition 4.10.9: *There exists a* (perfect) *zero-knowledge Hidden-Bits proof system for Graph Hamiltonicity. Furthermore, the prover can be implemented by a polynomial-time machine that gets a Hamiltonian cycle as auxiliary input.*

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

Next, suppose that G is non-Hamiltonian. By Fact 4.10.8, with probability at least $\Omega(n^{-3/2})$, the matrix M is useful. Fixing any useful matrix M, we show that the verifier rejects G, no matter what the prover does. Clearly, if the prover behaves as in Case 2, then the verifier rejects (since M is useful). Thus we focus on the case in which the prover outputs a pair of matchings (π_1, π_2) (as in Case 1). Let H denote the (unique) n-by-n Hamiltonian sub-matrix of M, and consider the following sub-cases:

1. $\pi_1(V) \times \pi_2(V)$ does not equal H. Because the prover must reveal all entries not in the sub-matrix $\pi_1(V) \times \pi_2(V)$, it follows that it must reveal some row or column of H. But such a row or column must contain a 1-entry, and so the verifier will reject.

2. Otherwise, $\pi_1(V) \times \pi_2(V) = H$. Also, each non-edge of G must be mapped to a 0-entry of H (or else the verifier will reject). It follows that the pre-image of each 1-entry in H must be an edge in G, which implies that G has a Hamiltonian cycle (in contradiction to our hypothesis).

We conclude that in case G is non-Hamiltonian, it is rejected with probability $\Omega(n^{-3/2})$.

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

Using Propositions 4.10.9 and 4.10.5 and Remark 4.10.6, we conclude the following:

Theorem 4.10.10: *Assuming the existence of one-way permutations,*[25] *each language in $\mathcal{NP}$ has a zero-knowledge non-interactive proof system. Furthermore, assuming the existence of families of trapdoor permutations, each language in $\mathcal{NP}$ has a zero-knowledge non-interactive proof system in which the prover can*

[25] As usual in this chapter, here and later, we mean constructs for which the hardness requirement also holds with respect to non-uniform (polynomial-size) circuits.

be implemented by a probabilistic polynomial-time machine that gets an $\mathcal{NP}$-witness as auxiliary input.

4.10.3. Extensions

We present the two extensions mentioned at the beginning of this section: First we consider proof systems that preserve zero-knowledge when applied polynomially many times (with the same common reference string), and later we consider proof systems that preserve security when the assertions (i.e., common inputs) are adversarially selected after the common reference string has been fixed.

4.10.3.1. Proving Many Assertions of Varying Lengths

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

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

We now turn to the actual definitions. First we note that nothing needs to be changed regarding the definition of non-interactive proof systems (Definition 4.10.1). We still require the ability to be convinced by valid assertions, as well as "protection" from false assertions. Alas, a minor technical difference is that whereas in Definition 4.10.1 we denoted by n the length of the assertion and considered a common reference string of length poly(n), here we let n denote the length of the common reference string used for assertions of length n^{ε}. We call ε the *fundamental constant* of the proof system. In contrast, the definition of zero-knowledge has to be extended to handle an (a priori) unbounded sequence of proofs. (Recall that U_n denotes a random variable, uniformly distributed over $\{0, 1\}^n$.)

Definition 4.10.11 (Non-Interactive Zero-Knowledge, Unbounded Version): *A non-interactive proof system (P, V), with fundamental constant ε, for a* *language L is* **unboundedly zero-knowledge** *if for every polynomial p there exists a probabilistic polynomial-time algorithm M such that the following two ensembles are computationally indistinguishable:*

1. $\{((x_1, \ldots, x_{p(n)}), U_n, (P(x_1, U_n), \ldots, P(x_{p(n)}, U_n)))\}_{x_1, \ldots, x_{p(n)} \in L_{n^\varepsilon}}$
2. $\{M(x_1, \ldots, x_{p(n)})\}_{x_1, \ldots, x_{p(n)} \in L_{n^\varepsilon}}$

   where $L_\ell \stackrel{\text{def}}{=} L \cap \{0,1\}^\ell$.

We comment that the non-interactive proof systems presented earlier (e.g., Construction 4.10.4) are not unboundedly zero-knowledge; see Exercise 34.

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

1. The assertion holds for a negligible fraction of the strings of the same length. Thus, adding this potential ability does not significantly affect the soundness condition.
2. Strings satisfying the assertion are computationally indistinguishable from uniformly distributed strings of the same length. Thus, it will be acceptable for the simulator to use such strings, rather than uniformly chosen ones (used in the real proof system).
3. The decision problem for the assertion is in $\mathcal{NP}$. This will allow a reduction to an $\mathcal{NP}$-complete problem.

An immediate assertion, concerning strings, that comes to mind is being produced by a pseudorandom generator. This yields the following construction, where G denotes such a generator.

Construction 4.10.12 (An Unboundedly Zero-Knowledge Non-Interactive Proof System): *Let $G : \{0,1\}^\ell \to \{0,1\}^{2\ell}$, let L_1 be an $\mathcal{NP}$-complete language, let L be an arbitrary $\mathcal{NP}$ language, and consider the following $\mathcal{NP}$ language:*

$$L_2 \stackrel{\text{def}}{=} \{(x, p) : x \in L \bigvee \exists w' \in \{0,1\}^{|x|} \text{ s.t. } G(w') = p\}$$

Consider a standard reduction of L_2 to L_1, and let q be a polynomial such that 3ℓ-bit-long instances of L_2 are mapped to $q(\ell)$-bit-long instances of L_1. Let (P, V) be an ordinary non-interactive proof system for L_1, and suppose that for some polynomial q' the system (P, V) uses a common reference string of length $q'(\ell)$ for assertions of length $q(\ell)$. Suppose that P takes as auxiliary input an $\mathcal{NP}$-witness for membership in L_1, and let $n = q'(\ell) + 2\ell$. Following is a specification of a non-interactive proof system for $L \in \mathcal{NP}$:

- Common input: $x \in \{0,1\}^\ell$.
- Common reference string: $r = (p, s)$, *where* $p \in \{0,1\}^{2\ell}$ *and* $s \in \{0,1\}^{n-2\ell}$.

- Prover:
  1. *Using a standard reduction of L_2 to L_1, the prover reduces $(x, p) \in \{0,1\}^{\ell+2\ell}$ to $y \in \{0,1\}^{q(\ell)}$. In addition, when given an $\mathcal{NP}$-witness u for $x \in L$, the prover reduces*[26] *u to a witness, denoted w, for $y \in L_1$.*
  2. *The prover invokes P on common input y, auxiliary input w, and common reference string s, obtaining output π, which it outputs/sends.*
- Verifier:
  1. *Reduces (x, p) to y using the same standard reduction of L_2 to L_1.*
  2. *Invokes V on common input y, common reference string s, and prover's output π, and decides as V does.*

Note that the reduction maps $(\ell + 2\ell)$-bit-long instances of L_2 to instances of L_1 having length $q(\ell)$. Recall that by the hypothesis, the proof system (P, V) handles L_1 instances of length $q(\ell)$ by using a reference string of length $q'(\ell) = n - 2\ell$, which exactly matches the length of s. Let $\varepsilon > 0$ be a constant satisfying $n^\varepsilon \leq \ell$ (i.e., $(2\ell + q'(\ell))^\varepsilon \leq \ell$). Then we have the following:

Proposition 4.10.13: *Let (P, V) be as before, and let G be a pseudorandom generator. Furthermore, suppose that P is zero-knowledge and that when given an $\mathcal{NP}$-witness as auxiliary input, it can be implemented in probabilistic polynomial time. Then Construction 4.10.12 constitutes an unboundedly zero-knowledge non-interactive proof system for L, with fundamental constant ε. Furthermore, the prover can be implemented by a probabilistic polynomial-time machine that gets an $\mathcal{NP}$-witness as auxiliary input.*

Proof Sketch: The completeness and efficiency claims for the new prover follow immediately from the hypotheses concerning (P, V). The soundness condition follows by observing that the probability that p is in the range of G is at most $2^{-\ell}$ (and relying on the soundness of (P, V)). To prove the zero-knowledge property, we construct a simulator as follows. The simulator uniformly selects $u' \in \{0,1\}^\ell$ and $s \in \{0,1\}^{n-2\ell}$, sets $p = G(u')$, and handles each instance $x \in \{0,1\}^\ell$ in a sequence of L instances as follows: The simulator emulates the prover's program (on input x), except that it uses u' as the $\mathcal{NP}$-witness for $(x, p) \in L_2$. Namely, the simulator reduces $(x, p) \in L_2$ to $y' \in L_1$, along with reducing the $\mathcal{NP}$-witness u' to a witness w' (for y'). Next, the simulator invokes P on common input y', auxiliary input w', and common reference string s. Thus, when given a sequence of instances $\overline{x} = (x_1, \ldots, x_t)$, the simulator outputs $(\overline{x}, (p, s), P_{w'}(y'_1, s), \ldots, P_{w'}(y'_t, s))$, where y_i is the result of applying the reduction to (x_i, p). Note that the efficiency of the simulator relies on the efficient implementation of P (and on the efficiency of G). To prove that the simulator's output is computationally indistinguishable from the verifier's view,

[26]We again use the fact that the standard reductions are coupled with an adequate witness-reduction (see Exercise 16).

we combine the following two observations (which also rely on the efficient implementation of P):

1. *The distributions of the common reference string* are indeed very different in the two cases (i.e., real execution versus simulator's output). Yet, by the pseudorandomness of G, this difference is computationally indistinguishable. Thus, the verifier's view in real execution is computationally indistinguishable from its view in the case in which the common reference string is selected exactly as in the simulation (but the prover acts as in Construction 4.10.12).
2. The zero-knowledge property of P implies that P is witness-indistinguishable (as defined in Section 4.6). Thus, one cannot distinguish the case in which P uses a witness for $x \in L$ (as in Construction 4.10.12) from the case in which P uses as witness a seed for the pseudorandom sequence p (as done by the simulator). The same holds when repeating the proving process polynomially many times.

In other words, the zero-knowledge claim is proved by using a hybrid argument, where the (single) intermediate hybrid corresponds to executing the prover strategy (as is) on a pseudorandom reference string as produced by the simulator (rather than on a truly random reference string). These two observations establish that this intermediate hybrid is computationally indistinguishable from both of the extreme hybrids (which are the ensembles we wish to relate). ■

Using Theorem 4.10.10 and Proposition 4.10.13, we obtain the following:

Theorem 4.10.14: *Assuming the existence of families of trapdoor permutations,*[27] *each language in $\mathcal{NP}$ has an* unboundedly *zero-knowledge non-interactive proof system. Furthermore, the prover can be implemented by a probabilistic polynomial-time machine that gets an $\mathcal{NP}$-witness as auxiliary input.*

4.10.3.2. Adaptive Zero-Knowledge

As mentioned in Section 4.10.1, the definitions used thus far are non-adaptive. This refers to both the soundness and the zero-knowledge conditions. (The same applies also to the completeness condition; but because all commonly used schemes have perfect completeness,[28] this issue is of little interest). In the adaptive analogies, the common input is adversarially selected after the common reference string is fixed. The formulation of adaptive soundness is straightforward, and we call the reader's attention to the formulation of adaptive zero-knowledge.

Definition 4.10.15 (Non-Interactive Zero-Knowledge Proofs, Adaptive Version): *Let (P, V) be a non-interactive proof system for a language L (i.e., as in Definition 4.10.1).*

[27] See footnote 25.

[28] That is, for every $x \in L$, it actually holds that $\Pr[V(x, R, P(x, R)) = 1] = 1$.

- Adaptive soundness: *We say that (P, V) is* **adaptively sound** *if for every n and every pair of functions $\Xi : \{0,1\}^{\text{poly}(n)} \to (\{0,1\}^n \setminus L)$ and $\Pi : \{0,1\}^{\text{poly}(n)} \to \{0,1\}^{\text{poly}(n)}$,*

$$\Pr[V(\Xi(R), R, \Pi(R)) = 1] \leq \frac{1}{3}$$

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

- Adaptive zero-knowledge: *We say that (P, V) is* **adaptively zero-knowledge** *if there exist two probabilistic polynomial-time algorithms M_1 and M_2 such that for every function $\Xi : \{0,1\}^{\text{poly}(n)} \to (\{0,1\}^n \cap L)$ the ensembles $\{(R_n, \Xi(R_n), P(\Xi(R_n), R_n))\}_{n\in\mathbb{N}}$, and $\{M^{\Xi}(1^n)\}_{n\in\mathbb{N}}$ are computationally indistinguishable, where R_n is a random variable uniformly distributed in $\{0,1\}^{\text{poly}(n)}$, and $M^{\Xi}(1^n)$ denotes the output of the following randomized process:*

  1. $(r, s) \leftarrow M_1(1^n)$

  2. $x \leftarrow \Xi(r)$

  3. $\pi \leftarrow M_2(x, s)$

  4. Output (r, x, π)

  (That is, M_1 generates a pair (r, s) consisting of a supposedly common reference string r and auxiliary information s to be used by M_2. The latter, given an adaptively selected input x and the auxiliary information s, generates an alleged proof π. We stress that x can depend on r, but not on s.)

As usual, the error probability (in the adaptive-soundness condition) can be reduced (from $\frac{1}{3}$) down to $2^{-\text{poly}(|x|)}$. Also, any non-interactive proof system (i.e., of non-adaptive soundness) can be transformed into a system that is adaptively sound by merely reducing the error probability and applying the union bound; that is, for every $\Xi : \{0,1\}^{\text{poly}(n)} \to (\{0,1\}^n \setminus L)$ and $\Pi : \{0,1\}^{\text{poly}(n)} \to \{0,1\}^{\text{poly}(n)}$, we have

$$\begin{aligned}\Pr[V(\Xi(R), R, \Pi(R)) = 1] &\leq \sum_{x\in\{0,1\}^n\setminus L} \Pr[V(x, R, \Pi(R)) = 1] \\ &\leq 2^n \cdot \max_{x\in\{0,1\}^n\setminus L} \{\Pr[V(x, R, \Pi(R)) = 1]\}\end{aligned}$$

In contrast to the foregoing trivial transformation (from non-adaptive to adaptive soundness), we do not know of a simple transformation of non-interactive zero-knowledge proofs into ones that are adaptively zero-knowledge. Fortunately, however, the exposition in Section 4.10.2 extends to the adaptive setting. (The key idea is that the reference string in these proof systems can be generated obliviously of the common input.[29]) We obtain the following:

Theorem 4.10.16: *Assuming the existence of one-way permutations,[30] each language in $\mathcal{NP}$ has a non-interactive proof system that is adaptively*

[29]Specifically, this is obvious for the simulator presented in the proof of Proposition 4.10.9. We stress that this simulator determines the values of all hidden bits independently of the common input (i.e., either they form a random unuseful matrix or they are "effectively" all zeros). The simulator for the proof of Proposition 4.10.5 can be easily modified to work for such hidden-bit model simulators.

[30]See footnote 25.

zero-knowledge. Furthermore, assuming the existence of families of trapdoor permutations, the prover strategy in such a proof system can be implemented by a probabilistic polynomial-time machine that gets an $\mathcal{NP}$-witness as auxiliary input.

The "furthermore" statement extends to a model that allows the adaptive selection of polynomially many assertions (i.e., a model that combines the two extensions discussed in this subsection).

4.11.* Multi-Prover Zero-Knowledge Proofs

In this section we consider an extension of the notion of an interactive proof system. Specifically, we consider the interaction of a verifier with more than one prover (say, two provers). The provers can share an a-priori-selected strategy, but it is assumed that they cannot interact with each other during the time period in which they interact with the verifier. Intuitively, the provers can coordinate their strategies prior to, but not during, their interrogation by the verifier. Indeed, the multi-prover model is reminiscent of the common police procedure of isolating suspected collaborators and interrogating each of them separately. We discuss one realistic (digital) setting in which this model is applicable.

The notion of a multi-prover interactive proof plays a fundamental role in complexity theory. That aspect is not addressed here. In the current section we merely address the zero-knowledge aspects of multi-party interactive proofs. Most importantly, the multi-prover model enables the construction of (perfect) zero-knowledge proof systems for $\mathcal{NP}$, *independent of any complexity-theoretic assumptions*.

4.11.1. Definitions

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

4.11.1.1. The Two-Partner Model

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

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

- *The input tapes of all three parties coincide, and their content is called the* **common input**.
- *The random tapes of the two partners coincide and are called the* **partners' random tape**. *(The solitary has a separate random tape.)*
- *The solitary has two pairs of communication tapes and two switch tapes, instead of a single pair of communication tapes and a single switch tape (as in Definition 4.2.1).*
- *The two partners have the same identity, and the solitary has an opposite identity (see Definitions 4.2.1 and 4.2.2).*
- *The first (resp., second) switch tape of the solitary coincides with the switch tape of the first (resp., second) partner, and the first (resp., second) read-only communication tape of the solitary coincides with the write-only communication tape of the first (resp., second) partner, and vice versa.*
- *The* **joint computation** *of the three parties, on a common input x, is a sequence of triplets. Each triplet consists of the local configurations of the three machines. The behavior of each partner-solitary pair is as in the definition of the joint computation of a pair of interactive machines.*

We denote by $\langle P_1, P_2, S\rangle(x)$ the output of the solitary S after interacting with the partners P_1 and P_2, on common input x.

4.11.1.2. Two-Prover Interactive Proofs

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

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

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

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

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

$$\Pr[\langle B_1, B_2, V\rangle(x) = 1] \leq \frac{1}{3}$$

As usual, the error probability in both conditions can be reduced (from $\frac{1}{3}$) down to $2^{-\text{poly}(|x|)}$ by *sequentially* repeating the protocol sufficiently many times. Error reduction via *parallel* repetitions is problematic (in general) in this context; see the suggestions for further reading at the end of the chapter.

The notion of zero-knowledge (for multi-prover systems) remains exactly as in the one-prover case. Actually, we make the definition of perfect zero-knowledge more strict by requiring that the simulator never fail (i.e., never outputs the special symbol $\perp$).[31] Namely:

Definition 4.11.3: *We say that a (two-prover) proof system* (P_1, P_2, V) *for a language* L *is* **perfect zero-knowledge** *if for every probabilistic polynomial-time interactive machine* V^* *there exists a probabilistic polynomial-time algorithm* M^* *such that for every* $x \in L$ *the random variables* $\langle P_1, P_2, V^* \rangle(x)$ *and* $M^*(x)$ *are identically distributed.*

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

4.11.2. Two-Sender Commitment Schemes

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

4.11.2.1. A Definition

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

1. *Secrecy*: At the end of the *commit phase*, the solitary, called the *receiver*, does not gain any *information* about the senders' value.
2. *Unambiguity*: Suppose that the commit phase is successfully completed. Then if later the senders can perform the *reveal phase* such that the receiver accepts the value 0 with probability p, then they cannot perform the *reveal phase* such that the receiver accepts the value 1 with probability substantially greater than $1 - p$. We stress that no interaction is allowed between the senders throughout the *entire* commit and reveal process. (We comment that for every p the senders can always conduct the commit phase such that they can later reveal the value 0 with probability p and the value 1 with probability $1 - p$. See Exercise 35.)

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

[31] Recall that in Definition 4.3.1, the simulator was allowed to fail (with probability at most $\frac{1}{2}$).

phase the second sender sends the content of the joint random tape (used by the first sender in the commit phase) to the receiver. We stress again that the two senders cannot exchange information between themselves throughout the entire commit and reveal process; thus, in particular, the second sender does not know the messages sent by the receiver to the first sender during the commit phase.

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

- Input specification: *The common input is an integer n presented in unary, called the* **security parameter**. *The two partners, called the* **senders**, *have an auxiliary private input $v \in \{0, 1\}$.*
- Secrecy: *The 0-commitment and the 1-commitment are identically distributed. Namely, for every probabilistic (not necessarily polynomial-time) machine R^* interacting with the first sender (i.e., S_1), the random variables $\langle S_1(0), R^*\rangle(1^n)$ and $\langle S_1(1), R^*\rangle(1^n)$ are identically distributed.*
- Unambiguity: *Preliminaries: For simplicity, $v \in \{0, 1\}$ and $n \in \mathbb{N}$ are implicit in all notations.*

1. *As in Definition 4.4.1, a* receiver's view of an interaction *with the (first) sender, denoted $(r, \overline{m})$, consists of the random coins used by the receiver, denoted r, and the sequence of messages received from the (first) sender, denoted $\overline{m}$.*

2. *Let $\sigma \in \{0, 1\}$. We say that the string s is a* **possible σ-opening** *of the receiver's view $(r, \overline{m})$ if $\overline{m}$ describes the messages received by R when R uses local coins r and interacts with machine S_1, which uses local coins s and input $(\sigma, 1^n)$.*

3. *Let S_1^* be an arbitrary program for the first sender. Let p be a real and $\sigma \in \{0, 1\}$. We say that p is an* **upper bound on the probability of a σ-opening of the receiver's view of the interaction with S_1^*** *if for every random variable X* (representing the string sent by the second sender in the reveal phase)*, which is statistically independent of the receiver's coin tosses, the probability that X is a possible σ-opening of the receiver's view of an interaction with S_1^* is at most p. That is,*

$$\mathsf{Pr}[X \textit{ is a } \sigma\textit{-opening of } \langle S_1^*, R\rangle(1^n)] \leq p$$

(The probability is taken over the coin tosses of the receiver, the strategy S_1^*, and the random variable X.)

4. *Let S_1^* be as before, and for each $\sigma \in \{0, 1\}$ let p_σ be an upper bound on the probability of a σ-opening of the receiver's view of the interaction with S_1^*. We say that* **the receiver's view of the interaction with S_1^* is unambiguous** *if $p_0 + p_1 \leq 1 + 2^{-n}$.*

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

In the formulation of the unambiguity requirement, the random variables X represent possible strategies of the second sender. Such a strategy may depend on the random input that is shared by the two senders, but is independent of the receiver's random coins (since information on these coins, if any, is only sent to the first sender). The strategies employed by the two senders determine, for each possible coin-tossing of the receiver, a pair of probabilities corresponding to their success in a 0-opening and a 1-opening. (In fact, bounds on these probabilities are determined merely by the strategy of the first sender.) The unambiguity condition asserts that the average of these pairs, taken over all possible receiver's coin tosses, is a pair that sums up to at most $1 + 2^{-n}$. Intuitively, this means that the senders cannot do more harm than deciding at random whether to commit to 0 or to 1. Both the secrecy and unambiguity requirements are information-theoretic (in the sense that no computational restrictions are placed on the adversarial strategies). We stress that we have implicitly assumed that the reveal phase takes the following canonical form:

1. The second sender sends to the receiver the initial private input v and the random coins s used by the first sender in the commit phase.
2. The receiver verifies that v and s (together with the private coins (i.e., r) used by R in the commit phase) indeed yield the messages that R has received in the commit phase. Verification is done in polynomial time (by running the programs S_1 and R).

Consider the pairs (p_0, p_1) assigned to each strategy S_1^* in the unambiguity condition of Definition 4.11.4. We note that the highest possible value of $p_0 + p_1$ is attainable by deterministic strategies for both senders.[32] Thus, it suffices to consider an arbitrary deterministic strategy S_1^* for the first sender and fixed σ-openings, denoted s^σ, for $\sigma \in \{0, 1\}$. The unambiguity condition thus says that for every such S_1^*, s^0, and s^1,

$$\sum_{\sigma\in\{0,1\}} \mathsf{Pr}[s^\sigma \text{ is a } \sigma\text{-opening of } \langle S_1^*, R\rangle(1^n)] \leq 1 + 2^{-n}$$

In fact, for the construction presented next, we shall establish a stronger condition:

Strong unambiguity condition: *For every deterministic strategy S_1^* and every pair of strings (s^0, s^1),*

$$\mathsf{Pr}[\forall \sigma \in \{0, 1\},\ \ s^\sigma \text{ is a } \sigma\text{-opening of } \langle S_1^*, R\rangle(1^n)] \leq 2^{-n}$$

(Clearly, if the unambiguity condition is violated, then so is the strong unambiguity condition.)

4.11.2.2. A Construction

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

[32] We use an averaging argument. First note that for every (probabilistic) S_1^* and σ there exists a string s^σ maximizing the probability that any fixed string is a σ-opening of $\langle S_1^*, R\rangle(1^n)$. Thus, the probability that s^σ is a σ-opening of $\langle S_1^*, R\rangle(1^n)$ is an upper bound on the probability that X (as in the definition) is a σ-opening of $\langle S_1^*, R\rangle(1^n)$. Similarly, fixing such a pair (s^0, s^1), we view S_1^* as a distribution over deterministic strategies for the first sender and consider the sum of the two probabilities assigned to each such strategy S_1^{**}. Thus, there exists a deterministic strategy S_1^{**} for which this sum is at least as large as the sum associated with S_1^*.

Construction 4.11.5 (A Two-Sender Bit Commitment):

- Preliminaries: *Let π_0 and π_1 denote two fixed permutations over* $\{0, 1, 2\}$ *such that π_0 is the identity permutation and π_1 is a permutation consisting of a single transposition, say* $(1, 2)$. *Namely,* $\pi_1(1) = 2$, $\pi_1(2) = 1$, *and* $\pi_1(0) = 0$.
- Common input: *The security parameter n (in unary).*
- Sender's input: $\sigma \in \{0, 1\}$.
- A convention: *Suppose that the content of the senders' random tape encodes a uniformly selected* $\overline{s} = s_1 \cdots s_n \in \{0, 1, 2\}^n$.
- Commit phase:

  1. The receiver uniformly selects $\overline{r} = r_1 \cdots r_n \in \{0, 1\}^n$ *and sends* $\overline{r}$ *to the first sender.*

  2. For each i, the first sender computes $c_i \overset{\text{def}}{=} \pi_{r_i}(s_i) + \sigma \bmod 3$ *and sends* $c_1 \cdots c_n$ *to the receiver.*

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

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

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

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

We now turn to the actual proof of the strong unambiguity property. The arbitrary (deterministic) strategy of the first sender is captured by a function, denoted f, mapping n-bit-long strings into sequences in $\{0, 1, 2\}^n$. Thus, the receiver's view, when using coin sequence $\overline{r} = r_1 \cdots r_n \in \{0, 1\}^n$, consists of $(\overline{r}, f(\overline{r}))$. Let $\overline{s}^0$ and $\overline{s}^1$ denote arbitrary opening attempts (i.e., 0-opening and 1-opening, respectively) of the second sender. Without loss of generality, we can assume that both $\overline{s}^0$ and $\overline{s}^1$ are in $\{0,1,2\}^n$ and let $\overline{s}^\sigma = s_1^\sigma \cdots s_n^\sigma$ (with $s_j^\sigma \in \{0,1,2\}$). The strong unambiguity property asserts that for a uniformly selected $\overline{r} \in \{0,1\}^n$ the probability that $\overline{s}^0$ and $\overline{s}^1$ are 0-opening and 1-opening, respectively, of the receiver's view $(\overline{r}, f(\overline{r}))$ is at most 2^{-n}.

Let us denote by R^σ the set of all strings $\overline{r} \in \{0,1\}^n$ for which the sequence $\overline{s}^\sigma$ is a possible σ-opening of the receiver's view $(\overline{r}, f(\overline{r}))$. Namely,

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

where $\overline{r} = r_1 \cdots r_n$, and $f(\overline{r}) = f_1(\overline{r}) \cdots f_n(\overline{r})$. Then the strong unambiguity property asserts that $|R^0 \cap R^1| \leq 2^{-n} \cdot |\{0,1\}^n|$. That is:

Claim 4.11.6.1: $|R^0 \cap R^1| \leq 1$.

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

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

Contradiction follows as in the motivating discussion. That is, using the first two equations and the fact that π_0 is the identity, we have $s_i^1 + 1 \equiv s_i^0 \pmod 3$, and combining this with the last two equations, we have

$$\pi_1\left(s_i^1 + 1\right) = \pi_1\left(s_i^0\right) \equiv \pi_1\left(s_i^1\right) + 1 \pmod 3$$

in contradiction to the (readily verified) fact that $\pi_1(s + 1 \bmod 3) \not\equiv \pi_1(s) + 1 \pmod 3$ for every $s \in \{0,1,2\}$. □

This completes the proof of the proposition. ■

Remark 4.11.7 (Parallel Executions). The proof extends to the case in which many instances of the protocol are executed in parallel. In particular, by t parallel executions of Construction 4.11.5, we obtain a two-sender commitment scheme for t-bit-long strings. Note that we are content in asserting that the probability that the verifier's view has two conflicting openings is at most 2^{-n} (or even $t \cdot 2^{-n}$), rather than seeking error reduction (i.e., a probability bound of $2^{-t \cdot n}$).

4.11.3. Perfect Zero-Knowledge for $\mathcal{NP}$

Two-prover perfect zero-knowledge proof systems for any language in $\mathcal{NP}$ follow easily by modifying Construction 4.4.7. The modification consists of replacing the bit-commitment scheme used in Construction 4.4.7 with the two-sender bit-commitment scheme of Construction 4.11.5. Specifically, the modified proof system for Graph Coloring proceeds as follows.

Two-Prover Atomic Proof of Graph Coloring

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

2. The verifier uniformly selects an edge and sends it to the second prover. In response, the second prover reveals the colors of the endpoints of the required edge by sending the portions of the prover's random tape used in the corresponding instance of the commit phase.

As usual, one can see that the provers can always convince the verifier of valid claims (i.e., the completeness condition holds). Using the unambiguity property of the two-sender commitment scheme (and ignoring the 2^{-n} deviation from the "perfect case"), we can think of the first prover as selecting at random, with *arbitrary* probability distribution, a color assignment to the vertices of the graph. We stress that this claim holds although many instances of the commit protocol are performed concurrently (see Remark 4.11.7). If the graph is not 3-colored, then each of the possible color assignments chosen by the first prover is illegal, and a weak soundness property follows. Yet, by executing this protocol polynomially many times, even in parallel, we derive a protocol satisfying the soundness requirement. We stress that the fact that parallelism is effective here (as means for decreasing error probability) follows from the unambiguity property of the two-sender commitment scheme and *not* from a general "parallel-composition lemma" (which is highly non-trivial in the two-prover setting).

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

Simulation of the Atomic Proof of Graph Coloring

1. The simulator generates random "commitments to nothing." Namely, the simulator invokes the verifier and answers the verifier's messages that belong to the commit phase by a sequence of uniformly chosen strings over $\{0, 1, 2\}$.

2. Upon receiving the query-edge (u, v) from the verifier, the simulator uniformly selects two different colors, ϕ_u and ϕ_v, and opens the corresponding commitments so as to reveal these values. The simulator has no difficulty in doing so, because, unlike the second prover, it knows the messages sent by the verifier in the commit phase. Specifically, given the receiver's view of the commit phase, $(r_1 \cdots r_n, c_1 \cdots c_n)$, a 0-opening (resp., 1-opening) is computed by setting $s_i = \pi_{r_i}^{-1}(c_i)$ (resp., $s_i = \pi_{r_i}^{-1}(c_i - 1)$) for all i. Note that the receiver's view of the commit phase equals the messages exchanged by the verifier and the first prover, and these were generated in Step 1.

Note that the simulator's messages are distributed identically to the provers' messages in the real interaction. (The only difference is in the way these messages are generated: In the real interaction, the s_i's are selected uniformly in $\{1, 2, 3\}$ and (together with the r_i's and the randomly permuted coloring) determine the c_i's, whereas in the simulation the c_i's are selected uniformly in $\{1, 2, 3\}$ and (together with the r_i's and a random pair in $\{1, 2, 3\}$) determine the revealed s_i's.)

We remark that the entire argument extends easily to the case in which polynomially many instances of the protocol are performed concurrently. Thus, we obtain the following:

Theorem 4.11.8: *Every language in $\mathcal{NP}$ has a perfect zero-knowledge two-prover proof system. Furthermore, this proof system has the following additional properties:*

- *Communication is conducted in a single round: The verifier sends a single message to each of the two provers, which in turn respond with a single message.*
- *The soundness error is exponentially vanishing.*
- *The strategies of the two provers can be implemented by probabilistic polynomial-time machines that get an $\mathcal{NP}$-witness as auxiliary input.*

Efficiency Improvement. A dramatic improvement in the efficiency of two-prover (perfect) zero-knowledge proofs for $\mathcal{NP}$ can be obtained by relying on results regarding probabilistically checkable proofs (PCPs). In particular, such proof systems, with negligible error probability, can be implemented in probabilistic polynomial time, so that the total number of bits exchanged in the interaction is poly-logarithmic.

4.11.4. Applications

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

One natural application is to the problem of identification and specifically the identification of a *user* at some *station*. In Section 4.7 we discuss how to reduce identification to a zero-knowledge proof of knowledge (for some $\mathcal{NP}$-relation). Here we suggest supplying each user with two smart-cards, implementing the two provers in a two-prover zero-knowledge proof of knowledge. These two smart-cards have to be inserted in two different slots of the station, and this should guarantee that the smart-cards cannot communicate with one another. The station will play the role of the verifier in the zero-knowledge proof of knowledge. This way, the station is perfectly protected against impersonation, whereas the users are perfectly protected against pirate stations that may try to extract knowledge from the smart-cards (so as to enable impersonation by their own agents).

4.12. Miscellaneous

4.12.1. Historical Notes

Interactive proof systems were introduced by Goldwasser, Micali, and Rackoff [124].[33] A restricted form of interactive proof, known by the name *Arthur-Merlin game* (or *public-coin* proof), was introduced in [8] and shown in [128] to be equivalent to general interactive proofs. The interactive proof for Graph Non-Isomorphism is due to Goldreich, Micali, and Wigderson [112]. The amazing theorem-proving power of interactive proofs was subsequently demonstrated in [157, 198], showing interactive proofs for co$\mathcal{NP}$ and (more generally) for $\mathcal{PSPACE}$, respectively.

The concept of zero-knowledge was introduced by Goldwasser, Micali, and Rackoff in the very same paper [124]. That paper also contained a perfect zero-knowledge proof for Quadratic Non-Residuosity. The perfect zero-knowledge proof system for Graph Isomorphism is due to Goldreich, Micali, and Wigderson [112].

The zero-knowledge proof systems for all languages in $\mathcal{NP}$, using any (non-uniform secure) commitment scheme, are also due to Goldreich, Micali, and Wigderson [112].[34] (Zero-knowledge proof systems for all languages in $\mathcal{IP}$ have been presented in [136] and [25].)

The cryptographic applications of zero-knowledge proofs were the very motivation for their introduction in [124]. Zero-knowledge proofs were applied to solve cryptographic problems in [81] and [54]. However, many more applications became possible once it was shown how to construct zero-knowledge proof systems for every language in $\mathcal{NP}$. In particular, general methodologies for the construction of cryptographic protocols have appeared in [112, 113].

The construction of commitment schemes based on one-way permutations can be traced to [31]. The construction of commitment scheme based on pseudorandom generators is due to Naor [170].

Credits for the Advanced Sections

Negative Results. The results demonstrating the necessity of randomness and interaction for zero-knowledge proofs are from [115]. The results providing upper bounds on the complexity of languages with almost-perfect zero-knowledge proofs (i.e., Theorem 4.5.8) are from [83] and [2]. The results indicating that one-way functions are necessary for non-trivial zero-knowledge are from [181]. The negative results

[33]Earlier versions of their paper date to early 1983. Yet the paper, having been rejected three times from major conferences, first appeared in public only in 1985, concurrently with the paper of Babai [8].

[34]A weaker result was shown *later* in [41]: It provides an alternative construction of zero-knowledge proof systems for $\mathcal{NP}$, using a *stronger* intractability assumption (specifically, the intractability of the Quadratic Residuosity problem).

concerning parallel composition of zero-knowledge proof systems (i.e., Proposition 4.5.9 and Theorem 4.5.11) are from [106].

Witness Indistinguishability. The notions of witness indistinguishability and witness-hiding, were introduced and developed by Feige and Shamir [78]. Section 4.6 is based on their work.

Proofs of Knowledge. The concept of proofs of knowledge originates from the paper of Goldwasser, Micali, and Rackoff [124]. Early attempts to provide a definition of that concept appear in [75] and [205]; however, those definitions were not fully satisfactory. The issue of defining proofs of knowledge has been extensively investigated by Bellare and Goldreich [17], and we follow their suggestions. The application of zero-knowledge proofs of knowledge to identification schemes was discovered by Feige, Fiat, and Shamir [80, 75]. The Fiat-Shamir identification scheme [80] is based on the zero-knowledge proof for Quadratic Residuosity of Goldwasser, Micali, and Rackoff [124].

Computationally Sound Proof Systems (Arguments). Computationally sound proof systems (i.e., arguments)[35] were introduced by Brassard, Chaum, and Crépeau [40]. Their paper also presents perfect zero-knowledge arguments for $\mathcal{NP}$ based on the intractability of factoring. Naor et al. [171] showed how to construct perfect zero-knowledge arguments for $\mathcal{NP}$ based on any one-way permutation, and Construction 4.8.3 is taken from their paper. The poly-logarithmic-communication argument system for $\mathcal{NP}$ (of Section 4.8.4) is due to Kilian [143].

Constant-Round Zero-Knowledge Protocols. The round-efficient zero-knowledge proof systems for $\mathcal{NP}$, based on any claw-free collection, is taken from [105]. The round-efficient zero-knowledge arguments for $\mathcal{NP}$, based on any one-way function, is due to [77], yet our presentation (which uses some of their ideas) is different. (The alternative construction outlined in Section 4.9.2.3 is much more similar to the construction in [77].)

Non-Interactive Zero-Knowledge Proofs. Non-interactive zero-knowledge proof systems were introduced by Blum, Feldman, and Micali [34]. The constructions presented in Section 4.10 are due to Feige, Lapidot, and Shamir [76]. For further detail on Remark 4.10.6, see [23].

Multi-Prover Zero-Knowledge Proofs. Multi-prover interactive proofs were introduced by Ben-Or, Goldwasser, Kilian, and Wigderson [26]. Their paper also presents a perfect zero-knowledge two-prover proof system for $\mathcal{NP}$. The perfect zero-knowledge two-prover proof for $\mathcal{NP}$ presented in Section 4.11 follows their ideas; however, we explicitly state the properties of the two-sender commitment scheme in use. Consequently, we observe that (sufficiently many) parallel repetitions of this *specific* proof system *will*

[35] Unfortunately, there is some confusion regarding terminology in the literature: In some work (particularly [40]), computationally sound proofs (arguments) are negligently referred to as "interactive proofs."

decrease the error probability to a negligible one.[36] (The efficiency improvement, briefly mentioned at the end of Section 4.11.3, is due to [66].)

We mention that multi-prover interactive proof systems are related to probabilistically checkable proof (PCP) systems. The complexity-theoretic aspects of these proof systems have been the focus of much interest. The interested reader is referred to Sections 2.4 and 2.5.2 of [97] (and to the references therein).

4.12.2. Suggestions for Further Reading

A wider perspective on probabilistic proof systems is offered by Goldreich [97]: In particular, Chapter 2 of [97] contains further details on interactive proof systems, an introduction to probabilistically checkable proof (PCP) systems and discussions of other types of probabilistic proof systems. The exposition focuses on the basic definitions and results concerning such systems and emphasizes both the similarities and differences between the various types of probabilistic proofs. Specifically, like zero-knowledge proof systems, all probabilistic proof systems share a common (untraditional) feature: They carry a probability of error. Yet this probability is explicitly bounded and can be reduced by successive applications of the proof system. The gain in allowing this untraditional relaxation is substantial, as demonstrated by three well-known results regarding *interactive proofs*, *zero-knowledge proofs*, and *probabilistic checkable proofs*: In each of these cases, allowing a bounded probability of error makes the system much more powerful and useful than the traditional (errorless) counterparts.

Since their introduction a decade and a half ago, zero-knowledge proofs have been the focus of much research. We refrain from offering a comprehensive list of suggestions for further reading. Instead, we merely point out some works that address obvious gaps in the current chapter.

- *A uniform-complexity treatment* of zero-knowledge is provided in [94]. In particular, it is shown how to use (uniformly) one-way functions to construct interactive proof systems for $\mathcal{NP}$ such that it is infeasible to find instances in which the prover leaks knowledge.
- *Statistical* (a.k.a *almost-perfect*) *zero-knowledge proofs* offer absolute levels of security for both the prover and the verifier; that is, both the zero-knowledge and soundness conditions are satisfied in a strong probabilistic sense rather than in a computational one. The class of problems possessing statistical zero-knowledge proofs, denoted $\mathcal{SZK}$, is quite intriguing (e.g., it contains some hard problems [124, 109], has complete problems [194, 118, 120], and is closed under complementation [180, 194, 118]). The interested reader is directed to Vadhan's thesis [206].

  We mention that some of the techniques developed toward studying $\mathcal{SZK}$ are also applicable in the context of ordinary (computational) zero-knowledge proofs (e.g., the transformation from public-coin proof systems that are zero-knowledge with respect to an honest verifier to similar systems that are zero-knowledge in general [118]).

[36]This observation escaped the authors of [146], who, being aware of the problematics of parallel repetitions (of general multi-prover systems), suggested an alternative construction.

- In Section 4.5 we discussed *the problematics of parallel repetition* in the context of zero-knowledge. As mentioned there, parallel repetition is also problematic in the context of computationally sound proofs [19] and in the context of multi-prover proofs [74, 190].
- In continuation of Section 4.9, we mention that round-efficient *perfect* zero-knowledge arguments for $\mathcal{NP}$, based on the intractability of the discrete-logarithm problem, have been published [42].
- In continuation of Section 4.10, we mention that a much *more efficient construction of non-interactive proof systems for* $\mathcal{NP}$, based on the same assumptions as [76], has appeared in [144]. Further strengthenings of non-interactive zero-knowledge have been suggested in [193].
- The paper by Goldwasser, Micali, and Rackoff [124] also contains a suggestion for a general measure of "knowledge" revealed by a prover. For further details on this measure, which is called *knowledge complexity*, see [116] (and the references therein). (Indeed, knowledge-complexity zero coincides with zero-knowledge.)

Finally, we mention recent research taking place regarding the preservation of zero-knowledge in settings such as *concurrent* asynchronous executions [68, 189, 60] and *resettable* executions [47]. It would be unwise to attempt to summarize those research efforts at the current stage.

4.12.3. Open Problems

Our formulation of zero-knowledge (e.g., perfect zero-knowledge as defined in Definition 4.3.1) is different from the standard definition used in the literature (e.g., Definition 4.3.6). The standard definition refers to *expected* polynomial-time machines rather than to strictly (probabilistic) polynomial-time machines. Clearly, Definition 4.3.1 implies Definition 4.3.6 (see Exercise 7), but it is unknown whether or not the converse holds. In particular, the known constant-round zero-knowledge protocols for $\mathcal{NP}$ are known to be zero-knowledge only when allowing *expected* polynomial-time simulators. This state of affairs is quite annoying, and resolving it will be of theoretical and practical importance.

Whereas zero-knowledge proofs for $\mathcal{NP}$ can be constructed based on any (non-uniformly) one-way function (which is the most general assumption used in this book), some other results mentioned earlier require stronger assumptions. Specifically, it would be nice to construct *constant-round* zero-knowledge proofs, *perfect* zero-knowledge arguments, and *non-interactive zero-knowledge* proofs for $\mathcal{NP}$ based on weaker assumptions than the ones currently used.

4.12.4. Exercises

The exercises in this first batch are intended for coverage of the basic material (i.e., Sections 4.1–4.4).

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

Guideline: Execute the weaker interactive proof sufficiently many times, using independently chosen coin tosses for each execution, and rule by comparing the number of accepting executions to an appropriate threshold. Observe that the bounds on completeness and soundness need to be efficiently computable. Be careful when demonstrating the soundness of the resulting verifier (i.e., do not assume that the cheating prover executes each copy independently of the other copies). We note that the statement remains valid regardless of whether these repetitions are executed sequentially or "in parallel," but demonstrating that the soundness condition is satisfied is much easier in the sequential case.

Exercise 2: *The role of randomization in interactive proofs, Part 1*: Prove that if L has an interactive proof system in which the verifier is deterministic, then $L \in \mathcal{NP}$.

Guideline: Note that if the verifier is deterministic, then the entire interaction between the prover and the verifier can be determined by the prover.

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

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

Exercise 4: *The role of randomization in interactive proofs, Part 3*: Consider the following (bad) modification to the definition of a pair of linked interactive machines (and interactive proofs). By this modification, also the random tapes of the prover and verifier coincide (i.e., intuitively, both use the same sequence of coin tosses that is known to both of them). We call such proof systems *shared-randomness interactive proofs*. Show that only languages in $\mathcal{MA}$ have a shared-randomness interactive proof system, where a language L is in $\mathcal{MA}$ if there exists a language R_L in $\mathcal{BPP}$ and a polynomial p such that $x \in L$ if and only if there exists $y \in \{0, 1\}^{p(|x|)}$ such that $(x, y) \in R_L$.

Guideline: First convert a shared-randomness interactive proof system into an interactive proof system (of the original kind) in which the verifier reveals all its coin tosses up-front. Next, use reasoning as in Exercise 2.

Show that $\mathcal{MA}$ actually equals the class of languages having shared-randomness interactive proof systems.

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

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

Exercise 6: *Simulator error in perfect zero-knowledge simulators, Part 1*: Consider a modification of Definition 4.3.1 in which condition 1 is replaced by requiring that for some function $\beta(\cdot)$, $\Pr[M^*(x) = \perp] < \beta(|x|)$. Assume that $\beta(\cdot)$ is polynomial time-computable. Show that the following hold:

1. If for some polynomial $p_1(\cdot)$ and all sufficiently large n's, $\beta(n) < 1 - (1/p_1(n))$, then the modified definition is equivalent to the original one.
2. If for some polynomial $p_2(\cdot)$ and all sufficiently large n's, $\beta(n) > 2^{-p_2(n)}$, then the modified definition is equivalent to the original one.

Justify the bounds placed on the function $\beta(\cdot)$.

Guideline: Invoke the simulator sufficiently many times.

Exercise 7: *Simulator error in perfect zero-knowledge simulators, Part 2*: Prove that Definition 4.3.1 implies Definition 4.3.6.

Exercise 8: *Perfect versus almost-perfect zero-knowledge*: Prove that every perfect zero-knowledge system is also almost-perfect zero-knowledge. (That is, prove that Definition 4.3.1 implies Definition 4.3.4.)

Guideline: Using Item 2 of Exercise 6, note that the statistical difference between $M^*(x)$ and $m^*(x)$ (i.e., "$M^*(x)$ conditioned that it not be $\perp$") is negligible.

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

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

Guideline: To show that Definition 4.3.3 implies Definition 4.3.2, observe that the output of every interactive machine can be easily computed from its view of the interaction. To show that Definition 4.3.2 implies Definition 4.3.3, show that for every probabilistic polynomial-time V^* there exists a probabilistic polynomial-time V^{**} such that $\text{view}_{V^*}^{P}(x) = \langle P, V^{**}\rangle(x)$.

Exercise 11: Prove that Definition 4.3.10 is equivalent to a version where the auxiliary input to the verifier is explicitly bounded in length. That is, the alternative zero-knowledge clause reads as follows:

> *for every polynomial* ℓ and for every probabilistic polynomial-time interactive machine V^* there exists a probabilistic polynomial-time algorithm M^* such that the following two ensembles are computationally indistinguishable:
>
> - $\{\langle P(y_x), V^*(z)\rangle(x)\}_{x \in L, z \in \{0,1\}^{\ell(|x|)}}$
> - $\{M^*(x, z)\}_{x \in L, z \in \{0,1\}^{\ell(|x|)}}$
>
> where y_x is as in Definition 4.3.10.

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

Exercise 12: Present a *simple* probabilistic polynomial-time algorithm that simulates the view of the interaction of the verifier described in Construction 4.3.8 with the prover defined there. The simulator, on input $x \in GI$, should have output that is distributed identically to $\text{view}_{V_{GI}}^{P_{GI}}(x)$.

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

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

Exercise 14: Considering the commitment scheme of Construction 4.4.4, suggest a cheating sender that induces a receiver's view (of the commit phase) that is unlikely to have any possible opening and still is computationally indistinguishable from the receiver's view in interactions with the prescribed sender. That is, present a probabilistic polynomial-time interactive machine S^* such that the following two conditions hold:

1. With overwhelmingly high probability, $\langle S^*(0), R\rangle(1^n)$ is neither a possible 0-commitment nor a possible 1-commitment.
2. The ensembles $\langle S^*(0), R\rangle(1^n)$ and $\langle S(0), R\rangle(1^n)$ are computationally indistinguishable.

Guideline: The sender simply replies with a uniformly chosen string.

Exercise 15: *Using Construction 4.4.4 as a commitment scheme in Construction 4.4.7*: Prove that when the commitment scheme of Construction 4.4.4 is used in the $G3C$ protocol, then the resulting scheme remains zero-knowledge. Consider the modifications required to prove Claim 4.4.8.2.

Exercise 16: *Strong reductions*: Let L_1 and L_2 be two languages in $\mathcal{NP}$, and let R_1 and R_2 be binary relations characterizing L_1 and L_2, respectively. We say that the relation R_1 is *Levin-reducible*[37] to the relation R_2 if there exist two polynomial-time-computable functions f and g such that the following two conditions hold:

Standard requirement: $x \in L_1$ if and only if $f(x) \in L_2$.

Additional requirement: For every $(x, w) \in R_1$, it holds that $(f(x), g(x, w)) \in R_2$.

Prove the following statements:

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

[37]We name this reduction after Levin because it was he who, upon discovering (independently of Cook and Karp) the existence of $\mathcal{NP}$-complete problems, used a stronger definition of a reduction that implies the one here. We assume that the reader is familiar with standard reductions among languages such as Bounded Halting, SAT, and 3SAT (as in [86]).

Exercise 17: Prove the existence of a Karp reduction of any $\mathcal{NP}$ language L to SAT that when considered as a function can be inverted in polynomial time. Same for the reduction of SAT to $3SAT$ and the reduction of $3SAT$ to $G3C$. (In fact, the standard Karp reductions have this property.)

Exercise 18: *Applications of Theorem 4.4.11*: This exercise assumes a basic familiarity with the notions of a public-key encryption scheme and a signature scheme. Assuming the existence of non-uniformly one-way functions, present solutions to the following cryptographic problems:

1. Suppose that party S sends, over a public channel, encrypted data to several parties, $R_1, \ldots, R_t$. Specifically, the data sent to R_i are encrypted using the public encryption key of party R_i. We assume that all parties have access to the ciphertexts sent over the public channel. Suppose that S wants to prove to some other party that it has sent the same data to all R_i's, but it wants to do so without revealing the data.
2. Referring to the same communication setting, consider a party R that has received data encrypted using its own public encryption key. Suppose that these data consist of two parts, and party R wishes to reveal to someone the first part of the data but not the second. Further suppose that the other party wants a proof that R has indeed revealed the correct content of the first part of the data.
3. Suppose that party S wishes to send party R a signature to a publicly known document such that only R receives the signature, but everyone else can verify that such a signature was indeed sent by S. (We assume, again, that all parties share a public channel.)

Exercise 19: *On knowledge tightness*: Prove that the protocol resulting from executing Construction 4.4.7 for $k(n) = O(\log n)$ times in parallel is zero-knowledge. Furthermore, prove that it has knowledge tightness $(3/2)^{k(n)}$ (approximately).

Exercise 20: *More efficient zero-knowledge proofs for $\mathcal{NP}$*: Consider the basic proof system for the Hamiltonian-cycle problem (HC) presented in Construction 4.7.14.

1. Evaluate its acceptance probabilities (i.e., completeness and soundness bounds).
2. Provide a sketch of the proof of the zero-knowledge property (i.e., describe the simulator). Specifically, present a simulator that establishes knowledge tightness of approximately 2. If you are really serious, provide a full proof of the zero-knowledge property.

Exercises for the Advanced Sections. The rest of the exercises refer to the material in the advanced sections (i.e., Sections 4.5–4.11).

Exercise 21: *An alternative formulation of black-box zero-knowledge*: Here we say that a probabilistic polynomial-time oracle machine M is a **black-box simulator** *for the prover P and the language L* if for every (not necessarily uniform) polynomial-size circuit family $\{B_n\}_{n\in\mathbb{N}}$, the ensembles $\{\langle P, B_{|x|}\rangle(x)\}_{x\in L}$ and $\{M^{B_{|x|}}(x)\}_{x\in L}$ are indistinguishable by (non-uniform) polynomial-size circuits. Namely, for every polynomial-size circuit family $\{D_n\}_{n\in\mathbb{N}}$, every polynomial $p(\cdot)$, all sufficiently large n, and $x \in \{0,1\}^n \cap L$,

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

Prove that the current formulation is equivalent to the one presented in Definition 4.5.10.

Exercise 22: Prove that the protocol presented in Construction 4.4.7 is indeed a black-box zero-knowledge proof system for $G3C$.

Guideline: Use the formulation presented in Exercise 21.

Exercise 23: Prove that black-box zero-knowledge is preserved under sequential composition. (Note that this does not follow merely from the fact that auxiliary-input zero-knowledge is preserved under sequential composition.)

Guideline: Adapt the proof of Lemma 4.3.11.

Exercise 24: *Refuting another parallel-composition conjecture*: Prove that there exists a zero-knowledge prover P such that the prover resulting from running two copies of P in parallel yields knowledge (e.g., a cheating verifier can extract from this prover a solution to a problem that is not solvable in polynomial time).

Guideline: Let P_1 and P_2 be as in Proposition 4.5.9, and consider the prover P that randomly selects which of the two programs to execute. Alternatively, the choice can be determined by the verifier.

Exercise 25: Assuming that one-way permutations exist, present a witness-indistinguishable proof system (with a probabilistic polynomial-time prover) that is NOT strongly witness-indistinguishable.

Guideline: Consider a one-way permutation f, a hard-core predicate b of f, and the witness relation $\{(f(w),w): w \in \{0,1\}^*\}$. Consider a prover that on input $f(w)$ (and auxiliary input w) sends w to the verifier, and consider the ensembles $\{X_n^0\}_{n\in\mathbb{N}}$ and $\{X_n^1\}_{n\in\mathbb{N}}$, where X_n^i is uniform on $\{f(w): w \in \{0,1\}^n \,\&\, b(w) = i\}$.

Exercise 26: *Some basic zero-knowledge proofs of knowledge*:

1. Show that Construction 4.3.8 is a proof of knowledge of an isomorphism with knowledge error $\frac{1}{2}$.
2. Show that Construction 4.4.7 (when applied on common input $G = (V, E)$) is a proof of knowledge of a 3-coloring with knowledge error $1 - \frac{1}{|E|}$.

See also Part 1 of Exercise 28.

Guideline: Observe that in these cases, if the verifier accepts with probability greater than the knowledge error, then it accepts with probability 1. Also observe that the number of possible verifier messages in these proof systems is polynomial in the common input. Thus, the extractor can emulate executions of these systems with all possible verifier messages.

Exercise 27: *Parallel repetitions of some basic proofs of knowledge*: Let $k : \mathbb{N} \to \mathbb{N}$ be polynomially bounded. Consider the proof systems resulting by executing each of the basic systems mentioned in Exercise 26 for k times in parallel.

1. Show that the k parallel execution of Construction 4.3.8 constitutes a proof of knowledge of an isomorphism with knowledge error $2^{-k(\cdot)}$. (Analogously for Construction 4.7.12.)
2. Show that the k parallel execution of Construction 4.4.7 provides a proof of knowledge of a 3-coloring with knowledge error $(1 - (1/|E|))^{-k(|G|)}$.

Note that we make no claim regarding zero-knowledge.

See also Part 2 of Exercise 28.

Guideline: For Part 1, note that any two different transcripts in which the verifier accepts will yield an isomorphism. In Part 2 this simple observation fails. Still, observe that $|E|$ accepting transcripts that differ in any fixed copy of the basic system do yield a 3-coloring.

Exercise 28: *More efficient zero-knowledge proofs of knowledge for $\mathcal{NP}$*: As in Exercise 20, consider the basic proof system for the Hamiltonian-cycle problem (HC) presented in Construction 4.7.14.

1. Prove that the basic proof system is a proof of knowledge of a Hamiltonian cycle with knowledge error $\frac{1}{2}$.
2. Prove that the proof system that results from iterating the basic system k times is a proof of knowledge of a Hamiltonian cycle with knowledge error 2^{-k}. Consider *both* sequential and parallel repetitions.

Exercise 29: *More on the equivalence of Definitions 4.7.2 and 4.7.3*: Suppose that R is polynomially bounded and that the extractor in Definition 4.7.3 outputs either a valid solution or a special failure symbol. Referring to this relation R, show that V satisfies the validity-with-error κ condition of Definition 4.7.2 if and only if V satisfies the alternative validity-with-error κ condition of (the modified) Definition 4.7.3.

Guideline: Follow the outline of the proof of Proposition 4.7.4, noting that all references to the hypothesis that R is an $\mathcal{NP}$-relation can be replaced by the hypothesis that the extractor in Definition 4.7.3 outputs either a valid solution or a special failure symbol. In particular, in the second direction, omit the exhaustive search that takes place with probability $2^{-\text{poly}(|x|)}$, and use the fact that $p(x, y, r) > \kappa(|x|)$ implies $p(x, y, r) \geq \kappa(|x|) + 2^{-\text{poly}(|x|)}$.

Exercise 30: *Zero-knowledge strong proofs of knowledge for $\mathcal{NP}$*: Consider again the basic proof system for the Hamiltonian-cycle problem (HC) presented in Construction 4.7.14. Prove that the proof system that results from sequentially iterating the basic system sufficiently many times is a strong proof of knowledge of a Hamiltonian cycle. (Recall that it is indeed zero-knowledge.)

Exercise 31: *Error reduction in computationally sound proofs*: Given a computationally sound proof (with error probability $\frac{1}{3}$) for a language L, construct a computationally sound proof with negligible error probability (for L).

Guideline: Use sequential repetitions. In fact, the error probability can be made exponentially vanishing. Parallel repetitions may fail to reduce computational soundness in some cases (see [19]).

Exercise 32: *Commitment schemes, an impossibility result*: Prove that there exists no two-party protocol that simultaneously satisfies the perfect secrecy requirement of Definition 4.8.2 and the (information-theoretic) unambiguity requirement of Definition 4.4.1.

Exercise 33: *Failure of ordinary hashing in Construction 4.8.3*: Show that in Construction 4.8.3, replacing the iterative hashing by an ordinary one results in a scheme that is NOT binding (not even in a computational sense). That is, using the notation of

Construction 4.8.3, consider replacement of the iterative hashing step with the following step (where b and the r^i's are as in Construction 4.8.3):

- (Ordinary hashing): The receiver sends the message $(r^1, \dots, r^{n-1})$ to the sender, which replies with the message $(c^1, \dots, c^{n-1})$, where $c^i \stackrel{\text{def}}{=} b(y, r^i)$, for $i = 1, \dots, n-1$.

  That is, the prescribed sender computes the c^i's as in Construction 4.8.3, but a cheating sender can determine all c^i's based on all r^i's (rather that determine each c^i based only on $(r^1, \dots, r^i)$).

Present an efficient strategy that allows the sender to violate the unambiguity condition.

Guideline: Given any one-way permutation f', first construct a one-way permutation f satisfying $f(0^{|x'|}, x') = (0^{|x'|}, x')$ and $f(x', 0^{|x'|}) = (x', 0^{|x'|})$ for every x'. (Hint: First obtain a one-way permutation f'' that satisfies $f''(0^n) = 0^n$ for all n's,[38] and then let $f(0^{|x''|}, x'') \stackrel{\text{def}}{=} (0^{|x''|}, x'')$, $f(x', 0^{|x'|}) \stackrel{\text{def}}{=} (x', 0^{|x'|})$, and $f(x', x'') \stackrel{\text{def}}{=} (f''(x'), f''(x''))$ for $x', x'' \in \{0,1\}^{|x'|} \setminus \{0\}^{|x'|}$.)

Assuming that the modified protocol is executed with f as constructed here, consider a cheating sender that upon receiving the message $(r^1, \dots, r^{n-1})$ finds $y^1 \in \{0,1\}^{n/2}\{0\}^{n/2}$, $y^2 \in \{0\}^{n/2}\{0,1\}^{n/2}$, and $\overline{c} = (c^1, \dots, c^{n-1})$ such that the following conditions hold:

1. $c^i = b(y^j, r^i)$ for $i = 1, \dots, n-1$ and $j = 1, 2$

2. $b(y^j, r^n) \equiv j \pmod 2$ for $j = 1, 2$

(where r^n is the unique vector independent of $r^1, \dots, r^{n-1}$).

Note that f is invariant under such y^j's, and thus they can serve as valid decommitments.

Finally, prove that such a solution $y^1, y^2, \overline{c}$ always exists and can be found by solving a linear system. (Hint: Consider the linear system $b(x^1 0^{n/2}, r^i) = b(0^{n/2} x^2, r^i)$ for $i = 1, \dots, n-1$ and $b(x^1 0^{n/2}, r^n) \equiv b(0^{n/2} x^2, r^n) + 1 \pmod 2$. Extra hint: Things may become more clear when writing the conditions in matrix form.)

Exercise 34: *Non-interactive zero-knowledge, bounded versus unbounded*: Show that Construction 4.10.4 is not unboundedly zero-knowledge unless $\mathcal{NP} \subseteq \mathcal{BPP}$.

Guideline: Consider invoking this proof system twice: first on a graph consisting of a simple cycle and then on a graph for which a Hamiltonian cycle is to be found.

Exercise 35: Regarding the definition of a two-sender commitment scheme (Definition 4.11.4), show that for every p there exist senders' strategies such that each resulting receiver view can be 0-opened with probability p and 1-opened with probability $1 - p$.

Guideline: Use the perfect-secrecy requirement and the fact that you can present computationally unbounded senders' strategies.

[38] See Exercise 13 in Chapter 2.

APPENDIX A

Background in Computational Number Theory

The material presented in this appendix is merely the minimum needed for the few examples of specific constructions presented in this book. What we cover here are a few structural and algorithmic facts concerning prime and composite numbers. For a more comprehensive treatment, consult any standard textbook (e.g., [10]).

A.1. Prime Numbers

A *prime* is a natural number that is not divisible by any natural number other than itself and 1. For simplicity, say that 1 is NOT a prime.

For a prime P, the *additive group modulo* P, denoted $\mathbb{Z}_P$, consists of the set $\{0, \ldots, P-1\}$ and the operation of *addition* mod P. All elements except the identity (i.e., 0) have order P(in this group). The *multiplicative group modulo* P, denoted $\mathbb{Z}_P^*$, consists of the set $\{1, \ldots, P-1\}$ and the operation of *multiplication* mod P. This group is cyclic too. In fact, at least $1/\log_2 P$ of the elements of the group have order $P-1$ and are called *primitive*.[1]

A.1.1. Quadratic Residues Modulo a Prime

A *quadratic residue modulo a prime* P is an integer s such that there exists an $r \in \mathbb{Z}_P^*$ satisfying $s \equiv r^2 \pmod{P}$. Thus, in particular, s has to be relatively prime to P. Clearly, if r is a square root of s modulo P, then so is $-r$ (since $(-r)^2 \equiv r^2$). Furthermore, if $x^2 \equiv s \pmod{P}$ has a solution modulo P, then it has exactly two such solutions (as otherwise $r_1 \not\equiv \pm r_2 \pmod{P}$ are both solutions, and $0 \equiv r_1^2 - r_2^2 \equiv (r_1 - r_2)(r_1 + r_2) \pmod{P}$ follows, in contradiction to the primality of P).

The quadratic residues modulo P form a subgroup of the multiplicative group modulo P. The former subgroup contains exactly half of the members of the group.

[1]The exact number of primitive elements modulo P depends on the prime factorization of $P - 1 = \prod_{i=1}^{t} P_i^{e_i}$ (see Section A.2): It equals $\prod_{i=1}^{t}((P_i - 1) \cdot P_i^{e_i - 1})$.

Furthermore, squaring modulo P is a 2-to-1 mapping of the group to the subgroup. In case $P \equiv 3 \pmod 4$, each image of this mapping has one pre-image in the subgroup (i.e., a quadratic residue) and one pre-image that is not in the subgroup (i.e., a non-quadratic residue).[2]

A.1.2. Extracting Square Roots Modulo a Prime

In general, extracting square roots module a prime can be done by using Berlekamp's algorithm [28]. The latter is a randomized algorithm for factoring polynomials modulo a prime. (Note that extracting a square root of s modulo a prime P amounts to solving the equation $x^2 \equiv s \pmod P$, which can be cast as the problem of factoring the polynomial $x^2 - s$ modulo P.)

A more direct approach is possible in the special case in which the prime is congruent to $3 \pmod 4$, which is the case in most cryptographic applications. In this case we observe that for a quadratic residue $s \equiv x^2 \pmod P$, we have

$$\begin{aligned} s^{(P+1)/4} &\equiv x^{(P+1)/2} \pmod P \\ &\equiv x^{(P-1)/2} \cdot x \pmod P \\ &\equiv \pm x \pmod P \end{aligned}$$

where in the last equality we use Fermat's little theorem, by which $x^{(P-1)/2} \equiv \pm 1 \pmod P$ for every integer x and prime P. Thus, in this special case, we obtain a square root of s modulo P by raising s to the power $\frac{P+1}{4}$ modulo P. (Note that this square root is a quadratic residue modulo P.)

A.1.3. Primality Testers

The common approach to testing whether or not an integer is a prime is to utilize Rabin's randomized primality tester [185], which is related to a deterministic algorithm due to Miller [166].[3] The alternative of using a somewhat different randomized algorithm, discovered independently by Solovay and Strassen [202], seems less popular. Here we present a third alternative, which seems less well known (and was discovered independently by several researchers, one of them being Manuel Blum). The only number-theoretic facts that we use are as follows:

1. For every *prime* $P > 2$, each quadratic residue mod P has exactly two square roots mod P (and they sum up to P).
2. For every odd and non-integer-power *composite* number N, each quadratic residue mod N has at least four square roots mod N.

[2] This follows from the fact that -1 is a non-quadratic residue modulo such primes. In contrast, in case $P \equiv 1 \pmod 4$, it holds that -1 is a quadratic residue modulo P. Thus, in case $P \equiv 1 \pmod 4$, for each quadratic residue the two square roots either are both quadratic residues or are both non-quadratic residues.

[3] Miller's algorithm relies on the Extended Riemann Hypothesis (ERH).

Our algorithm uses as a black box an algorithm, denoted SQRT, that given a prime P and a quadratic residue s mod P, returns a square root of s mod P. There is no guarantee as to what the algorithm does in case the input is not of this form (and, in particular, in case P is not a prime).

Algorithm. On input a natural number $N > 2$, do the following:

1. If N is either even or an integer-power, then `reject`.

2. Uniformly select $r \in \{1, \ldots, N-1\}$ and set $s \leftarrow r^2 \bmod N$.

3. Let $r' \leftarrow \texttt{SQRT}(N, s)$. If $r' \equiv \pm r \pmod N$, then `accept`, else `reject`.

Analysis. By Fact 1, on input a prime number N, the algorithm always accepts (since in this case $\texttt{SQRT}(N, r^2 \bmod N) = \pm r$ for any $r \in \{1, \ldots, N-1\}$). On the other hand, suppose that N is an odd composite that is not an integer-power. Then, by Fact 2, each quadratic residue s has at least four square roots, and each is equally likely to be chosen at Step 2 (since s yields no information on the specific r). Thus, for every such s, the probability that $\pm\texttt{SQRT}(N, s)$ is chosen in Step 2 is at most $\frac{2}{4}$. It follows that on input a composite number, the algorithm rejects with probability at least $\frac{1}{2}$.

Comment. The analysis presupposes that the algorithm SQRT is always correct when fed with a pair (P, s), where P is prime and s is a quadratic residue mod P. Such an algorithm was described for the special case where $P \equiv 3 \pmod 4$. Thus, whenever the candidate number is congruent to $3 \pmod 4$, which typically is the case in our applications, this description suffices. For the case $P \equiv 1 \pmod 4$, we employ the randomized modular square-root-extraction algorithm mentioned earlier and observe that in case SQRT has error probability $\varepsilon < \frac{1}{2}$, our algorithm still distinguishes primes from composites (since on the former it accepts with probability at least $1 - \varepsilon > \frac{1}{2}$, whereas on the latter it accepts with probability at most $\frac{1}{2}$). The statistical difference between the two cases can be amplified by invoking the algorithm several times.

We mention that error-free probabilistic polynomial-time algorithms for testing primality do exist [121, 1], but currently are much slower. (These algorithms output either the correct answer or a special `don't know` symbol, where the latter is output with probability at most $\frac{1}{2}$.)

A.1.4. On Uniform Selection of Primes

A simple method for uniformly generating a prime number in some interval, say between N and $2N$, consists of repeatedly selecting at random an integer in this interval and testing it for primality. The question, of course, is, *How many times do we need to repeat the procedure before a prime number is found?* This question is intimately related to the *density of primes*, which has been extensively studied in number theory [7]. For our purposes it suffices to assert that in case the sampling interval is sufficiently large (when compared with the size of the integers in it), then the density of primes in it is noticeable (i.e., is a polynomial fraction). Specifically, the density of primes in the interval $[N, 2N]$ is $\Theta(1/\log N)$. Hence, on input N, we can expect to hit a prime in the interval $[N, 2N]$ within $\Theta(\log N)$ trials. Furthermore, with probability at least $1-(1/N)^2$ we will hit a prime before conducting $\Theta((\log N)^2)$ trials. Hence, for all practical purposes, we can confine ourselves to conducting a number of trials that is polynomial (i.e., n^2) in the length of the prime we want to generate (i.e., $n = \log_2 N$). (We comment that an analogous discussion applies for primes that are congruent to 3 mod 4.)

We remark that there exists a probabilistic polynomial-time algorithm [9] that produces a uniformly selected prime P together with the factorization of $P-1$. The prime factorization of $P-1$ can be used to verify that a given residue is a generator of the multiplicative group modulo P: If $g^{P-1} \equiv 1 \pmod{P}$ and $g^N \not\equiv 1 \pmod{P}$ for every N that divides $P-1$, then g is a generator of the multiplicative group modulo P. (Note that it suffices to check that $g^{P-1} \equiv 1 \pmod{P}$ and $g^{(P-1)/Q} \not\equiv 1 \pmod{P}$ for every prime Q that divides $P-1$.) We mention that a noticeable fraction of the residues modulo P will be generators of the multiplicative group modulo P.

Finally, we comment that more randomness-efficient procedures for generating an n-bit-long prime do exist and utilize only $O(n)$ random bits.[4]

A.2. Composite Numbers

A natural number (other than 1) that is not a prime is called a *composite*. Such a number N is uniquely represented as a product of prime powers; that is, $N = \prod_{i=1}^{t} P_i^{e_i}$, where the P_i's are distinct primes, the e_i's are natural numbers, and either $t > 1$ or $e_1 > 1$. These P_i's are called the *prime factorization of* N. It is widely believed that given a composite number, it is infeasible to find its prime factorization. Specifically, it is assumed that it is infeasible to find the factorization of a composite number that is the product of two random primes. That is, it is assumed that any probabilistic polynomial-time algorithm, given the product of two uniformly chosen n-bit-long primes, can successfully recover these primes only with negligible probability. Rivest, Shamir, and Adleman [191] have suggested the use of this assumption for the construction of cryptographic schemes. Indeed, they have done so in proposing the RSA function, and their suggestion has turned out to have a vast impact (i.e., being the most popular intractability assumption in use in cryptography).

For a composite N, the *additive group modulo* N, denoted $\mathbb{Z}_N$, consists of the set $\{0, \ldots, N-1\}$ and the operation of *addition* mod N. All elements that are relatively prime to N have order N (in this group). The *multiplicative group modulo* N, denoted $\mathbb{Z}_N^*$, consists of the set of natural numbers that are smaller than $N$and relatively prime to it, and the operation is *multiplication* mod N.

[4]For example, one can use a generic transformation of [177]. Loosely speaking, the latter transformation takes any polynomial-time linear-space randomized algorithm and returns a similar algorithm that has linear randomness complexity. Note that the selection process described in the preceding text satisfies the premise of the transformation.

A.2.1. Quadratic Residues Modulo a Composite

For simplicity, we focus on odd composite numbers that are not divisible by any strict prime power; that is, we consider numbers of the form $\prod_{i=1}^{t} P_i$, where the P_i's are *distinct odd* primes and $t > 1$.

Let $N = \prod_{i=1}^{t} P_i$ be such a composite number. A *quadratic residue modulo* N is an integer s such that there exists an $r \in \mathbb{Z}_N^*$ satisfying $s \equiv r^2 \pmod{N}$. Using the Chinese Remainder Theorem, one can show that s is a quadratic residue modulo N if and only if it is a quadratic residue modulo each of the P_i's. Suppose that s is a quadratic residue modulo N. Then the equation $x^2 \equiv s \pmod{N}$ has 2^t distinct (integer) solutions modulo N. Again, this can be proved by invoking the Chinese Remainder Theorem: First observe that the system

$$x^2 \equiv s \pmod{P_i} \quad \text{for } i = 1, \ldots, t \tag{A.1}$$

has a solution. Next note that each single equation has two distinct solutions $\pm r_i \pmod{P_i}$, and finally note that each of the 2^t different combinations yields a distinct-solution to Eq. (A.1) modulo N (i.e., a distinct square root of s modulo N).

The quadratic residues modulo N form a subgroup of the multiplicative group modulo N. The subgroup contains exactly a 2^{-t} fraction of the members of the group. Furthermore, for $N = \prod_{i=1}^{t} P_i$ (as before), squaring modulo N is a 2^t-to-1 mapping of the group to the subgroup. For further discussion of this mapping, in the special case where $t = 2$ and $P_1 \equiv P_2 \equiv 3 \pmod{4}$, see Section A.2.4.

A.2.2. Extracting Square Roots Modulo a Composite

By the preceding discussion (and the effectiveness of the Chinese Remainder Theorem),[5] it follows that given the prime factorization of N, one can efficiently extract square roots modulo N. On the other hand, any algorithm that extracts square roots modulo a composite can be transformed into a factoring algorithm [187]: It suffices to show how an algorithm for extraction of square roots (modulo a composite N) can be used to produce non-trivial divisors of N. The argument is very similar to the one employed in Section A.1.3, the difference being that there the root-extraction algorithm was assumed to work only for extracting square roots modulo a prime (and such efficient algorithms do exist), whereas here we assume that the algorithm works for extracting square roots modulo composites (and such efficient algorithms are assumed not to exist).

Reduction of Factoring to Extracting Modular Square Roots. On input a composite number N, do the following:

1. Uniformly select $r \in \{1, \ldots, N-1\}$.

2. Compute $g \leftarrow \texttt{GCD}(N, r)$. If $g > 1$, then *output* g and halt.[6]

[5]Specifically, the system $x \equiv a_i \pmod{P_i}$ for $i = 1, \ldots, t$ is solved by $\sum_{i=1}^{t} c_i \cdot a_i \bmod \prod_{i=1}^{t} P_i$, where $c_i \overset{\text{def}}{=} Q_i \cdot (Q_i^{-1} \bmod P_i)$ and $Q_i \overset{\text{def}}{=} \prod_{j \neq i} P_j$.

[6]This step takes place in order to allow us to invoke the root-extraction algorithm only on relatively prime pairs (s, N).

3. Set $s \leftarrow r^2 \bmod N$ and invoke the root-extraction algorithm to obtain r' such that $(r')^2 \equiv s \pmod{N}$.

4. Compute $g \leftarrow \texttt{GCD}(N, r - r')$. If $g > 1$, then *output* g and halt.

In case the algorithm halts with some output, the output is a non-trivial divisors of N. The prime factorization of N can be obtained by invoking the algorithm recursively on each of the two non-trivial divisors of N.

Analysis. We can assume that r selected in Step 1 is relatively prime to N, or else the GCD of r and N yields the desired divisor. Invoking the root-extraction algorithm, we obtain r' such that $(r')^2 \equiv s \equiv r^2 \pmod{N}$. Because the root-extraction algorithm has no information on r (beyond $r^2 \pmod{N}$) with probability $2/2^t$, we have $r' \equiv \pm r \pmod{N}$. Otherwise, $r' \not\equiv \pm r \pmod{N}$, and still $0 \equiv (r - r')(r + r') \pmod{N}$. Therefore, $r - r'$ (resp., $r + r'$) has a non-trivial GCD with N, which is found in Step 4. Thus, with probability at least $\frac{1}{2}$, we obtain a non-trivial divisor of N.

A.2.3. The Legendre and Jacobi Symbols

The *Legendre symbol* of integer r modulo a prime P, denoted $\texttt{LS}_P(r)$, is defined as 0 if P divides r, as $+1$ in case r is a quadratic residue modulo P, and as -1 otherwise. Thus, for r that is relatively prime to P, the Legendre symbol of r modulo P indicates whether or not r is a quadratic residue.

The Jacobi symbol of residues modulo a composite N is defined based on the prime factorization of N. Let $\prod_{i=1}^{t} P_i^{e_i}$ denote the prime factorization of N. Then the *Jacobi symbol* of r modulo N, denoted $\texttt{JS}_N(r)$, is defined as $\prod_{i=1}^{t} \texttt{LS}_{P_i}(r)^{e_i}$. Although the Jacobi symbol (of r modulo N) is defined in terms of the prime factorization of the modulus, the Jacobi symbol can be computed efficiently *without knowledge of the factorization of the modulus*. That is, there exists a polynomial-time algorithm that given a pair (r, N) computes $\texttt{JS}_N(r)$. The algorithm proceeds in "GCD-like" manner[7] and utilizes the following facts regarding the Jacobi symbol:

1. $\texttt{JS}_N(r) = \texttt{JS}_N(r \bmod N)$
2. $\texttt{JS}_N(a \cdot b) = \texttt{JS}_N(a) \cdot \texttt{JS}_N(b)$, and $\texttt{JS}_N(1) = 1$
3. $\texttt{JS}_N(2) = (-1)^{(N^2-1)/8}$ (i.e., $\texttt{JS}_N(2) = -1$ iff $N \equiv 4 \pm 1 \pmod{8}$)
4. $\texttt{JS}_N(r) = (-1)^{(N-1)(r-1)/4} \cdot \texttt{JS}_r(N)$ for odd integers N and r

Note that a quadratic residue modulo N must have Jacobi symbol 1, but not all residues of Jacobi symbol 1 are quadratic residues modulo N. (In fact, for $N = \prod_{i=1}^{t} P_i$, as in Section A.2.1, half of the residues with non-zero Jacobi symbols have Jacobi symbol 1, but only a 2^{-t} fraction of these residues are squares modulo N.)[8] The fact that

[7] E.g., $\texttt{JS}_{21}(10) = \texttt{JS}_{21}(2) \cdot \texttt{JS}_{21}(5) = (-1)^{55} \cdot (-1)^{20} \cdot \texttt{JS}_5(21) = -\texttt{JS}_5(1) = -1$. In general, Fact 2 is used only with $a = 2$ (i.e., $\texttt{JS}_N(2 \cdot r) = \texttt{JS}_N(2) \cdot \texttt{JS}_N(r)$). Also, at the very beginning, one can use $\texttt{JS}_{2N}(r) = \texttt{JS}_2(r) \cdot \texttt{JS}_N(r) = (r \bmod 2) \cdot \texttt{JS}_N(r)$.

[8] The elements of $\mathbb{Z}_N^*$ having Jacobi symbol 1 form a subgroup of $\mathbb{Z}_N^*$. This subgroup contains exactly half of the members of the group.

the Jacobi symbol can be computed efficiently (without knowledge of the factorization of the modulus) does *not* seem to yield an *efficient* algorithm for determining whether or not a given residue is a square modulo a given composite (of unknown factorization). In fact, it is believed that determining whether or not a given integer is a quadratic residue modulo a given composite (of unknown factorization) is infeasible. Goldwasser and Micali [123] have suggested use of the conjectured intractability of this problem toward the construction of cryptographic schemes, and that suggestion has been followed in numerous works.

A.2.4. Blum Integers and Their Quadratic-Residue Structure

We call $N = P \cdot Q$, where P and Q are primes, a *Blum integer* if $P \equiv Q \equiv 3 \pmod 4$. For such P (resp., Q), the integer -1 is not a quadratic residue mod P (resp., mod Q), and it follows that -1 is not a quadratic residue modulo N and that -1 has Jacobi symbol 1 mod N.

By earlier discussion, each quadratic residue s modulo N has four square roots, denoted $\pm x$ and $\pm y$, so that $\texttt{GCD}(N, x \pm y) \in \{P, Q\}$. The important fact about Blum Integers is that exactly one of these square roots is a quadratic residue itself.[9] Consequently, $x \mapsto x^2 \bmod N$ induces a *permutation* on the set of quadratic residues modulo N.

(We comment that some sources use a more general definition of Blum integers, but the preceding special case suffices for our purposes. The term "Blum integers" is commonly used in honor of Manuel Blum, who advocated the use of squaring modulo such numbers as a one-way *permutation*.)

We mention that in case $P \not\equiv Q \pmod 8$, the Jacobi symbol of 4 modulo $N = P \cdot Q$ is -1. In this case, obtaining a square root of 4 mod N that is a quadratic residue itself allows us to factor N (since such a residue r satisfies $r \not\equiv \pm 2 \pmod N$ and $(r-2)\cdot(r+2) \equiv 0 \pmod N$).

[9] Let a and b be such that $a^2 \equiv s \pmod P$ and $b^2 \equiv s \pmod Q$. Then, either a or $-a$ (but not both) is a quadratic residue mod P, and similarly for b. Suppose, without loss of generality, that a (resp., b) is a quadratic residue mod P (resp., mod Q). The x satisfying $x \equiv a \pmod P$ and $x \equiv b \pmod Q$ is a square root of s modulo N that is a quadratic residue itself. The other square roots of s modulo N (i.e., $-x$ and $\pm y$, such that $y \equiv a \pmod P$ and $y \equiv -b \pmod Q$) are not quadratic residues mod N.

APPENDIX B

Brief Outline of Volume 2

This first volume contains only material on the *basic tools* of modern cryptography, that is, one-way functions, pseudorandomness, and zero-knowledge proofs. These basic tools are used in the construction of the *basic applications* (to be covered in the second volume). The latter will cover encryption, signatures, and general cryptographic protocols. In this appendix we provide brief summaries of the treatments of these basic applications.

B.1. Encryption: Brief Summary

Both private-key and public-key encryption schemes consist of three efficient algorithms: *key generation*, *encryption*, and *decryption*. The difference between the two types of schemes is reflected in the definition of security: The security of a public-key encryption scheme should also hold when the adversary is given the encryption key, whereas that is not required for private-key encryption schemes. Thus, public-key encryption schemes allow each user to broadcast its encryption key, so that any other user can send it encrypted messages (without needing to first agree on a private encryption key with the receiver). Next we present definitions of security for private-key encryption schemes. The public-key analogies can be easily derived by considering adversaries that get the encryption key as additional input. (For private-key encryption schemes, we can assume, without loss of generality, that the encryption key is identical to the decryption key.)

B.1.1. Definitions

For simplicity, we consider only the encryption of a single message; however, this message can be longer than the key (which rules out information-theoretic secrecy [200]). We present two equivalent definitions of security. The first, called *semantic security*, is a computational analogue of Shannon's definition of *perfect secrecy* [200]. The second definition views secure encryption schemes as those for which it is infeasible to distinguish encryptions of any (known) pair of messages (e.g., the all-zeros message and the all-ones message). The latter definition is technical in nature and is referred to as *indistinguishability of encryptions*.

We stress that the definitions presented here go way beyond saying that it is infeasible to recover the plaintext from the ciphertext. The latter statement is indeed a minimal requirement for a secure encryption scheme, but we claim that it is far too weak a requirement: An encryption scheme typically is used in applications where obtaining specific partial information on the plaintext endangers the security of the application. When designing an application-independent encryption scheme, we do not know which partial information endangers the application and which does not. Furthermore, even if one wants to design an encryption scheme tailored to one's own specific applications, it is rare (to say the least) that one has a precise characterization of all possible partial information that can endanger these applications. Thus, we require that it be *infeasible* to obtain any information about the plaintext from the ciphertext. Furthermore, in most applications the plaintext may not be uniformly distributed, and some a priori information regarding it is available to the adversary. We require that the secrecy of all partial information also be preserved in such a case. That is, even in the presence of a priori information on the plaintext, it is *infeasible* to obtain any (new) information about the plaintext from the ciphertext (beyond what it is feasible to obtain from the a priori information on the plaintext). The definition of semantic security postulates all of this. The equivalent definition of indistinguishability of encryptions is useful in demonstrating the security of candidate constructions, as well as for arguing about their usage as parts of larger protocols.

The Actual Definitions. In both definitions, we consider (feasible) adversaries that obtain, in addition to the ciphertext, auxiliary information that may depend on the potential plaintext (but not on the key). By $E(x)$ we denote the distribution of encryptions of x, when the key is selected at random. To simplify the exposition, let us assume that on security parameter n, the key-generation algorithm produces a key of length n, whereas the scheme is used to encrypt messages of length n^2.

Definition B.1.1 (Semantic Security (Following [123])): *An encryption scheme is* **semantically secure** *if for every feasible algorithm, A, there exists a feasible algorithm B such that for every two functions $f, h : \{0,1\}^* \to \{0,1\}^*$ and all sequences of pairs $(X_n, z_n)_{n\in\mathbb{N}}$, where X_n is a random variable ranging over $\{0,1\}^{n^2}$ and $|z_n|$ is of feasible* (in n) *length,*

$$\mathsf{Pr}[A(E(X_n)h(X_n), z_n) = f(X_n)] < \mathsf{Pr}[B(h(X_n), z_n) = f(X_n)] + \mu(n)$$

where μ is a negligible function. Furthermore, the complexity of B should be related to that of A.

What Definition B.1.1 says is that a feasible adversary does not gain anything by looking at the ciphertext. That is, whatever information (captured by the function f) it tries to compute about the ciphertext when given a priori information (captured by the function h) can essentially be computed as efficiently from the available a priori information alone. In particular, the ciphertext does not help in (feasibly) computing the least significant bit of the plaintext or any other information regarding the plaintext. This holds for any distribution of plaintexts (captured by the random variable X_n). We now turn to an equivalent definition.

Definition B.1.2 (Indistinguishability of Encryptions (Following [123])): *An encryption scheme has* **indistinguishable encryptions** *if for every feasible algorithm A and all sequences of triples $(x_n, y_n, z_n)_{n\in\mathbb{N}}$, where $|x_n| = |y_n| = n^2$ and $|z_n|$ is of feasible* (in n) *length,*

$$|\Pr[A(E(x_n), z_n) = 1] - \Pr[A(E(y_n), z_n) = 1]| < \mu(n)$$

where μ is a negligible function.

In particular, z_n may equal (x_n, y_n). Thus, it is infeasible to distinguish the encryptions of any two fixed messages such as the all-zeros message and the all-ones message.

Theorem B.1.3: *An encryption scheme is semantically secure if and only if it has indistinguishable encryptions.*

Probabilistic Encryption. It is easy to see that a secure public-key encryption scheme must employ a probabilistic (i.e., randomized) *encryption* algorithm. Otherwise, given the encryption key as (additional) input, it is easy to distinguish the encryption of the all-zeros message from the encryption of the all-ones message. The same holds for private-key encryption schemes when considering the security of encrypting several messages (rather than a single message as done before).[1] This explains the linkage between the foregoing robust security definitions and the *randomization paradigm* (discussed later).

B.1.2. Constructions

Private-key encryption schemes can be constructed based on the existence of one-way functions. In contrast, the known constructions of public-key encryption schemes seem to require stronger assumptions (such as the existence of trapdoor permutations).

B.1.2.1. Private-Key Schemes

It is common practice to use "pseudorandom generators" as a basis for private-key stream ciphers. We stress that this is a very dangerous practice when the "pseudorandom generator" is easy to predict (such as the linear congruential generator or some modifications of it that output a constant fraction of the bits of each resulting number [38, 84]). However, this common practice can become sound provided one uses pseudorandom generators as defined in Section 3.3. Thus, we obtain a *private-key stream cipher* that allows us to encrypt a stream of plaintext bits. Note that such a

[1]Here, for example, using a deterministic encryption algorithm allows the adversary to distinguish two encryptions of the same message from the encryptions of a pair of different messages.

stream cipher does not conform with our formulation of an encryption scheme, since for encrypting several messages we are required to maintain a counter. In other words, we obtain an encryption scheme with a variable state that is modified after the encryption of each message. To obtain a stateless encryption scheme, as in our earlier definitions, we can use a pseudorandom function.

Private-Key Encryption Scheme Based on Pseudorandom Functions. The key-generation algorithm consists of selecting a seed, denoted s, for such a function, denoted f_s. To encrypt a message $x \in \{0, 1\}^n$ (using key s), the encryption algorithm uniformly selects a string $r \in \{0, 1\}^n$ and produces the ciphertext $(r, x \oplus f_s(r))$. To decrypt the ciphertext (r, y) (using key s), the decryption algorithm just computes $y \oplus f_s(r)$. The proof of security of this encryption scheme consists of two steps (suggested as a general methodology in Section 3.6):

1. Prove that an idealized version of the scheme, in which one uses a uniformly selected function $f : \{0, 1\}^n \to \{0, 1\}^n$, rather than the pseudorandom function f_s, is secure.
2. Conclude that the real scheme (as presented earlier) is secure (since otherwise one could distinguish a pseudorandom function from a truly random one).

Note that we could have gotten rid of the randomization if we had allowed the encryption algorithm to be history-dependent (e.g., use a counter in the role of r). Furthermore, if the encryption scheme is used for FIFO communication between the parties and both can maintain the counter value, then there is no need for the message sender to transmit the counter value.

B.1.2.2. Public-Key Schemes

Here we use a collection of trapdoor one-way permutations, $\{p_\alpha\}_\alpha$, and a hard-core predicate, b, for it.

The Randomization Paradigm [123]. To demonstrate this paradigm, we first construct a simple public-key encryption scheme.

Key generation: The key-generation algorithm consists of selecting at random a permutation p_α together with a trapdoor for it; the permutation (or rather its description) serves as the public key, whereas the trapdoor serves as the private key.

Encrypting: To encrypt a single bit σ (using public key p_α), the encryption algorithm uniformly selects an element r in the domain of p_α and produces the ciphertext $(p_\alpha(r), \sigma \oplus b(r))$.

Decrypting: To decrypt the ciphertext (y, τ) using the private key, the decryption algorithm simply computes $\tau \oplus b(p_\alpha^{-1}(y))$, where the inverse is computed using the trapdoor (i.e., private key).

This scheme is quite wasteful of bandwidth. However, the paradigm underlying its construction is valuable in practice. For example, it is certainly better to randomly pad messages (say, using padding equal in length to the message) before encrypting them using RSA than to employ RSA on the plain message. Such a heuristic can be placed on firm ground if the following *conjecture* is supported: Assume that the first $n/2$ least significant bits of the argument constitute a hard-core function of RSA with n-bit-long moduli. Then, encrypting $n/2$-bit messages by padding the message with $n/2$ random bits and applying RSA (with an n-bit modulus) on the result will constitute a secure public-key encryption system, hereafter referred to as *Randomized RSA*.

An alternative public-key encryption scheme is presented in [35]. That encryption scheme augments Construction 3.4.4 (of a pseudorandom generator based on one-way permutations) as follows:

Key generation: As before, the key-generation algorithm consists of selecting at random a permutation p_α together with a trapdoor.

Encrypting: To encrypt the n-bit string x (using public key p_α), the encryption algorithm uniformly selects an element s in the domain of p_α and produces the ciphertext $(p_\alpha^n(s), x \oplus G_\alpha(s))$, where

$$G_\alpha(s) = b(s) \cdot b(p_\alpha(s)) \cdots b\big(p_\alpha^{n-1}(s)\big)$$

(We use the notation $p_\alpha^{i+1}(x) = p_\alpha(p_\alpha^i(x))$ and $p_\alpha^{-(i+1)}(x) = p_\alpha^{-1}(p_\alpha^{-i}(x))$.)

Decrypting: To decrypt the ciphertext (y, z) using the private key, the decryption algorithm first recovers $s = p_\alpha^{-n}(y)$ and then outputs $z \oplus G_\alpha(s)$.

Assuming that factoring Blum integers (i.e., products of two primes each congruent to $3 \pmod 4$) is hard, one can use the modular squaring function in the role of the trapdoor permutation and the least significant bit (denoted lsb) in the role of its hard-core predicate [35, 5, 208, 82]. This yields a secure public-key encryption scheme (depicted in Figure B.1) with efficiency comparable to that of RSA. Recall that RSA itself is not secure (as it employs a deterministic encryption algorithm), whereas Randomized

Private key: Two $n/2$-bit-long primes, p and q, each congruent to $3 \pmod 4$.
Public key: Their product $N \stackrel{\text{def}}{=} pq$.
Encryption of message $x \in \{0, 1\}^n$:

```
    1. Uniformly select $s_0 \in \{1, \ldots, N\}$.
    2. For $i = 1, \ldots, n+1$, compute $s_i \leftarrow s_{i-1}^2 \bmod N$ and $\sigma_i = \text{lsb}(s_i)$.
  The ciphertext is $(s_{n+1}, y)$, where $y = x \oplus \sigma_1\sigma_2 \cdots \sigma_n$.
```

Decryption of the ciphertext (r, y):

```
  Precomputed: $d_p = ((p+1)/4)^n \bmod p-1$, $d_q = ((q+1)/4)^n \bmod q-1$,
  $c_p = q \cdot (q^{-1} \bmod p)$, and $c_q = p \cdot (p^{-1} \bmod q)$.
    1. Let $s' \leftarrow r^{d_p} \bmod p$ and $s'' \leftarrow r^{d_q} \bmod q$.
    2. Let $s_1 \leftarrow c_p \cdot s' + c_q \cdot s'' \bmod N$.
    3. For $i = 1, \ldots, n$, compute $\sigma_i = \text{lsb}(s_i)$ and $s_{i+1} \leftarrow s_i^2 \bmod N$.
  The plaintext is $y \oplus \sigma_1\sigma_2 \cdots \sigma_n$.
```

Figure B.1: The Blum-Goldwasser public-key encryption scheme [35].

RSA (defined earlier) is not known to be secure under standard assumptions such as the intractability of factoring (or of inverting the RSA function).[2]

B.1.3. Beyond Eavesdropping Security

The foregoing definitions refer only to a "passive" attack in which the adversary merely eavesdrops on the communication line (over which ciphertexts are being sent). Stronger types of attacks, culminating in the so-called chosen ciphertext attack, may be possible in various applications. Furthermore, these definitions refer to an adversary that tries to extract explicit information about the plaintext. A less explicit attempt, captured by the so-called notion of *malleability*, is to generate an encryption of a related plaintext (possibly without learning anything about the original plaintext). Thus, we have a "matrix" of adversaries, with one dimension (parameter) being the *type of attack* and the second being its *purpose*.

Types of Attacks. The following mini-taxonomy of attacks certainly is not exhaustive:

1. *Passive attacks*, as captured in the foregoing definitions. Among public-key schemes, we distinguish two sub-cases:
   (a) A *key-oblivious* passive attack, as captured in the foregoing definitions. By "key-obliviousness" we refer to the fact that the choice of plaintext does not depend on the public key.
   (b) A *key-dependent* passive attack, in which the choice of plaintext may depend on the public key.

   (In Definition B.1.1 the choice of plaintext means the random variable X_n, whereas in Definition B.1.2 it means the pair of strings (x_n, y_n). In both of these definitions, the choice of the plaintext is non-adaptive.)
2. *Chosen plaintext attacks*. Here the attacker can obtain the encryption of any plaintext of its choice (under the key being attacked). Such an attack does not add power in case of public-key schemes.
3. *Chosen ciphertext attacks*. Here the attacker can obtain the decryption of any ciphertext of its choice (under the key being attacked). That is, the attacker is given oracle access to the decryption function corresponding to the decryption key in use. We distinguish two types of such attacks:
   (a) In an *a-priori-chosen* ciphertext attack, the attacker is given this oracle access prior to being presented the ciphertext that it will attack (i.e., the ciphertext for which it has to learn partial information or form a related ciphertext). That is, the attack consists of two stages: In the first stage the attacker is given the oracle access, and in the second stage the oracle is removed and the attacker is given a "test ciphertext" (i.e., a target to be learned or modified in violation of non-malleability).

[2] Recall that Randomized RSA is secure assuming that the $n/2$ least significant bits constitute a hard-core function for n-bit RSA moduli. We only know that the $O(\log n)$ least significant bits constitute a hard-core function for n-bit moduli [5].

(b) In an *a-posteriori-chosen* ciphertext attack the attacker is given the target ciphertext first, but its access to the oracle is restricted in that it is not allowed to make a query equal to the target ciphertext.

In both cases, the adversary can make queries that do not correspond to a legitimate ciphertext, and the answers will be accordingly (i.e., a special "failure" symbol).

Purpose of Attacks. Again, the following is not claimed to be exhaustive:

1. Standard *security*: the infeasibility of *obtaining information regarding the plaintext*. As defined earlier, such information must be a function (or a randomized process) applied to the bare plaintext and cannot depend on the encryption (or decryption) key.
2. In contrast, the notion of *non-malleability* [64] refers to generating a string depending on both the plaintext and the current encryption key. Specifically, one requires that it be infeasible for an adversary, given a ciphertext, to produce a valid ciphertext for a related plaintext. For example, given a ciphertext of a plaintext of the form $1x$, it should be infeasible to produce a ciphertext to the plaintext $0x$.

With the exception of passive attacks on private-key schemes, non-malleability always implies security against attempts to obtain information on the plaintext. Security and non-malleability are equivalent under a-posteriori-chosen ciphertext attack (cf. [64, 16]). For a detailed discussion of the relationships among the various notions of secure private-key and public-key encryptions, the reader is referred to [142] and [16], respectively.

Some Known Constructions. As in the basic case, the (strongly secure) private-key encryption schemes can be constructed based on the existence of one-way functions, whereas the (strongly secure) public-key encryption schemes are based on the existence of trapdoor permutations.

Private-key schemes: The private-key encryption scheme based on pseudorandom functions (described earlier) is secure also against a-priori-chosen ciphertext attacks.[3]

It is easy to turn any passively secure private-key encryption scheme into a scheme secure under (a posteriori) chosen ciphertext attacks by using a message-authentication scheme[4] on top of the basic encryption.

Public-key schemes: Public-key encryption schemes secure against a-priori-chosen ciphertext attacks can be constructed assuming the existence of trapdoor permutations and utilizing non-interactive zero-knowledge proofs see [176]. (Recall that the latter proof systems can be constructed under the former assumption.)

[3]Note that this scheme is not secure under an a-posteriori-chosen ciphertext attack: On input a ciphertext $(r, x \oplus f_s(r))$, we obtain $f_s(r)$ by making the query (r, y'), where $y' \neq x \oplus f_s(r)$. (This query is answered with x' such that $y' = x' \oplus f_s(r)$.)

[4]See definition in Section B.2.

Public-key encryption schemes secure against a-posteriori-chosen ciphertext attacks can also be constructed under the same assumption [64], but this construction is even more complex.

In fact, both constructions of *public-key* encryption schemes secure against chosen ciphertext attacks are to be considered as plausibility results (which also offer some useful construction paradigms). Presenting "reasonably efficient" public-key encryption schemes that are secure against (a posteriori) chosen ciphertext attacks, under widely believed assumptions, is an important open problem.[5]

B.1.4. Some Suggestions

B.1.4.1. Suggestions for Further Reading

Fragments of a preliminary draft for the intended chapter on encryption schemes can be obtained online [99].

In addition, there are the original papers: There is a good motivating discussion in [123], but we prefer the definitional treatment of [92, 94], which can be substantially simplified if one adopts non-uniform complexity measures (as done above).[6] Further details on the construction of public-key encryption schemes (sketched above) can be found in [123, 92, 35, 5]. For discussion of non-malleable cryptography, which actually transcends the domain of encryption, see [64].

B.1.4.2. Suggestions for Teaching

We suggest a focus on the basic notion of security (treated in Sections B.1.1 and B.1.2): Present both definitions, prove their equivalence, and discuss the need to use *randomness* during the encryption process in order to meet these definitions. Next, present all constructions described in Section B.1.2. We believe that the draft available online [99] provides sufficient details for all of these.

B.2. Signatures: Brief Summary

Again, there are private-key and public-key versions, both consisting of three efficient algorithms: *key generation*, *signing*, and *verification*. (Private-key signature schemes are commonly referred to as *message-authentication schemes* or *codes* (MAC).) The difference between the two types is again reflected in the definitions of security. This difference yields different functionalities (even more than in the case of encryption): Public-key signature schemes (hereafter referred to as signature schemes) can be used to produce signatures that are *universally verifiable* (given access to the public key of the signer). Private-key signature schemes (hereafter referred to as message-authentication schemes) typically are used to authenticate messages sent

[5]The "reasonably efficient" scheme of [57] is based on a strong assumption regarding the Diffie-Hellman key exchange. Specifically, it is assumed that for a prime P and primitive element g, given $(P, g, (g^x \bmod P), (g^y \bmod P), (g^z \bmod P))$, it is infeasible to decide whether or not $z \equiv xy \pmod{P-1}$.

[6]We comment that [92] follows [94] in providing a uniform-complexity treatment of the security of encryption schemes.

among a (small) set of *mutually trusting* parties (since the ability to verify signatures may be linked to the ability to produce them). In other words, message-authentication schemes are used to authenticate information sent between (typically two) parties, and the purpose is to *convince the receiver* that the information has indeed been sent by the legitimate sender. In particular, message-authentication schemes cannot *convince a third party* that the sender has indeed sent the information (rather than the receiver having generated it by itself). In contrast, public-key signatures can be used to convince third parties: A signature to a document typically is sent to a second party, so that in the future that party can (by merely presenting the signed document) convince third parties that the document was indeed generated/sent/approved by the signer.

B.2.1. Definitions

We consider very powerful attacks on the signature scheme as well as a very liberal notion of breaking it. Specifically, the attacker is allowed to obtain signatures to any message of its choice. One may argue that in many applications such a general attack is not possible (since messages to be signed must have a specific format). Yet our view is that it is impossible to define a general (i.e., application-independent) notion of admissible messages, and thus it seems that a general/robust definition of an attack must be formulated as suggested here. (Note that, at worst, our approach is overly cautious.) Likewise, the adversary is said to be successful if it can produce a valid signature to ANY message for which it has not asked for a signature during its attack. Again, this defines the ability to form signatures to possibly "non-sensical" messages as a breaking of the scheme. Yet, again, we see no way to have a general (i.e., application-independent) notion of "meaningful" messages (so that only forging signatures to them would be consider a breaking of the scheme).

Definition B.2.1 (Unforgeable Signatures [125]):

- *A* **chosen message attack** *is a process that on input a verification key can obtain signatures* (relative to the corresponding signing key) *to messages of its choice.*
- *Such an attack is said to* **succeed** (in existential forgery) *if it outputs a valid signature to a message for which it has* not *requested a signature during the attack.*
- *A signature scheme is* **secure** (or unforgeable) *if every* feasible *chosen message attack succeeds with at most negligible probability.*

We stress that *plain* RSA (like plain versions of Rabin's scheme [187] and DSS [169]) is not secure under the foregoing definition. However, it may be secure if the message is "randomized" before RSA (or another scheme) is applied [22]. Thus the randomization paradigm seems pivotal here too.

The definition of security for message-authentication schemes is similar, except that the attacker does not get the verification key as input.

B.2.2. Constructions

Both message-authentication and signature schemes can be constructed based on the existence of one-way functions.

B.2.2.1. Message Authentication

Message-authentication schemes can be constructed using pseudorandom functions [103]: To authenticate the message x with respect to key s, one generates the tag $f_s(x)$, where f_s is the pseudorandom function associated with s. Verification is done in the same (analogous) way. However, as noted in [15], *extensive* use of pseudorandom functions would seem to be overkill for achieving message authentication, and more efficient schemes can be obtained based on other cryptographic primitives. We mention two approaches:

1. *fingerprinting* the message using a scheme that is *secure against forgery provided that the adversary does not have access to the scheme's outcome* (e.g., using Universal Hashing [49]), and "*hiding*" the result using a *non-malleable* scheme (e.g., a private-key encryption or a pseudorandom function). (Non-malleability is not required in certain cases [209].)
2. *hashing* the message *using a collision-free scheme* [58, 59] and *authenticating* the result using a MAC that operates on (short) fixed-length strings [15].

B.2.2.2. Signature Schemes

Three central paradigms in the construction of signature schemes are the "refreshing" of the "effective" signing key, the use of an "authentication tree," and the "hashing paradigm."

The Refreshing Paradigm [125]. To demonstrate this paradigm, suppose we have a signature scheme that is robust against a "random message attack" (i.e., an attack in which the adversary obtains signatures only to randomly chosen messages). Further suppose that we have a *one-time* signature scheme (i.e., a signature scheme that is secure against an attack in which the adversary obtains a signature to a *single* message of its choice). Then we can obtain a secure signature scheme as follows: When a new message needs to be signed, we generate a new random signing key for the one-time signature scheme, use it to sign the message, and sign the corresponding (one-time) verification key using the fixed signing key of the main signature scheme[7] (which is robust against a "random message attack") [71]. We note that one-time signature schemes (as utilized here) are easy to construct (e.g., [161]).

The Authentication-Tree Paradigm [160, 125]. To demonstrate this paradigm, we show how to construct a general signature scheme using only a one-time signature scheme (alas, one where a $2n$-bit string can be signed with respect to an n-bit-long

[7]Alternatively, one can generate the one-time key pair and the signature to its verification key ahead of time, leading to an "off-line/on-line" signature scheme [71].

verification key). The idea is to use the initial signing key (i.e., the one corresponding to the public verification key) in order to sign/authenticate two new/random verification keys. The two corresponding signing keys are used to sign/authenticate four new/random verification keys (two per each signing key), and so on. Stopping after ℓ such steps, this process forms a binary tree with 2^ℓ leaves, where each leaf corresponds to an instance of the one-time signature scheme. The signing keys at the leaves can be used to sign the actual messages, and the corresponding verification keys can be authenticated using the path from the root. That is, to sign a new message, we proceed as follows:

1. Allocate a new leaf in the tree. This requires either keeping a counter of the number of messages signed thus far or selecting a leaf at random (assuming that the number of leaves is much larger than the square of the number of messages to be signed).
2. Generate or retrieve from storage the pairs of signing/verification keys corresponding to each vertex on the path from the root to the selected leaf, along with the key pairs of the siblings of the vertices on the path. That is, let $v_0, v_1, \ldots, v_\ell$ denote the vertices along the path from the root v_0 to the selected leaf v_ℓ, and let u_i be the sibling of v_i (for $i = 1, \ldots, \ell$). Then we generate/retrieve the key pairs of each v_i and each u_i, for $i = 1, \ldots, \ell$.

   It is important to use the same key pair when encountering the same vertex in the process of signing two different messages.
3. Sign the message using the signing key associated with the selected leaf v_ℓ. Sign each pair of verification keys associated with the children of each internal vertex, along the foregoing path, using the signing key associated with the parent vertex. That is, for $i = 1, \ldots, \ell$, sign the verification keys of v_i and u_i (placed in some canonical order) using the signing key associated with vertex v_{i-1}.

   The signature is obtained by concatenating all these signatures (along with the corresponding verification keys). Recall that the key pair associated with the root is the actual key pair of the signature scheme; that is, the verification component is placed in the public file, and the signature of the verification keys of the root's children (relative to the root's signing key) is part of all signatures.

Pseudorandom functions can be used to eliminate the need to store the values of vertices used in previous signatures [89].

Employing this paradigm, and assuming that the RSA function is infeasible to invert, one obtains a secure signature scheme [125, 89] (with a counter of the number of messages signed) in which the ith message is signed/verified in time $2 \log_2 i$ slower than plain RSA. Using a tree of large fan-in (and assuming again that RSA is infeasible to invert), one can obtain a secure signature scheme [67, 56] that for reasonable parameters is only five times slower than plain RSA.[8] We stress that plain RSA is not a secure signature scheme, whereas the security of its randomized version (mentioned earlier) is not known to be reducible to the assumption that RSA is hard to invert.

[8]This figure refers to signing up to 1,000,000,000 messages. The scheme in [67] requires a universal set of system parameters consisting of 1000–2000 integers of the size of the moduli. In the scheme of [56], that requirement is removed.

The Hashing Paradigm. A common practice is to sign real documents via a two-stage process: First the document is hashed into a (relatively) short bit string, and then the basic signature scheme is applied to the resulting string. We note that this heuristic becomes sound provided the hashing function is *collision-free* (as defined in [58]). Collision-free hashing functions can be constructed, assuming the existence of claw-free collections (as in Definition 2.4.6) [58]. One can indeed postulate that certain off-the-shelf products (e.g., MD5 or SHA) are collision-free, but such assumptions need to be tested (and indeed may turn out false). We stress that using a hashing scheme in the foregoing two-stage process without carefully evaluating whether or not it is collision-free is a very dangerous practice.

One useful variant on the foregoing paradigm is the use of *universal one-way hashing functions* (as defined in [175]), rather than the collision-free hashing used earlier. In such a case, a new hashing function is selected for each application of the scheme, and the basic signature scheme is applied both to the (succinct) description of the hashing function and to the resulting (hashed) string. (In contrast, when using a collision-free hashing function, the same function, the description of which is part of the signer's public key, is used in all applications of the signature scheme.) The advantage of using universal one-way hashing functions is that their security requirement seems weaker than that for the collision-free condition (e.g., the former can be constructed using any one-way function [192], whereas this is NOT known for the latter).

Theorem B.2.2 (Plausibility Result [175, 192]): *Signature schemes exist if and only if one-way functions exist.*

Unlike the paradigms (and some of the constructions) described earlier, the known construction of signature schemes from *arbitrary* one-way functions has no practical significance. It is indeed an important open problem to provide an alternative construction that can be practical and still utilize an *arbitrary* one-way function.

B.2.3. Some Suggestions

B.2.3.1. Suggestions for Further Reading

Fragments of a preliminary draft for the intended chapter on signature schemes can be obtained on line [100].

In addition, there are the original papers: For a definitional treatment of *signature schemes*, the reader is referred to [125] and [183]. Easy-to-understand constructions appear in [20, 71, 67]. The proof of Theorem B.2.2 can be extracted from [175, 192]: The first paper presents the basic approach and implements it using any one-way permutation, whereas the second paper shows how to implement this approach using any one-way function. Variants on the basic model are discussed in [183] and in [50, 137]. For discussion of *message-authentication schemes* (MACs) the reader is referred to [15].

B.2.3.2. Suggestions for Teaching

We suggest a focus on signature schemes, presenting the main definition and some construction. One may use [125] for the definitional treatment, but should *not* use it for the construction, the underlying ideas of which are more transparent in papers such as [20] and [175]. Actually, we suggest presenting a variant on the signature scheme of [175], using collision-free hashing (cf. [58]) instead of universal one-way hashing (cf. [175]). This allows one to present, within a few lectures, many important paradigms and techniques (e.g., the refreshing paradigm, authentication trees, the hashing paradigm, and one-time signature schemes). We believe that the draft available online [100] provides sufficient details for such a presentation.

A basic treatment of message authentication (i.e., motivation, definition, and construction based on pseudorandom functions) can be presented within one lecture, and [100] can be used for this purpose too. (This, however, will not cover alternative approaches employed toward the construction of more efficient message-authentication schemes.)

B.3. Cryptographic Protocols: Brief Summary

A general framework for casting cryptographic (protocol) problems consists of specifying a random process that maps n inputs to n outputs. The inputs to the process are to be thought of as local inputs of n parties, and the n outputs are their corresponding local outputs. The random process describes the desired functionality. That is, if the n parties were to trust each other (or trust some outside party), then each could send its local input to the trusted party, who would compute the outcome of the process and send each party the corresponding output. The question addressed in this section is the extent to which this trusted party can be "emulated" by the mutually distrustful parties themselves.

B.3.1. Definitions

For simplicity, we consider the special case where the specified process is deterministic and the n outputs are identical. That is, we consider an arbitrary n-ary function and n parties that wish to obtain the value of the function on their n corresponding inputs. Each party wishes to obtain the correct value of the function and prevent any other party from gaining anything else (i.e., anything beyond the value of the function and what is implied by it).

We first observe that (one thing that is unavoidable is that) each party can change its local input before entering the protocol. However, this is also unavoidable when the parties utilize a trusted party. In general, the basic paradigm underlying the definitions of *secure multi-party computations*[9] amounts to saying that situations that may occur in the real protocol can be simulated in an ideal model (where the parties can employ a trusted party). Thus, the "effective malfunctioning" of parties in secure protocols is restricted to what is postulated in the corresponding ideal model. The specific definitions

[9] Our current understanding of the definitional issues is most indebted to the high-level discussions in the unfinished manuscript of [165]. A similar definitional approach is presented in [11, 12]. The approach of [122] is more general: It avoids the definition of security (w.r.t a given functionality) and defines instead a related notion of protocol robustness. One minimalistic instantiation of the definitional approach of [165, 11, 12] is presented in [45] and is shown to satisfy the main conceptual concerns.

differ in the specific restrictions and/or requirements placed on the parties in the real computation. This typically is reflected in the definition of the corresponding ideal model; see the examples that follow.

B.3.1.1. An Example: Computations with Honest Majority

Here we consider an ideal model in which any minority group (of the parties) can collude as follows. First, this minority shares its original inputs and decides together on replacement inputs[10] to be sent to the trusted party. (The other parties send their respective original inputs to the trusted party.) When the trusted party returns the output, each majority player outputs it locally, whereas the colluding minority can compute an output based on all they know (i.e., the output and all the local inputs of these parties). A *secure multi-party computation with honest majority* is required to simulate this ideal model. That is, the effect of any feasible adversary that controls a minority of the players in the actual protocol can essentially be simulated by a (different) feasible adversary that controls the corresponding players in the ideal model. This means that in a secure protocol the effect of each minority group is "essentially restricted" to replacing its own local inputs (independently of the local inputs of the majority players) before the protocol starts and replacing its own local outputs (depending only on its local inputs and outputs) after the protocol terminates. (We stress that in the real execution the minority players do obtain additional pieces of information; yet in a secure protocol they gain essentially nothing from these additional pieces of information.)

Secure protocols according to this definition can even tolerate a situation where a minority of the parties choose to abort the execution. An aborted party (in the real protocol) is simulated by a party (in the ideal model) that aborts the execution either before supplying its input to the trusted party (in which case a default input is used) or after supplying its input. In either case, the majority players (in the real protocol) are able to compute the output even though a minority aborted the execution. This cannot be expected to happen when there is no honest majority (e.g., in a two-party computation) [53].

B.3.1.2. Another Example: Two-Party Computations Allowing Abort

In light of the foregoing, we consider an ideal model where each of the two parties can "shut down" the trusted (third) party at any point in time. In particular, this can happen after the trusted party has supplied the outcome of the computation to one party but before it has supplied it to the second. A *secure two-party computation allowing abort* is required to simulate this ideal model. That is, each party's "effective malfunctioning" in such a secure protocol is restricted to supplying an initial input of its choice and aborting the computation at any point in time. We stress that, as before, the choice of the initial input of each party cannot depend on the input of the other party.

[10]Such replacement can be avoided if the local inputs of parties are verifiable by the other parties. In such a case, a party (in the ideal model) has the choice of either joining the execution of the protocol with its correct local input or not joining the execution at all (but it cannot join with a replaced local input). Secure protocols simulating this ideal model can be constructed as well.

Generalizing the preceding, we can consider *secure multi-party computation allowing abort*. Here, in the ideal model, each of the parties can "shut down" the trusted party at any point in time; in particular, this can happen after the trusted party has supplied the outcome of the computation to some but not all of the parties.

B.3.2. Constructions

Theorem B.3.1 (General Plausibility Results, Loosely Stated): *Suppose that trapdoor permutations exist. Then*

- *any multi-party functionality can be securely computed in a model allowing abort* (cf. [211] for the two-party case and [113] for the case of more than two parties).
- *any multi-party functionality can be securely computed provided that a strict majority of the parties are honest* [112, 113].

The proof of each item proceeds in two steps [98]:

1. Presenting secure protocols for a "semi-honest" model in which the bad parties follow the protocol, except that they also keep a record of all intermediate results.[11]

   One key idea is to consider the propagation of values along the wires of a circuit (which computes the desired function), going from the input wires to the output wires. The execution of these protocols starts by each party sharing its inputs with all other parties, using a secret sharing scheme, so that any strict subset of the shares yields no information about the secret (e.g., each party is given a uniformly chosen share, and the dealer's share is set to the XOR of all other shares). A typical step consists of the secure computation of shares of the output wire of a gate from the shares of the input wires of this gate. That is, the m parties employ a secure protocol for computing the randomized m-party functionality $((a_1, b_1), \ldots, (a_m, b_m)) \mapsto (c_1, \ldots, c_m)$, where the c_i's are uniformly distributed subject to $\oplus_{i=1}^m c_i = \text{gate}(\oplus_{i=1}^m a_i, \oplus_{i=1}^m b_i)$. Repeating this step for each gate of the circuit (in a suitable order), the parties securely propagate shares along the wires of the circuit, going from the input wires of the circuit to its output wires. At the end of this propagation process, each party announces its shares in the output wires of the circuit, and the actual output is formed. Thus, securely computing an arbitrary functionality (which may be quite complex) is reduced to securely computing a few specific simple functionalities (i.e., given shares for the inputs of a Boolean gate, securely compute random shares for the output of this gate). Indeed, secure protocols for computing these simple functionalities are also provided.

2. Transforming protocols secure in the "semi-honest" model into full-fledged secure protocols. Here zero-knowledge proofs and protocols for fair coin-tossing are used in order to "force" parties to behave properly (i.e., as in the "semi-honest" model).

   Fair coin-tossing protocols are constructed using non-oblivious commitment schemes (see Section 4.9.2), which in turn rely on zero-knowledge proofs of knowledge (see Section 4.7).

[11] In other words, we need to simulate the local views of the dishonest players when given only the local inputs and outputs of the honest players. Indeed, this model corresponds to the honest-verifier model of zero-knowledge (see Section 4.3.1.7).

We stress the general nature of these constructions and view them as plausibility results asserting that a host of cryptographic problems are solvable, assuming the existence of trapdoor permutations. As discussed in the case of zero-knowledge proofs, the value of these general results is in allowing one to easily infer that the problem he/she faces is solvable in principle (as typically it is easy to cast problems within this framework). However, we do not recommend using (in practice) the solutions derived by these general results; one should rather focus on the specifics of the problem at hand and solve it using techniques and/or insights available from these general results.[12]

Analogous plausibility results have been obtained in a variety of models. In particular, we mention secure computations in the private-channels model [27, 51] and in the presence of mobile adversaries [182].

B.3.3. Some Suggestions

B.3.3.1. Suggestions for Further Reading

A draft of a manuscript that is intended to cover this surveyed material is available online from [98]. The draft provides an exposition of the basic definitions and results, as well as detailed proofs for the latter. More refined discussions of definitional issues can be found in [11, 12, 44, 45, 122, 165]; our advice is to start with [45].

B.3.3.2. Suggestions for Teaching

This area is very complex, and so we suggest that one merely present *sketches* of some definitions and constructions. Specifically, we suggest picking one of the two settings (i.e., computation with honest majority or two-party computation) and sketching the definition and the construction. Our own choice would be the two-party case; alas, the definition (allowing abort) is more complicated (but this is more than compensated for by simpler notation and a simpler construction that relies on relatively fewer ideas). We suggest emphasizing the definitional approach (i.e., "emulating a trusted party" as simulation of any adversary operating in the real model by an ideal-model adversary) and presenting the main ideas underlying the construction (while possibly skipping a few). We believe that the draft available online from [98] provides sufficient details for all of these.

[12]For example, although Threshold Cryptography (cf., [62, 87]) is merely a special case of multi-party computation, it is indeed beneficial to focus on its specifics.

Bibliography

[1] L.M. Adleman and M. Huang. *Primality Testing and Abelian Varieties Over Finite Fields*. Springer-Verlag Lecture Notes in Computer Science (Vol. 1512), 1992. (Preliminary version in *19th ACM Symposium on the Theory of Computing*, 1987.)

[2] W. Aiello and J. Håstad. Perfect Zero-Knowledge Languages Can Be Recognized in Two Rounds. In *28th IEEE Symposium on Foundations of Computer Science*, pages 439–448, 1987.

[3] M. Ajtai. Generating Hard Instances of Lattice Problems. In *28th ACM Symposium on the Theory of Computing*, pages 99–108, 1996.

[4] M. Ajtai, J. Komlos, and E. Szemerédi. Deterministic Simulation in LogSpace. In *19th ACM Symposium on the Theory of Computing*, pages 132–140, 1987.

[5] W. Alexi, B. Chor, O. Goldreich, and C.P. Schnorr. RSA/Rabin Functions: Certain Parts Are as Hard as the Whole. *SIAM Journal on Computing*, Vol. 17, April, pages 194–209, 1988.

[6] N. Alon and J.H. Spencer. *The Probabilistic Method*. Wiley, 1992.

[7] T.M. Apostol. *Introduction to Analytic Number Theory*. Springer, 1976.

[8] L. Babai. Trading Group Theory for Randomness. In *17th ACM Symposium on the Theory of Computing*, pages 421–420, 1985.

[9] E. Bach. *Analytic Methods in the Analysis and Design of Number-Theoretic Algorithms*. ACM Distinguished Dissertation (1984). MIT Press, Cambridge, MA, 1985.

[10] E. Bach and J. Shallit. *Algorithmic Number Theory. Vol. I: Efficient Algorithms*. MIT Press, Cambridge, MA, 1996.

[11] D. Beaver. Foundations of Secure Interactive Computing. In *Crypto91*, Springer-Verlag Lecture Notes in Computer Science (Vol. 576), pages 377–391, 1992.

[12] D. Beaver. Secure Multi-Party Protocols and Zero-Knowledge Proof Systems Tolerating a Faulty Minority. *Journal of Cryptology*, Vol. 4, pages 75–122, 1991.

[13] M. Bellare. A Note on Negligible Functions. Tech. Rep. CS97-529, Department of Computer Science and Engineering, UCSD, March 1997.

[14] M. Bellare, R. Canetti, and H. Krawczyk. Pseudorandom Functions Revisited: The Cascade Construction and Its Concrete Security. In *37th IEEE Symposium on Foundations of Computer Science*, pages 514–523, 1996.

[15] M. Bellare, R. Canetti, and H. Krawczyk. Keying Hash Functions for Message Authentication. In *Crypto96*, Springer-Verlag Lecture Notes in Computer Science (Vol. 1109), pages 1–15, 1996.

[16] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations Among Notions of Security for Public-Key Encryption Schemes. In *Crypto98*, Springer-Verlag Lecture Notes in Computer Science (Vol. 1462), pages 26–45, 1998.

[17] M. Bellare and O. Goldreich. On Defining Proofs of Knowledge. In *Crypto92*, Springer-Verlag Lecture Notes in Computer Science (Vol. 740), pages 390–420, 1992.

[18] M. Bellare, S. Halevi, A. Sahai, and S. Vadhan. Trapdoor Functions and Public-Key Cryptosystems. In *Crypto98*, Springer-Verlag Lecture Notes in Computer Science (Vol. 1462), pages 283–298, 1998.

[19] M. Bellare, R. Impagliazzo, and M. Naor. Does Parallel Repetition Lower the Error in Computationally Sound Protocols? In *38th IEEE Symposium on Foundations of Computer Science*, pages 374–383, 1997.

[20] M. Bellare and S. Micali. How to Sign Given Any Trapdoor Function. *Journal of the ACM*, Vol. 39, pages 214–233, 1992.

[21] M. Bellare and P. Rogaway. Random Oracles Are Practical: A Paradigm for Designing Efficient Protocols. In *1st Conference on Computer and Communications Security*, ACM, pages 62–73, 1993.

[22] M. Bellare and P. Rogaway. The Exact Security of Digital Signatures: How to Sign with RSA and Rabin. In *EuroCrypt96*, Springer-Verlag Lecture Notes in Computer Science (Vol. 1070), pp. 399–416, 1996.

[23] M. Bellare and M. Yung. Certifying Permutations: Noninteractive Zero-Knowledge Based on Any Trapdoor Permutation. *Journal of Cryptology*, Vol. 9, pages 149–166, 1996.

[24] S. Ben-David, B. Chor, O. Goldreich, and M. Luby. On the Theory of Average Case Complexity. *Journal of Computer and System Science*, Vol. 44, No. 2, April, pages 193–219, 1992.

[25] M. Ben-Or, O. Goldreich, S. Goldwasser, J. Håstad, J. Kilian, S. Micali, and P. Rogaway. Everything Provable Is Probable in Zero-Knowledge. In *Crypto88*, Springer-Verlag Lecture Notes in Computer Science (Vol. 403), pages 37–56, 1990.

[26] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson. Multi-Prover Interactive Proofs: How to Remove Intractability. In *20th ACM Symposium on the Theory of Computing*, pages 113–131, 1988.

[27] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In *20th ACM Symposium on the Theory of Computing*, pages 1–10, 1988.

[28] E.R. Berlekamp. Factoring Polynomials over Large Finite Fields. *Mathematics of Computation*, Vol. 24, pages 713–735, 1970.

[29] E.R. Berlekamp, R.J. McEliece, and H.C.A. van Tilborg. On the Inherent Intractability of Certain Coding Problems. *IEEE Transactions on Information Theory*, 1978.

[30] M. Blum. How to Exchange Secret Keys. *ACM Trans. Comput. Sys.*, Vol. 1, pages 175–193, 1983.

[31] M. Blum. Coin Flipping by Phone. In *24th IEEE Computer Conference* (*CompCon*), February, pages 133–137, 1982. (See also *SIGACT News*, Vol. 15, No. 1, 1983.)

[32] L. Blum, M. Blum, and M. Shub. A Simple Secure Unpredictable Pseudo-Random Number Generator. *SIAM Journal on Computing*, Vol. 15, pages 364–383, 1986.

[33] M. Blum, A. De Santis, S. Micali, and G. Persiano. Non-interactive Zero-Knowledge Proof Systems. *SIAM Journal on Computing*, Vol. 20, No. 6, pages 1084–1118, 1991. (Considered the journal version of [34].)

[34] M. Blum, P. Feldman, and S. Micali. Non-Interactive Zero-Knowledge and Its Applications. In *20th ACM Symposium on the Theory of Computing*, pages 103–112, 1988. (See [33].)

[35] M. Blum and S. Goldwasser. An Efficient Probabilistic Public-Key Encryption Scheme

which Hides All Partial Information. In *Crypto84*, Springer-Verlag Lecture Notes in Computer Science (Vol. 196), pages 289–302, 1985.

[36] M. Blum and S. Micali. How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits. *SIAM Journal on Computing*, Vol. 13, pages 850–864, 1984. (Preliminary version in *23rd IEEE Symposium on Foundations of Computer Science*, 1982.)

[37] R. Boppana, J. Håstad, and S. Zachos. Does Co-NP Have Short Interactive Proofs? *Information Processing Letters*, Vol. 25, May, pages 127–132, 1987.

[38] J.B. Boyar. Inferring Sequences Produced by Pseudo-Random Number Generators. *Journal of the ACM*, Vol. 36, pages 129–141, 1989.

[39] G. Brassard. A Note on the Complexity of Cryptography. *IEEE Transactions on Information Theory*, Vol. 25, pages 232–233, 1979.

[40] G. Brassard, D. Chaum, and C. Crépeau. Minimum Disclosure Proofs of Knowledge. *Journal of Computer and System Science*, Vol. 37, No. 2, pages 156–189, 1988. (Preliminary version by Brassard and Crépeau in *27th IEEE Symposium on Foundations of Computer Science*, 1986.)

[41] G. Brassard and C. Crépeau. Zero-Knowledge Simulation of Boolean Circuits. In *Crypto86*, Springer-Verlag Lecture Notes in Computer Science (Vol. 263), pages 223–233, 1987.

[42] G. Brassard, C. Crépeau, and M. Yung. Constant-Round Perfect Zero-Knowledge Computationally Convincing Protocols. *Theoretical Computer Science*, Vol. 84, pages 23–52, 1991.

[43] E.F. Brickell and A.M. Odlyzko. Cryptanalysis: A Survey of Recent Results. In *Proceedings of the IEEE*, Vol. 76, pages 578–593, 1988.

[44] R. Canetti. *Studies in Secure Multi-Party Computation and Applications*. Ph.D. thesis, Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel, June 1995. (Available from `http://theory.lcs.mit.edu/~tcryptol/BOOKS/ran-phd.html`.)

[45] R. Canetti. Security and Composition of Multi-party Cryptographic Protocols. *Journal of Cryptology*, Vol. 13, No. 1, pages 143–202, 2000.

[46] R. Canetti, O. Goldreich, and S. Halevi. The Random Oracle Methodology, Revisited. In *30th ACM Symposium on the Theory of Computing*, pages 209–218, 1998.

[47] R. Canetti, O. Goldreich, S. Goldwasser, and S. Micali. Resettable Zero-Knowledge. In *32nd ACM Symposium on the Theory of Computing*, pages 235–244, 2000.

[48] E.R. Canfield, P. Erdos, and C. Pomerance. On a Problem of Oppenheim Concerning "factorisatio numerorum." *Journal of Number Theory*, Vol. 17, pages 1–28, 1983.

[49] L. Carter and M. Wegman. Universal Hash Functions. *Journal of Computer and System Science*, Vol. 18, pages 143–154, 1979.

[50] D. Chaum. Blind Signatures for Untraceable Payments. In *Crypto82*, pages 199–203, Plenum Press, New York, 1983.

[51] D. Chaum, C. Crépeau, and I. Damgård. Multi-party Unconditionally Secure Protocols. In *20th ACM Symposium on the Theory of Computing*, pages 11–19, 1988.

[52] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults. In *26th IEEE Symposium on Foundations of Computer Science*, pages 383–395, 1985.

[53] R. Cleve. Limits on the Security of Coin Flips When Half the Processors Are Faulty. In *18th ACM Symposium on the Theory of Computing*, pages 364–369, 1986.

[54] J.D. Cohen and M.J. Fischer. A Robust and Verifiable Cryptographically Secure Election Scheme. In *26th IEEE Symposium on Foundations of Computer Science*, pages 372–382, 1985.

[55] A. Cohen and A. Wigderson. Dispensers, Deterministic Amplification, and Weak Random

Sources. In *30th IEEE Symposium on Foundations of Computer Science*, pages 14–19, 1989.

[56] R. Cramer and I. Damgård. New Generation of Secure and Practical RSA-based Signatures. In *Crypto96*, Springer-Verlag Lecture Notes in Computer Science (Vol. 1109), pages 173–185, 1996.

[57] R. Cramer and V. Shoup. A Practical Public-Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attacks. In *Crypto98*, Springer-Verlag Lecture Notes in Computer Science (Vol. 1462), pages 13–25, 1998.

[58] I. Damgård. Collision Free Hash Functions and Public Key Signature Schemes. In *EuroCrypt87*, Springer-Verlag Lecture Notes in Computer Science (Vol. 304), pages 203–216, 1988.

[59] I. Damgård. A Design Principle for Hash Functions. In *Crypto89*, Springer-Verlag Lecture Notes in Computer Science (Vol. 435), pages 416–427, 1990.

[60] I. Damgård. Concurrent Zero-Knowledge Is Easy in Practice. Theory of Cryptography Library, 99-14, June 1999. `http://philby.ucsd.edu/cryptolib`.

[61] I. Damgård, O. Goldreich, T. Okamoto, and A. Wigderson. Honest Verifier vs Dishonest Verifier in Public Coin Zero-Knowledge Proofs. In *Crypto95*, Springer-Verlag Lecture Notes in Computer Science (Vol. 963), pages 325–338, 1995.

[62] Y. Desmedt and Y. Frankel. Threshold Cryptosystems. In *Crypto89*, Springer-Verlag Lecture Notes in Computer Science (Vol. 435), pages 307–315, 1990.

[63] W. Diffie and M.E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22 (Nov.), pages 644–654, 1976.

[64] D. Dolev, C. Dwork, and M. Naor. Non-malleable Cryptography. In *23rd ACM Symposium on the Theory of Computing*, pages 542–552, 1991. (Full version available from authors.)

[65] D. Dolev and A.C. Yao. On the Security of Public-Key Protocols. *IEEE Transactions on Information Theory*, Vol. 30, No. 2, pages 198–208, 1983.

[66] C. Dwork, U. Feige, J. Kilian, M. Naor, and S. Safra. Low Communication Perfect Zero Knowledge Two Provers Proof Systems. In *Crypto92*, Springer-Verlag Lecture Notes in Computer Science (Vol. 740), pages 215–227, 1992.

[67] C. Dwork and M. Naor. An Efficient Existentially Unforgeable Signature Scheme and its Application. *Journal of Cryptology*, Vol. 11, No. 3, pages 187–208, 1998.

[68] C. Dwork, M. Naor, and A. Sahai. Concurrent Zero-Knowledge. In *30th STOC*, pages 409–418, 1998.

[69] S. Even and O. Goldreich. On the Security of Multi-party Ping-Pong Protocols. In *24th IEEE Symposium on Foundations of Computer Science*, pages 34–39, 1983.

[70] S. Even, O. Goldreich, and A. Lempel. A Randomized Protocol for Signing Contracts. *CACM*, Vol. 28, No. 6, pages 637–647, 1985.

[71] S. Even, O. Goldreich, and S. Micali. On-line/Off-line Digital Signatures. *Journal of Cryptology*, Vol. 9, pages 35–67, 1996.

[72] S. Even, A.L. Selman, and Y. Yacobi. The Complexity of Promise Problems with Applications to Public-Key Cryptography. *Information and Control*, Vol. 61, pages 159–173, 1984.

[73] S. Even and Y. Yacobi. Cryptography and NP-Completeness. In *Proceedings of 7th ICALP*, Springer-Verlag Lecture Notes in Computer Science (Vol. 85), pages 195–207, 1980. (See [72].)

[74] U. Feige. Error Reduction by Parallel Repetition – The State of the Art. Technical Report CS95-32, Computer Science Department, Weizmann Institute of Science, Rehovot, Israel, 1995.

[75] U. Feige, A. Fiat, and A. Shamir. Zero-Knowledge Proofs of Identity. *Journal of Cryptology*, Vol. 1, pages 77–94, 1988.

[76] U. Feige, D. Lapidot, and A. Shamir. Multiple Non-Interactive Zero-Knowledge Proofs under General Assumptions. *SIAM Journal on Computing*, Vol. 29, No. 1, pages 1–28, 1999.

[77] U. Feige and A. Shamir. Zero-Knowledge Proofs of Knowledge in Two Rounds. In *Crypto89*, Springer-Verlag Lecture Notes in Computer Science (Vol. 435), pages 526–544, 1990.

[78] U. Feige and A. Shamir. Witness Indistinguishability and Witness Hiding Protocols. In *22nd ACM Symposium on the Theory of Computing*, pages 416–426, 1990.

[79] W. Feller. *An Introduction to Probability Theory and Its Applications*. Wiley, New York, 1968.

[80] A. Fiat and A. Shamir. How to Prove Yourself: Practical Solution to Identification and Signature Problems. In *Crypto86*, Springer-Verlag Lecture Notes in Computer Science (Vol. 263), pages 186–189, 1987.

[81] M. Fischer, S. Micali, C. Rackoff, and D.K. Wittenberg. An Oblivious Transfer Protocol Equivalent to Factoring. Unpublished manuscript, 1986. (Preliminary versions were presented in *EuroCrypt84* and in the *NSF Workshop on Mathematical Theory of Security*, Endicott House (1985).)

[82] R. Fischlin and C.P. Schnorr. Stronger Security Proofs for RSA and Rabin Bits. In *EuroCrypt97*, Springer-Verlag Lecture Notes in Computer Science (Vol. 1233), pages 267–279, 1997.

[83] L. Fortnow. The Complexity of Perfect Zero-Knowledge. In *19th ACM Symposium on the Theory of Computing*, pages 204–209, 1987.

[84] A.M. Frieze, J. Håstad, R. Kannan, J.C. Lagarias, and A. Shamir. Reconstructing Truncated Integer Variables Satisfying Linear Congruences. *SIAM Journal on Computing*, Vol. 17, pages 262–280, 1988.

[85] O. Gaber and Z. Galil. Explicit Constructions of Linear Size Superconcentrators. *Journal of Computer and System Science*, Vol. 22, pages 407–420, 1981.

[86] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.

[87] P.S. Gemmell. An Introduction to Threshold Cryptography. In *CryptoBytes* (RSA Laboratories), Vol. 2, No. 3, 1997.

[88] R. Gennaro and L. Trevisan. Lower Bounds on the Efficiency of Generic Cryptographic Constructions. *ECCC*, TR00-022, May 2000.

[89] O. Goldreich. Two Remarks Concerning the GMR Signature Scheme. In *Crypto86*, Springer-Verlag Lecture Notes in Computer Science (Vol. 263), pages 104–110, 1987.

[90] O. Goldreich. Towards a Theory of Software Protection and Simulation by Oblivious RAMs. In *19th ACM Symposium on the Theory of Computing*, pages 182–194, 1987.

[91] O. Goldreich. *Foundation of Cryptography – Class Notes*. Preprint, spring 1989. (Superseded by the current book in conjunction with [92].)

[92] O. Goldreich. *Lecture Notes on Encryption, Signatures and Cryptographic Protocol*. (Extracts from [91]. Available from `http://theory.lcs.mit.edu/~oded/ln89.html`. Superseded by the combination of [99], [100], and [98].)

[93] O. Goldreich. A Note on Computational Indistinguishability. *Information Processing Letters*, Vol. 34, May, pages 277–281, 1990.

[94] O. Goldreich. A Uniform Complexity Treatment of Encryption and Zero-Knowledge. *Journal of Cryptology*, Vol. 6, No. 1, pages 21–53, 1993.

[95] O. Goldreich. *Foundation of Cryptography – Fragments of a Book*. February 1995. (Available from `http://theory.lcs.mit.edu/~oded/frag.html`. Superseded by the current book in conjunction with [99].)

[96] O. Goldreich. Notes on Levin's Theory of Average-Case Complexity. *ECCC*, TR97-058, December 1997.

[97] O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*. Algorithms and Combinatorics Series (Vol. 17), Springer-Verlag, 1999.

[98] O. Goldreich. *Secure Multi-Party Computation*. (In preparation, 1998. Working draft available from `http://theory.lcs.mit.edu/~oded/gmw.html`.)

[99] O. Goldreich. *Encryption Schemes – Fragments of a Chapter*. (December 1999. Available from `http://www.wisdom.weizmann.ac.il/~oded/foc-book.html`.)

[100] O. Goldreich. *Signature Schemes – Fragments of a Chapter*. (May 2000. Available from `http://www.wisdom.weizmann.ac.il/~oded/foc-book.html`.)

[101] O. Goldreich, S. Goldwasser, and S. Halevi. Collision-Free Hashing from Lattice Problems. *ECCC*, TR95-042, 1996.

[102] O. Goldreich, S. Goldwasser, and S. Micali. How to Construct Random Functions. *Journal of the ACM*, Vol. 33, No. 4, pages 792–807, 1986.

[103] O. Goldreich, S. Goldwasser, and S. Micali. On the Cryptographic Applications of Random Functions. In *Crypto84*, Springer-Verlag Lecture Notes in Computer Science (Vol. 263), pages 276–288, 1985.

[104] O. Goldreich, R. Impagliazzo, L.A. Levin, R. Venkatesan, and D. Zuckerman. Security Preserving Amplification of Hardness. In *31st IEEE Symposium on Foundations of Computer Science*, pages 318–326, 1990.

[105] O. Goldreich and A. Kahan. How to Construct Constant-Round Zero-Knowledge Proof Systems for NP. *Journal of Cryptology*, Vol. 9, No. 2, pages 167–189, 1996. (Preliminary versions date to 1988.)

[106] O. Goldreich and H. Krawczyk. On the Composition of Zero-Knowledge Proof Systems. *SIAM Journal on Computing*, Vol. 25, No. 1, February, pages 169–192, 1996.

[107] O. Goldreich and H. Krawczyk. On Sparse Pseudorandom Ensembles. *Random Structures and Algorithms*, Vol. 3, No. 2, pages 163–174, 1992.

[108] O. Goldreich, H. Krawcyzk, and M. Luby. On the Existence of Pseudorandom Generators. *SIAM Journal on Computing*, Vol. 22, No. 6, pages 1163–1175, 1993.

[109] O. Goldreich and E. Kushilevitz. A Perfect Zero-Knowledge Proof for a Decision Problem Equivalent to Discrete Logarithm. *Journal of Cryptology*, Vol. 6, No. 2, pages 97–116, 1993.

[110] O. Goldreich and L.A. Levin. Hard-Core Predicates for Any One-Way Function. In *21st ACM Symposium on the Theory of Computing*, pages 25–32, 1989.

[111] O. Goldreich and B. Meyer. Computational Indistinguishability – Algorithms vs. Circuits. *Theoretical Computer Science*, Vol. 191, pages 215–218, 1998.

[112] O. Goldreich, S. Micali, and A. Wigderson. Proofs that Yield Nothing but Their Validity or All Languages in NP Have Zero-Knowledge Proof Systems. *Journal of the ACM*, Vol. 38, No. 1, pages 691–729, 1991. (Preliminary version in *27th IEEE Symposium on Foundations of Computer Science*, 1986.)

[113] O. Goldreich, S. Micali, and A. Wigderson. How to Play Any Mental Game – A Completeness Theorem for Protocols with Honest Majority. In *19th ACM Symposium on the Theory of Computing*, pages 218–229, 1987.

[114] O. Goldreich, N. Nisan, and A. Wigderson. On Yao's XOR-Lemma. *ECCC*, TR95-050, 1995.

[115] O. Goldreich and Y. Oren. Definitions and Properties of Zero-Knowledge Proof Systems. *Journal of Cryptology*, Vol. 7, No. 1, pages 1–32, 1994.

[116] O. Goldreich and E. Petrank. Quantifying Knowledge Complexity. *Computational Complexity*, Vol. 8, pages 50–98, 1999.

[117] O. Goldreich, R. Rubinfeld, and M. Sudan. Learning Polynomials with Queries: The Highly Noisy Case. To appear in *SIAM Journal on Discrete Mathematics*.

[118] O. Goldreich, A. Sahai, and S. Vadhan. Honest-Verifier Statistical Zero-Knowledge Equals General Statistical Zero-Knowledge. In *30th ACM Symposium on the Theory of Computing*, pages 399–408, 1998.

[119] O. Goldreich and M. Sudan. Computational Indistinguishability: A Sample Hierarchy. *Journal of Computer and System Science*, Vol. 59, pages 253–269, 1999.

[120] O. Goldreich and S. Vadhan. Comparing Entropies in Statistical Zero-Knowledge with Applications to the Structure of SZK. In *14th IEEE Conference on Computational Complexity*, pages 54–73, 1999.

[121] S. Goldwasser and J. Kilian. Primality Testing Using Elliptic Curves. *Journal of the ACM*, Vol. 46, pages 450–472, 1999. (Preliminary version in *18th ACM Symposium on the Theory of Computing*, 1986.)

[122] S. Goldwasser and L.A. Levin. Fair Computation of General Functions in Presence of Immoral Majority. In *Crypto90*, Springer-Verlag Lecture Notes in Computer Science (Vol. 537), pages 77–93, 1991.

[123] S. Goldwasser and S. Micali. Probabilistic Encryption. *Journal of Computer and System Science*, Vol. 28, No. 2, pages 270–299, 1984. (Preliminary version in *14th ACM Symposium on the Theory of Computing*, 1982.)

[124] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on Computing*, Vol. 18, pages 186–208, 1989. (Preliminary version in *17th ACM Symposium on the Theory of Computing*, 1985.)

[125] S. Goldwasser, S. Micali, and R.L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM Journal on Computing*, April, pages 281–308, 1988.

[126] S. Goldwasser, S. Micali, and P. Tong. Why and How to Establish a Private Code in a Public Network. In *23rd IEEE Symposium on Foundations of Computer Science*, pages 134–144, 1982.

[127] S. Goldwasser, S. Micali, and A.C. Yao. Strong Signature Schemes. In *15th ACM Symposium on the Theory of Computing*, pages 431–439, 1983.

[128] S. Goldwasser and M. Sipser. Private Coins versus Public Coins in Interactive Proof Systems. *Advances in Computing Research: A Research Annual*, Vol. 5 (*Randomness and Computation*, S. Micali, ed.), pages 73–90, 1989.

[129] J. Håstad, R. Impagliazzo, L.A. Levin, and M. Luby. Construction of a Pseudorandom Generator from Any One-Way Function. *SIAM Journal on Computing*, Vol. 28, No. 4, pages 1364–1396, 1999. (Preliminary versions by Impagliazzo et al. in *21st ACM Symposium on the Theory of Computing* (1989) and Håstad in *22nd ACM Symposium on the Theory of Computing* (1990).)

[130] J. Håstad, A. Schrift, and A. Shamir. The Discrete Logarithm Modulo a Composite Hides $O(n)$ Bits. *Journal of Computer and System Science*, Vol. 47, pages 376–404, 1993.

[131] R. Impagliazzo and M. Luby. One-Way Functions are Essential for Complexity Based Cryptography. In *30th IEEE Symposium on Foundations of Computer Science*, pages 230–235, 1989.

[132] R. Impagliazzo and M. Naor. Efficient Cryptographic Schemes Provable as Secure as Subset Sum. *Journal of Cryptology*, Vol. 9, pages 199–216, 1996.

[133] R. Impagliazzo and S. Rudich. Limits on the Provable Consequences of One-Way Permutations. In *21st ACM Symposium on the Theory of Computing*, pages 44–61, 1989.

[134] R. Impagliazzo and A. Wigderson. P = BPP if E Requires Exponential Circuits:

Derandomizing the XOR Lemma. In *29th ACM Symposium on the Theory of Computing*, pages 220–229, 1997.

[135] R. Impagliazzo and D. Zuckerman. How to Recycle Random Bits. In *30th IEEE Symposium on Foundations of Computer Science*, pages 248–253, 1989.

[136] R. Impagliazzo and M. Yung. Direct Zero-Knowledge Computations. In *Crypto87*, Springer-Verlag Lecture Notes in Computer Science (Vol. 293), pages 40–51, 1987.

[137] A. Juels, M. Luby, and R. Ostrovsky. Security of Blind Digital Signatures. In *Crypto97*, Springer-Verlag Lecture Notes in Computer Science (Vol. 1294), pages 150–164, 1997.

[138] J. Justesen. A Class of Constructive Asymptotically Good Algebraic Codes. *IEEE Transactions on Information Theory*, Vol. 18, pages 652–656, 1972.

[139] N. Kahale. Eigenvalues and Expansion of Regular Graphs. *Journal of the ACM*, Vol. 42, No. 5, pages 1091–1106, 1995.

[140] J. Kahn, M. Saks, and C. Smyth. A Dual Version of Reimer's Inequality and a Proof of Rudich's Conjecture. In *15th IEEE Conference on Computational Complexity*, 2000.

[141] B.S. Kaliski. Elliptic Curves and Cryptography: A Pseudorandom Bit Generator and Other Tools. Ph.D. thesis, LCS, MIT, Cambridge, MA, 1988.

[142] J. Katz and M. Yung. Complete Characterization of Security Notions for Probabilistic Private-Key Encryption. In *32nd ACM Symposium on the Theory of Computing*, pages 245–254, 2000.

[143] J. Kilian. A Note on Efficient Zero-Knowledge Proofs and Arguments. In *24th ACM Symposium on the Theory of Computing*, pages 723–732, 1992.

[144] J. Kilian and E. Petrank. An Efficient Non-Interactive Zero-Knowledge Proof System for NP with General Assumptions. *Journal of Cryptology*, Vol. 11, pages 1–27, 1998.

[145] J.C. Lagarias and A.M. Odlyzko. Solving Low-Density Subset Sum Problems. *Journal of the ACM*, Vol. 32, pages 229–246, 1985.

[146] D. Lapidot and A. Shamir. Fully Parallelized Multi-prover Protocols for NEXP-Time. *Journal of Computer and System Science*, Vol. 54, No. 2, April, pages 215–220, 1997.

[147] A. Lempel. Cryptography in Transition. *Computing Surveys*, Vol. 11, No. 4, pages 285–303, December 1979.

[148] A.K. Lenstra, H.W. Lenstra, and L. Lovász. Factoring Polynomials with Rational Coefficients. *Mathematische Annalen*, Vol. 261, pages 515–534, 1982.

[149] L.A. Levin. Average Case Complete Problems. *SIAM Journal on Computing*, Vol. 15, pages 285–286, 1986.

[150] L.A. Levin. One-Way Function and Pseudorandom Generators. *Combinatorica*, Vol. 7, pages 357–363, 1987.

[151] L.A. Levin. Randomness and Non-determinism. *Journal of Symbolic Logic*, Vol. 58, No. 3, pages 1102–1103, 1993.

[152] M. Li and P. Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer-Verlag, 1993.

[153] J.H. van Lint. *Introduction to Coding Theory*. Graduate Texts in Mathematics (Vol. 88), Springer-Verlag, 1982.

[154] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan Graphs. *Combinatorica*, Vol. 8, pages 261–277, 1988.

[155] M. Luby. *Pseudorandomness and Cryptographic Applications*. Princeton University Press, 1996.

[156] M. Luby and C. Rackoff. How to Construct Pseudorandom Permutations from Pseudo-Random Functions. *SIAM Journal on Computing*, Vol. 17, pages 373–386, 1988.

[157] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic Methods for Interactive Proof Systems. *Journal of the ACM*, Vol. 39, No. 4, pages 859–868, 1992.

[158] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, FL, 1996.

[159] R.C. Merkle. Secure Communication over Insecure Channels. *CACM*, Vol. 21, No. 4, pages 294–299, 1978.

[160] R.C. Merkle. Protocols for Public Key Cryptosystems. In *Proceedings of the 1980 IEEE Symposium on Security and Privacy*, pages 122–134, 1980.

[161] R.C. Merkle. A Digital Signature Based on a Conventional Encryption Function. In *Crypto87*, Springer-Verlag Lecture Notes in Computer Science (Vol. 293), pages 369–378, 1987.

[162] R.C. Merkle. A Certified Digital Signature Scheme. In *Crypto89*, Springer-Verlag Lecture Notes in Computer Science (Vol. 435), pages 218–238, 1990.

[163] R.C. Merkle and M.E. Hellman. Hiding Information and Signatures in Trapdoor Knapsacks. *IEEE Transactions on Information Theory*, Vol. 24, pages 525–530, 1978.

[164] S. Micali, C. Rackoff, and B. Sloan. The Notion of Security for Probabilistic Cryptosystems. *SIAM Journal on Computing*, Vol. 17, pages 412–426, 1988.

[165] S. Micali and P. Rogaway. Secure Computation. In *Crypto91*, Springer-Verlag Lecture Notes in Computer Science (Vol. 576), pages 392–404, 1992.

[166] G.L. Miller. Riemann's Hypothesis and Tests for Primality. *Journal of Computer and System Science*, Vol. 13, pages 300–317, 1976.

[167] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[168] National Bureau of Standards. *Federal Information Processing Standards*, Publ. 46 (DES 1977).

[169] National Institute for Standards and Technology. Digital Signature Standard (DSS). *Federal Register*, Vol. 56, No. 169, August 1991.

[170] M. Naor. Bit Commitment Using Pseudorandom Generators. *Journal of Cryptology*, Vol. 4, pages 151–158, 1991.

[171] M. Naor, R. Ostrovsky, R. Venkatesan, and M. Yung. Zero-Knowledge Arguments for NP Can Be Based on General Assumptions. *Journal of Cryptology*, Vol. 11, pages 87–108, 1998.

[172] M. Naor and O. Reingold. Synthesizers and Their Application to the Parallel Construction of Pseudo-Random Functions. In *36th IEEE Symposium on Foundations of Computer Science*, pages 170–181, 1995.

[173] M. Naor and O. Reingold. On the Construction of Pseudo-Random Permutations: Luby-Rackoff Revisited. *Journal of Cryptology*, Vol. 12, No. 1, pages 29–66, 1999.

[174] M. Naor and O. Reingold. From Unpredictability to Indistinguishability: A Simple Construction of Pseudorandom Functions from MACs. In *Crypto98*, Springer-Verlag Lecture Notes in Computer Science (Vol. 1464), pages 267–282, 1998.

[175] M. Naor and M. Yung. Universal One-Way Hash Functions and Their Cryptographic Application. In *21st ACM Symposium on the Theory of Computing*, pages 33–43, 1989.

[176] M. Naor and M. Yung. Public-Key Cryptosystems Provably Secure Against Chosen Ciphertext Attacks. In *22nd ACM Symposium on the Theory of Computing*, pages 427–437, 1990.

[177] N. Nisan and D. Zuckerman. Randomness Is Linear in Space. *Journal of Computer and System Science*, Vol. 52, No. 1, pages 43–52, 1996.

[178] A.M. Odlyzko. The Future of Integer Factorization. *CryptoBytes* (RSA Laboratories), Vol. 1, No. 2, pages 5–12, 1995. (Available from `http://www.research.att.com/~amo`.)

[179] A.M. Odlyzko. Discrete Logarithms and Smooth Polynomials. In *Finite Fields: Theory, Applications and Algorithms*, G.L. Mullen and P. Shiue, eds., Contemporary Mathematics,

Vol. 168, American Mathematical Society, pages 269–278, 1994. (Available from `http://www.research.att.com/∼amo.`)

[180] T. Okamoto. On Relationships between Statistical Zero-Knowledge Proofs. In *28th ACM Symposium on the Theory of Computing*, pages 649–658, 1996.

[181] R. Ostrovsky and A. Wigderson. One-Way Functions Are Essential for Non-Trivial Zero-Knowledge. In *2nd Israel Symposium on Theory of Computing and Systems*, IEEE Comp. Soc. Press, pages 3–17, 1993.

[182] R. Ostrovsky and M. Yung. How to Withstand Mobile Virus Attacks. In *10th ACM Symposium on Principles of Distributed Computing*, pages 51–59, 1991.

[183] B. Pfitzmann. *Digital Signature Schemes* (*General Framework and Fail-Stop Signatures*). Springer-Verlag Lecture Notes in Computer Science (Vol. 1100), 1996.

[184] V. Pratt. Every Prime Has a Succinct Certificate. *SIAM Journal on Computing*, Vol. 4, pages 214–220, 1975.

[185] M.O. Rabin. Probabilistic Algorithm for Testing Primality. *Journal of Number Theory*, Vol. 12, pages 128–138, 1980.

[186] M.O. Rabin. Digitalized Signatures. In *Foundations of Secure Computation*, R.A. DeMillo et al., eds. Academic Press, 1977.

[187] M.O. Rabin. Digitalized Signatures and Public Key Functions as Intractable as Factoring. TR-212, LCS, MIT, Cambridge, MA, 1979.

[188] M.O. Rabin. How to Exchange Secrets by Oblivious Transfer. Tech. Memo. TR-81, Aiken Computation Laboratory, Harvard University, 1981.

[189] R. Richardson and J. Kilian. On the Concurrent Composition of Zero-Knowledge Proofs. In *EuroCrypt99*, Springer-Verlag Lecture Notes in Computer Science (Vol. 1592), pages 415–413, 1999.

[190] R. Raz. A Parallel Repetition Theorem. *SIAM Journal on Computing*, Vol. 27, No. 3, pages 763–803, 1998.

[191] R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public Key Cryptosystems. *CACM*, Vol. 21, pages 120–126, 1978.

[192] J. Rompel. One-Way Functions Are Necessary and Sufficient for Secure Signatures. In *22nd ACM Symposium on the Theory of Computing*, pages 387–394, 1990.

[193] A. Sahai. Non-Malleable Non-Interactive Zero Knowledge and Achieving Chosen-Ciphertext Security. In *40th IEEE Symposium on Foundations of Computer Science*, pages 543–553, 1999.

[194] A. Sahai and S. Vadhan. A Complete Promise Problem for Statistical Zero-Knowledge. In *38th IEEE Symposium on Foundations of Computer Science*, pages 448–457, 1997.

[195] C.P. Schnorr and H.H. Horner. Attacking the Chor-Rivest Cryptosystem by Improved Lattice Reduction. In *EuroCrypt95*, Springer-Verlag Lecture Notes in Computer Science (Vol. 921), pages 1–12, 1995.

[196] A. Shamir. How to Share a Secret. *CACM*, Vol. 22, pages 612–613, 1979.

[197] A. Shamir. A Polynomial-Time Algorithm for Breaking the Merkle-Hellman Cryptosystem. In *23rd IEEE Symposium on Foundations of Computer Science*, pages 145–152, 1982.

[198] A. Shamir. IP = PSPACE. *Journal of the ACM*, Vol. 39, No. 4, pages 869–877, 1992.

[199] A. Shamir, R.L. Rivest, and L. Adleman. Mental Poker. Report TM-125, LCS, MIT, Cambridge, MA, 1979.

[200] C.E. Shannon. Communication Theory of Secrecy Systems. *Bell Systems Technical Journal*, Vol. 28, pages 656–715, 1949.

[201] M. Sipser. A Complexity Theoretic Approach to Randomness. In *15th ACM Symposium on the Theory of Computing*, pages 330–335, 1983.

[202] M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing, Boston, MA, 1997.

[203] R. Solovay and V. Strassen. A Fast Monte-Carlo Test for Primality. *SIAM Journal on Computing*, Vol. 6, pages 84–85, 1977. (Addendum in *SIAM Journal on Computing*, Vol. 7, page 118, 1978.)

[204] M. Sudan. Decoding of Reed-Solomon Codes beyond the Error-Correction Bound. *Journal of Complexity*, Vol. 13, No. 1, pages 180–193, 1997.

[205] M. Tompa and H. Woll. Random Self-Reducibility and Zero-Knowledge Interactive Proofs of Possession of Information. In *28th IEEE Symposium on Foundations of Computer Science*, pages 472–482, 1987.

[206] S. Vadhan. A Study of Statistical Zero-Knowledge Proofs. Ph.D. thesis, Department of Mathematics, MIT, Cambridge, MA, 1999.

[207] A. Vardi. Algorithmic Complexity in Coding Theory and the Minimum Distance Problem. In *29th ACM Symposium on the Theory of Computing*, pages 92–108, 1997.

[208] U.V. Vazirani and V.V. Vazirani. Efficient and Secure Pseudo-Random Number Generation. In *25th IEEE Symposium on Foundations of Computer Science*, pages 458–463, 1984.

[209] M. Wegman and L. Carter. New Hash Functions and Their Use in Authentication and Set Equality. *Journal of Computer and System Science*, Vol. 22, pages 265–279, 1981.

[210] A.C. Yao. Theory and Application of Trapdoor Functions. In *23rd IEEE Symposium on Foundations of Computer Science*, pages 80–91, 1982.

[211] A.C. Yao. How to Generate and Exchange Secrets. In *27th IEEE Symposium on Foundations of Computer Science*, pages 162–167, 1986.

Index

Author Index

Subject Index