Computational Complexity:

A Conceptual Perspective

Oded Goldreich

Department of Computer Science and Applied Mathematics
Weizmann Institute of Science, Rehovot, Israel.

June 12, 2006

Preface. This manuscript contains preliminary versions of three related chapters and one appendix of the forthcoming book *Computational Complexity: A Conceptual Perspective.*

Chapter 7: The Bright Side of Hardness. We consider two conjectures that are related to $\mathcal{P} \neq \mathcal{NP}$. The first conjecture is that there are problems in $\mathcal{E}$ that are not solvable by (non-uniform) families of small (say polynomial-size) circuits, whereas the second conjecture is equivalent to the notion of *one-way functions*. Most of this chapter is devoted to converting these conjectures into tools that can be used for non-trivial dearndomizations of $\mathcal{BPP}$ and for a host of cryptographic applications.

Chapter 8: Pseudorandom Generators. The pivot of this chapter is the notion of *computational indistinguishablity* and corresponding notions of pseudorandomness. The definition of general-purpose pseudorandom generators (running in polynomial-time and withstanding any polynomial-time distinguisher) is presented as a special case of a general paradigm. The chapter also contains a presentation of other instatiations of the latter paradigm, including generators aimed at derandomizating complexity classes such as $\mathcal{BPP}$, generators withstanding space-bounded distinguishers, and some special-purpose generators.

Chapter 9: Probabilistic Proof Systems. This chapter provides an introduction to three types of probabilistic proof systems: *interactive proofs*, *zero-knowledge proofs*, and *probabilistic checkable proofs*. These 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 application of the proof system. The gain in allowing this untraditional relaxation is substantial, as they enable the construction of proof systems with properties that seem impossible to achieve via traditional proof systems.

Appendix C: On the Foundations of Modern Cryptography. The first part of this appendix augments the partial treatment of one-way functions, pseudorandom generators and zero-knowledge proofs, which is included in Chapters 7–9. Using these basic tools, the second part provides a treatment of basic cryptographic applications such as Encryption, Signatures, and General Cryptographic Protocols.

This material corresponds to the main material covered in the author's book [86], and superseeds it in almost all respects.

Chapter 7

The Bright Side of Hardness

So saying she donned her beautiful, glittering golden–Ambrosial sandals, which carry her flying like the wind over the vast land and sea; she grasped the redoubtable bronze-shod spear, so stout and sturdy and strong, wherewith she quells the ranks of heroes who have displeased her, the [bright-eyed] daughter of her mighty father.

Homer, Odyssey, 1:96–101

The existence of natural computational problems that are (or seem to be) infeasible to solve is usually perceived as bad news, because it means that we cannot do things we wish to do. But these bad news have a positive side, because hard problem can be "put to work" to our benefit, most notably in cryptography.

One key issue that arises whenever one tries to utilize hard problem is bridging the gap between "occasional" hardness (e.g., worst-case hardness or mild average-case hardness) and "typical" hardness (i.e., inapproximability). Much of the current chapter is devoted to this issue, which is known by the term *hardness amplification*.

Summary: We consider two conjectures that are related to $\mathcal{P} \neq \mathcal{NP}$. The first conjecture is that there are problems that are solvable in exponential-time (i.e., in $\mathcal{E}$) but are not solvable by (non-uniform) families of small (say polynomial-size) circuits. We show that this worst-case conjecture can be transformed into an average-case hardness result of the type that can be used towards derandomized $\mathcal{BPP}$ in a non-trivial way (see Section 8.4).

The second conjecture is that there are problems in NP (i.e., search problems in $\mathcal{PC}$) for which it is easy to generate (solved) instances that are typically hard to solve (for a party that did not generate these instances). This conjecture is captured in the formulation of *one-way functions*, which are functions that are easy to evaluate but hard to

invert (in an average-case sense). We show that functions that are hard to invert in a relatively mild average-case sense yield functions that are hard to invert almost everywhere, and that the latter yield predicates that are very hard to approximate (called *hard-core predicates*). The latter are useful for the construction of general-purpose pseudorandom generators (see Section 8.3) as well as for a host of cryptographic applications (see Appendix C).

The order of presentation of the two aforementioned conjectures and their consequences is actually reversed: We start (in Section 7.1) with the study of one-way function, and only later (in Section 7.2) turn to the study of problems in $\mathcal{E}$ that are hard for small circuits.

Teaching note: We list several reasons for preferring the aforementioned order of presentation. First, we mention the conceptual appeal of one-way functions and the fact that they have very practical applications. Second, hardness amplification in the context of one-way functions is technically simpler in comparison to the amplification of hardness in the context of $\mathcal{E}$. (In fact, Section 7.2 is the most technical text in this book.) Third, some of the techniques that are shared by both treatments seem easier to understand first in the context of one-way functions. Last, the current order facilitates the possibility of teaching hardness amplification only in one incarnation, where the context of one-way functions is recommended as the incarnation of choice (for the aforementioned reasons).
If you wish to teach hardness amplification and pseudorandomness in the two aforementioned incarnations, then we suggest following the order of the current text. That is, first teach hardness amplification in its two incarnations, and only next teach pseudorandomness in the corresponding incarnations.

Prerequisites: We assume a basic familiarity with elementary probability theory (see Appendix D.1) and randomized algorithms (see Section 6.1). In particular, standard conventions regarding random variables (presented in Appendix D.1.1) and various "laws of large numbers" (presented in Appendix D.1.2) will be extensively used.

7.1 One-Way Functions

Loosely speaking, one-way functions are functions that are easy to evaluate but hard (on the average) to invert. Thus, in assuming that one-way functions exist, we are postulating the existence of *efficient processes* (i.e., the computation of the function in the forward direction) *that are hard to reverse.* Analogous phenomena in daily life are known to us in abundance (e.g., the lighting of a match). Thus, the assumption that one-way functions exists is a complexity theoretic analogue of daily experience.

One-way functions can also be thought of as efficient ways for generating "puzzles" that are infeasible to solve; that is, the puzzle is a random image of the function and a solution is a corresponding preimage. Furthermore, the person generating the puzzle knows a solution to it and can efficiently verify the validity of (possibly other) solutions to the puzzle. In fact, as explained in Section 7.1.1, every mechanism for generating such puzzles can be converted to a one-way function.

The reader may note that when presented in terms of generating hard puzzles, one-way functions have a clear cryptographic flavor. Indeed, one-way functions are central to cryptography, but we shall not explore this aspect here (and rather refer the reader to Appendix C). Similarly, one-way functions are closely related to (general-purpose) pseudorandom generators, but this connection will be explored in Section 8.3. Instead, in the current section, we will focus on one-way functions *per se*.

Teaching note: While we recommend including a basic treatment of pseudorandomness within a course on complexity theory, we do not recommend doing so with respect to cryptography. The reason is that cryptography is far more complex than pseudorandomness (e.g., compare the definition of secure encryption to the the definition of pseudorandom generators). The extra complexity is due to conceptual richness, which is something good, except that some of these conceptual issues are central to cryptography but not to complexity theory. Thus, teaching cryptography in the context of a course on complexity theory is likely to either overload the course with material that is not central to complexity theory or cause a superficial and misleading treatment of cryptography. We are not sure as to which of these two possibilities is worse. Still, for the benefit of the interested reader, we have included an overview of the foundations of cryptography as an appendix to the main (see Appendix C).

7.1.1 The concept of one-way functions

Let us assume that $\mathcal{P} \neq \mathcal{NP}$ or even that $\mathcal{NP}$ is not contained in $\mathcal{BPP}$. Can we use this assumption to our benefit? Not really, because the assumption refers to the worst-case complexity of problems, and it may be that hard instances are hard to find. But then, it seems that if we cannot generate hard instances then we cannot benefit from their existence.

In Section 7.2 we shall see that worst-case hardness (of $\mathcal{NP}$ or even $\mathcal{E}$) can be transformed into average-case hardness of $\mathcal{E}$. Such a transformation is not known for $\mathcal{NP}$ itself, and in some applications (e.g., in cryptography) we wish the hard on the average problem to be in $\mathcal{NP}$. In this case, we need to assume that, for some problem in $\mathcal{NP}$, hard instances not only exist but are easy to generate. That is, $\mathcal{NP}$ is "hard on the average" with respect to a distribution that is efficiently sampleable. This assumption will be further discussed in Section 10.2.

However, for the aforementioned applications (e.g., in cryptography) this assumption does not seem to suffice either: we know how to utilize such "hard on the average" problems only when we can efficiently generate hard instances coupled with adequate solutions.[1] That is, we assume that, for some search problem in

[1]We wish to stress the difference between the two gaps discussed here. Our feeling is that worst-case hardness (*per se*) is far more difficult to utilize than average-case hardness that does not correspond to an efficient generation of "solved" instances.

$\mathcal{PC}$ (resp., decision problem in $\mathcal{NP}$), we can efficiently generate instance-solution pairs (resp., yes-instances coupled with corresponding NP-witnesses) such that the instance is hard to solve (of course, for a person that does not get the solution (resp., witness)).

Let us formulate the latter notion. Referring to Definition 2.3, we consider a relation R in $\mathcal{PC}$ (i.e., R is polynomially bounded and membership in R can be determined in polynomial-time), and assume that there exists a probabilistic polynomial-time algorithm G that satisfies the following two conditions:

1. On input 1^n, algorithm G always generates a pair in R such that the first element has length n. That is, $\mathsf{Pr}[G(1^n) \in R \cap (\{0,1\}^n \times \{0,1\}^*)] = 1$.

2. It is infeasible to find solutions to instances that are generated by G; that is, when only given the first element of $G(1^n)$, it is infeasible to find an adequate solution. Formally, denoting the first element of $G(1^n)$ by $G_1(1^n)$, for every probabilistic polynomial-time (solver) algorithm S, it holds that $\mathsf{Pr}[(G_1(1^n), S(G_1(1^n)) \in R] = \mu(n)$, where μ vanishes faster than any polynomial fraction (i.e., for every positive polynomial p and all sufficiently large n it is the case that $\mu(n) < 1/p(n)$).

We call G a generator of solved intractable instances for R. We will show that such a generator exists if and only if one-way functions exists, where one-way functions are functions that are easy to evaluate but hard (on the average) to invert. That is, a function $f:\{0,1\}^* \to \{0,1\}^*$ is called one-way if there is an efficient algorithm that on input x outputs $f(x)$, whereas any feasible algorithm that tries to find a preimage of $f(x)$ under f may succeed only with negligible probability (where the probability is taken uniformly over the choices of x and the algorithm's coin tosses). Associating feasible computations with probabilistic polynomial-time algorithms and negligible functions with functions that vanish faster than any polynomial fraction, we obtain the following definition.

Definition 7.1 (one-way functions): *A function $f:\{0,1\}^* \to \{0,1\}^*$ is called* one-way *if the following two conditions hold:*

1. Easy to evaluate: *There exist a polynomial-time algorithm A such that $A(x) = f(x)$ for every $x \in \{0,1\}^*$.*

2. Hard to invert: *For every probabilistic polynomial-time algorithm A', every polynomial p, and all sufficiently large n,*

$$\mathsf{Pr}_{x\in\{0,1\}^n}[A'(f(x),1^n) \in f^{-1}(f(x))] \; < \; \frac{1}{p(n)} \tag{7.1}$$

where the probability is taken uniformly over all the possible choices of $x \in \{0,1\}^n$ and all the possible outcomes of the internal coin tosses of algorithm A'.[2]

[2] An alternative formulation of Eq. (7.1) relies on the conventions in Appendix D.1.1. Specifically, letting U_n denote a random variable uniformly distributed in $\{0,1\}^n$, we may write Eq. (7.1) as $\mathsf{Pr}[A'(f(U_n),1^n) \in f^{-1}(f(U_n))] \; < \; 1/p(n)$, recalling that both occurrences of U_n refer to the same sample.

Algorithm A' is given the auxiliary input 1^n so as to allow it to run in time polynomial in the length of x, which is important in case f drastically shrinks its input (e.g., $|f(x)| = O(\log |x|)$). Typically (and, in fact, without loss of generality, see Exercise 7.1), f is length preserving, in which case the auxiliary input 1^n is redundant. Note that A' is not required to output a specific preimage of $f(x)$; any preimage (i.e., element in the set $f^{-1}(f(x))$) will do. (Indeed, in case f is 1-1, the string x is the only preimage of $f(x)$ under f; but in general there may be other preimages.) It is required that algorithm A' fails (to find a preimage) with overwhelming probability, when the probability is also taken over the input distribution. That is, f is "typically" hard to invert, not merely hard to invert in some ("rare") cases.

Proposition 7.2 *The following two conditions are equivalent:*

1. *There exists a generator of solved intractable instances for some $R \in \mathcal{NP}$.*
2. *There exist one-way functions.*

Proof Sketch: Suppose that G is such a generator of solved intractable instances for some $R \in \mathcal{NP}$, and suppose that on input 1^n it tosses $\ell(n)$ coins. For simplicity, we assume that $\ell(n) = n$, and consider the function $g(r) = G_1(1^{|r|}, r)$, where $G(1^n, r)$ denotes the output of G on input 1^n when using coins r (and G_1 is as in the foregoing discussion). Then g must be one-way, because an algorithm that inverts g on input $x = g(r)$ obtains r' such that $G_1(1^n, r') = x$ and $G(1^n, r')$ must be in R (which means that the second element of $G(1^n, r')$ is a solution to x). In case $\ell(n) \neq n$ (and assuming without loss of generality that $\ell(n) \geq n$), we define $g(r) = G_1(1^n, s)$ where n is the largest integer such that $\ell(n) \leq |r|$ and s is the $\ell(n)$-bit long prefix of r.

Suppose, on the other hand, that f is a one-way function. Then $R \stackrel{\text{def}}{=} \{(f(x), x) : x \in \{0,1\}^*\}$ is in $\mathcal{PC}$, and $G(1^n) = (f(r), r)$ for a uniformly selected $r \in \{0,1\}^n$ is a generator of solved intractable instances for R, because any solver of R is effectively inverting f on $f(U_n)$. $\square$

Comments. Several candidates one-way functions and variation on the basic definition appear in Appendix C.2.1. Here, for the sake of future discussions, we define a stronger version of one-way functions, which refers to the infeasibility of inverting the function by non-uniform circuits of polynomial-size. Here we use the form discussed in Footnote 2.

Definition 7.3 (one-way functions, non-uniformly hard): *A one-way function $f : \{0,1\}^* \to \{0,1\}^*$ is said to be* non-uniformly hard to invert *if for every family of polynomial-size circuits $\{C_n\}$, every polynomial p, and all sufficiently large n,*

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

We note that if a function is infeasible to invert by polynomial-size circuits then it is hard to invert by probabilistic polynomial-time algorithms; that is, non-uniformity (more than) compensates for lack of randomness. See Exercise 7.2.

7.1.2 Amplification of Weak One-Way Functions

In the forgoing discussion we have interpreted "hardness on the average" in a very strong sense. Specifically, we required that any feasible algorithm fails to solve the problem (e.g., invert the one-way function) *almost always* (i.e., *except with negligible probability*). This interpretation is indeed the one that is suitable for various applications. Still, a weaker interpretation of hardness on the average, which is also appealing, only requires that any feasible algorithm fails to solve the problem *often enough* (i.e., *with noticeable probability*). The main thrust of the current section is showing that the mild form of hardness on the average can be transformed into the strong form discussed in Section 7.1.1. Let us first define the mild form of hardness on the average, using the framework of one-way functions. Specifically, we define weak one-way functions.

Definition 7.4 (weak one-way functions): *A function* $f:\{0,1\}^* \to \{0,1\}^*$ *is called* weakly one-way *if the following two conditions hold:*

1. Easy to evaluate: *As in Definition 7.1.*
2. Weakly hard to invert: *There exists a positive polynomial* p *such that for every probabilistic polynomial-time algorithm* A' *and all sufficiently large* n*,*

$$\mathsf{Pr}_{x\in\{0,1\}^n}[A'(f(x),1^n) \notin f^{-1}(f(x))] \;>\; \frac{1}{p(n)} \tag{7.2}$$

where the probability is taken uniformly over all the possible choices of $x \in \{0,1\}^n$ *and all the possible outcomes of the internal coin tosses of algorithm* A'*. In such a case, we say that* f *is* $1/p$-one-way.

Here we require that algorithm A' fails (to find an f-preimage for a random f-image) with noticeable probability, rather than with overwhelmingly high probability (as in Definition 7.1). For clarity, we will occasionally refer to one-way functions as in Definition 7.1 by the term strong one-way functions.

We note that, assuming that one-way functions exist at all, there exists weak one-way functions that are not strongly one-way (see Exercise 7.3). Still, any weak one-way function can be transformed into a strong one-way function. This is indeed the main result of the current section.

Theorem 7.5 (amplification of one-way functions): *The existence of weak one-way functions implies the existence of strong one-way functions.*

Proof Sketch: The construction itself is straightforward. We just parse the argument to the new function into sufficiently many blocks, and apply the weak one-way function on the individual blocks. That is, suppose that f is $1/p$-one-way, for some polynomial p, and consider the following function

$$\begin{aligned} F(x_1,...,x_t) &= (f(x_1),...,f(x_t)) \\ &\quad \text{where } t \stackrel{\text{def}}{=} n\cdot p(n) \text{ and } x_1,...,x_t \in \{0,1\}^n. \end{aligned} \tag{7.3}$$

(Indeed F should be extended to strings of length outside $\{n^2 \cdot p(n) : n \in \mathbb{N}\}$ and this extension must be hard to invert on all preimage lengths.)[3]

We warn that the hardness of inverting the resulting function F is not established by mere "combinatorics" (i.e., considering the relative volume of S^t in $(\{0,1\}^n)^t$, for $S \subset \{0,1\}^n$, where S represents the set of "easy to invert" f-images). Specifically, one may *not* assume that the potential inverting algorithm works independently on each block. Indeed this assumption seems reasonable, but we should not make assumptions regarding the class of all efficient algorithms unless we can actually prove that nothing is lost by such assumptions.

The hardness of inverting the resulting function F is proved via a so called "reducibility argument" (which is used to prove all conditional results in the area). By a reducibility argument we actually mean a reduction, but one that is analyzed with respect to average case complexity. Specifically, we show that any algorithm that inverts the resulting function F with non-negligible success probability can be used to construct an algorithm that inverts the original function f with success probability that violates the hypothesis (regarding f). In other words, we reduce the task of "strongly inverting" f (i.e., violating its weak one-wayness) to the task of "weakly inverting" F (i.e., violating its strong one-wayness). In particular, on input $y = f(x)$, the reduction invokes the F-inverter (polynomially) many times, each time feeding it with a sequence of random f-images that contains y at a random location. (Indeed such a sequence corresponds to a random image of F.) Details follow.

Suppose towards the contradiction that F is not strongly one-way; that is, there exists a probabilistic polynomial-time algorithm B' and a polynomial $q(\cdot)$ so that for infinitely many m's

$$\mathsf{Pr}[B'(F(U_m)) \in F^{-1}(F(U_m))] > \frac{1}{q(m)} \tag{7.4}$$

Focusing on such a generic m and assuming (see Footnote 3) that $m = n^2 p(n)$, we present the following probabilistic polynomial-time algorithm, A', for inverting f. On input y and 1^n (where supposedly $y = f(x)$ for some $x \in \{0,1\}^n$), algorithm A' proceeds by applying the following probabilistic procedure, denoted I, on input y for $t'(n)$ times, where $t'(\cdot)$ is a polynomial that depends on the polynomials p and q (specifically, we set $t'(n) \stackrel{\text{def}}{=} 2n^2 \cdot p(n) \cdot q(n^2 p(n))$).

Procedure I (on input y and 1^n):

For $i = 1$ to $t(n) \stackrel{\text{def}}{=} n \cdot p(n)$ do begin

(1) Select uniformly and independently a sequence of strings $x_1, ..., x_{t(n)} \in \{0,1\}^n$.

(2) Compute $(z_1, ..., z_{t(n)}) \leftarrow B'(f(x_1), ..., f(x_{i-1}), y, f(x_{i+1}), ..., f(x_{t(n)}))$
(Note that y is placed in the i^{th} position instead of $f(x_i)$.)

(3) If $f(z_i) = y$ then halt and output z_i.
(This is considered a *success*).

[3] One simple extension is to define $F(x)$ to equal $F(x_1, ..., x_{n \cdot p(n)})$, where n is the largest integer satisfying $n^2 p(n) \leq |x|$ and x_i is the i^{th} consecutive n-bit long string in x (i.e., $x = x_1 \cdot x_{n \cdot p(n)} x'$, where $x_1, ..., x_{n \cdot p(n)} \in \{0,1\}^n$).

`end`

Using Eq. (7.4), we now present a lower bound on the success probability of algorithm A', deriving a contradiction to the theorem's hypothesis. To this end we define a set, denoted S_n, that contains all n-bit strings on which the procedure I succeeds with probability greater than $n/t'(n)$. (The probability is taken only over the coin tosses of procedure I). Namely,

$$S_n \stackrel{\text{def}}{=} \left\{x \in \{0,1\}^n : \mathsf{Pr}[I(f(x)) \in f^{-1}(f(x))] > \frac{n}{t'(n)}\right\}$$

In the next two claims we shall show that S_n contains all but at most a $1/2p(n)$ fraction of the strings of length n, and that for each string $x \in S_n$ algorithm A' inverts f on $f(x)$ with probability exponentially close to 1. It will follow that A' inverts f on $f(U_n)$ with probability greater than $1-(1/p(n))$, in contradiction to the theorem's hypothesis.

Claim 7.5.1: For every $x \in S_n$

$$\mathsf{Pr}\left[A'(f(x)) \in f^{-1}(f(x))\right] > 1 - 2^{-n}$$

This claim follows directly from the definitions of S_n and A'.

Claim 7.5.2:

$$|S_n| > \left(1 - \frac{1}{2p(n)}\right) \cdot 2^n$$

The rest of the proof is devoted to establishing this claim, and indeed combining Claims 7.5.1 and 7.5.2, the theorem follows.

The key observation is that, for every $i \in [t(n)]$ and every $x_i \in \{0,1\}^n \setminus S_n$, it holds that

$$\begin{aligned} &\mathsf{Pr}\left[B'(F(U_{n^2p(n)})) \in F^{-1}(F(U_{n^2p(n)}))\,\middle|\, U_n^{(i)} = x_i\right] \\ &\quad \leq \mathsf{Pr}\left[I(f(x_i)) \in f^{-1}(f(x_i))\right] \leq \frac{n}{t'(n)} \end{aligned}$$

where $U_n^{(1)}, ..., U_n^{(n \cdot p(n))}$ denote the n-bit long blocks in the random variable $U_{n^2p(n)}$. It follows that

$$\begin{aligned} \xi &\stackrel{\text{def}}{=} \mathsf{Pr}\left[B'(F(U_{n^2p(n)})) \in F^{-1}(F(U_{n^2p(n)})) \wedge \left(\exists i \text{ s.t. } U_n^{(i)} \in \{0,1\}^n \setminus S_n\right)\right] \\ &\leq \sum_{i=1}^{t(n)} \mathsf{Pr}\left[B'(F(U_{n^2p(n)})) \in F^{-1}(F(U_{n^2p(n)})) \wedge U_n^{(i)} \in \{0,1\}^n \setminus S_n\right] \\ &\leq t(n) \cdot \frac{n}{t'(n)} \,. \end{aligned}$$

On the other hand, using Eq. (7.4), we have

$$\begin{aligned} \xi &\geq \mathsf{Pr}\left[B'(F(U_{n^2p(n)})) \in F^{-1}(F(U_{n^2p(n)}))\right] - \mathsf{Pr}\left[(\forall i)\, U_n^{(i)} \in S_n\right] \\ &\geq \frac{1}{q(n^2p(n))} - \mathsf{Pr}\left[U_n \in S_n\right]^{t(n)} . \end{aligned}$$

Using $t'(n) = 2n^2 \cdot p(n) \cdot q(n^2 p(n))$ and $t(n) = n \cdot p(n)$, we get $\mathsf{Pr}[U_n \in S_n] > (1/2q(n^2p(n)))^{1/(n \cdot p(n))}$, which implies $\mathsf{Pr}[U_n \in S_n] > 1 - (1/2p(n))$ for sufficiently large n. Claim 7.5.2 follows, and so does the theorem. $\square$

Digest. Let us recall the structure of the proof of Theorem 7.5. Given a weak one-way function f, we first constructed a polynomial-time computable function F with the intention of later proving that F is strongly one-way. To prove that F is strongly one-way, we used a *reducibility argument*. The argument transforms efficient algorithms that supposedly contradict the strong one-wayness of F into efficient algorithms that contradict the hypothesis that f is weakly one-way. Hence F must be strongly one-way. We stress that our algorithmic transformation, which is in fact a randomized Cook reduction, makes no implicit or explicit assumptions about the structure of the prospective algorithms for inverting F. Such assumptions (e.g., the "natural" assumption that the inverter of F works independently on each block) cannot be justified (at least not at our current state of understanding of the nature of efficient computations).

We use the term a *reducibility argument*, rather than just saying a reduction so as to emphasize that we do *not* refer here to standard (worst-case complexity) reductions. Let us clarify the distinction: In both cases we refer to *reducing* the task of solving one problem to the task of solving another problem; that is, we use a procedure solving the second task in order to construct a procedure that solves the first task. However, in standard reductions one assumes that the second task has a perfect procedure solving it on all instances (i.e., on the worst-case), and constructs such a procedure for the first task. Thus, the reduction may invoke the given procedure (for the second task) on very "non-typical" instances. This cannot be allowed in our reducibility arguments. Here, we are given a procedure that solves the second task *with certain probability with respect to a certain distribution.* Thus, in employing a reducibility argument, we cannot invoke this procedure on any instance. Instead, we must consider the probability distribution, on instances of the second task, induced by our reduction. In our case (as in many cases) the latter distribution equals the distribution to which the hypothesis (regarding solvability of the second task) refers, but other cases may be handled too (e.g., these distributions may be "sufficiently close" for the specific purpose). In any case, a careful analysis of the distribution induced by the reducibility argument is due. (Indeed, the same issue arises in the context of reductions among "distributional problems" considered in Section 10.2.)

An information theoretic analogue. Theorem 7.5 has a natural information theoretic (or "probabilistic") analogue that asserts that repeating an experiment that has a noticeable failure probability, sufficiently many times yields some failure with very high probability. The reader is probably convinced at this stage that the proof of Theorem 7.5 is much more complex than the proof of the information theoretic analogue. In the information theoretic context the repeated events are independent by definition, whereas in the computational context no such independence (which corresponds to the naive argument discussed at the beginning of the proof of Theorem 7.5) can be guaranteed. Another indication to the difference between the two settings follows. In the information theoretic setting the probability that none of the failure events occurs decreases exponentially in the number of repetitions. In contrast, in the computational setting we can only reach an unspecified negligible bound on the inverting probabilities of polynomial-time algorithms. Furthermore, it may be the case that F constructed in the proof of Theorem 7.5 can be efficiently inverted on $F(U_{n^2 p(n)})$ with success probability that is sub-exponentially decreasing (e.g., with probability $2^{-(\log_2 n)^3}$), whereas the analogous information theoretic bound is exponentially decreasing (i.e., e^{-n}).

7.1.3 Hard-Core Predicates

One-way functions *per se* suffice for one central application: the construction of secure signature schemes (see Appendix C.6). For other applications, one relies not merely on the infeasibility of fully recovering the preimage of a one-way function, but rather on the infeasibility of meaningfully guessing bits in the preimage. The latter notion is captured by the definition of a hard-core predicate.

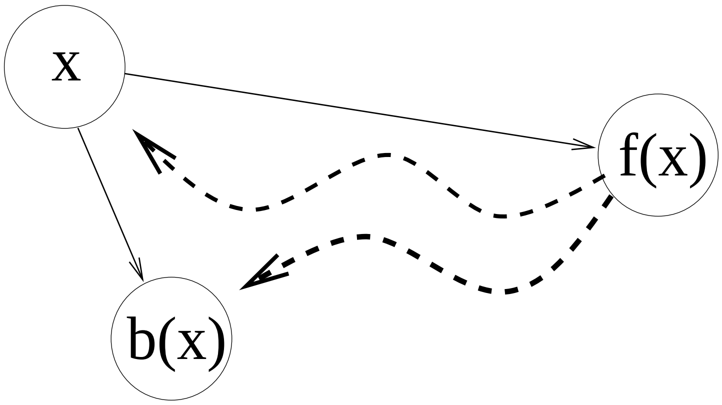
Recall that saying that a function f is one-way means that given a typical y (in the range of f) it is infeasible to find a preimage of y under f. This does not mean that it is infeasible to find partial information about the preimage(s) of y under f. Specifically, it may be easy to retrieve half of the bits of the preimage (e.g., given a one-way function f consider the function f' defined by $f'(x,r) \stackrel{\text{def}}{=} (f(x),r)$, for every $|x|=|r|$). We note that hiding partial information (about the function's preimage) plays an important role in more advanced constructs (e.g., pseudorandom generators and secure encryption). With this motivation in mind, we will show that essentially any one-way function hides specific partial information about its preimage, where this partial information is easy to compute from the preimage itself. This partial information can be considered as a "hard core" of the difficulty of inverting f. Loosely speaking, a *polynomial-time computable* (Boolean) predicate b, is called a hard-core of a function f if no feasible algorithm, given $f(x)$, can guess $b(x)$ with success probability that is non-negligibly better than one half.

Definition 7.6 (hard-core predicates): *A polynomial-time computable predicate $b : \{0,1\}^* \to \{0,1\}$ is called a* hard-core *of a function f if for every probabilistic polynomial-time algorithm A', every positive polynomial $p(\cdot)$, and all sufficiently large n's*

$$\mathsf{Pr}\left[A'(f(x)) = b(x)\right] < \frac{1}{2} + \frac{1}{p(n)}$$

where the probability is taken uniformly over all the possible choices of $x \in \{0,1\}^n$ and all the possible outcomes of the internal coin tosses of algorithm A'.

Note that for every $b : \{0,1\}^* \to \{0,1\}$ and $f : \{0,1\}^* \to \{0,1\}^*$, there exist obvious algorithms that guess $b(x)$ from $f(x)$ with success probability at least one half (e.g., the algorithm that, obliviously of its input, outputs a uniformly chosen bit). Also, if b is a hard-core predicate (of any function) then it follows that b is almost unbiased (i.e., for a uniformly chosen x, the difference $|\mathsf{Pr}[b(x)=0] - \mathsf{Pr}[b(x)=1]|$ must be



The solid arrows depict easily computable transformation while the dashed arrows depict infeasible transformations.

Figure 7.1: The hard-core of a one-way function – an illustration.

a negligible function in n). Finally, if b is a hard-core of a 1-1 function f that is polynomial-time computable then f must be a one-way function. In general, the interesting case is when being a hard-core is a computational phenomenon rather an information theoretic one (which is due to "information loss" of f).

Theorem 7.7 (a generic hard-core predicate): *For any one-way function f, the inner-product mod 2 of x and r, denoted $b(x,r)$, is a hard-core of $f'(x,r) = (f(x),r)$.*

In other words, given $f(x)$ and a random subset $S \subseteq [|x|]$, it is infeasible to guess $\oplus_{i\in S} x_i$ significantly better than with probability 1/2, where $x = x_1 \cdots x_n$ is uniformly distributed in $\{0,1\}^n$.

Proof Sketch: The proof is by a so-called "reducibility argument" (see Section 7.1.2). Specifically, we reduce the task of inverting f to the task of predicting the hard-core of f', while making sure that the reduction (when applied to input distributed as in the inverting task) generates a distribution as in the definition of the predicting task. Thus, a contradiction to the claim that b is a hard-core of f' yields a contradiction to the hypothesis that f is hard to invert. We stress that this argument is far more complex than analyzing the corresponding "probabilistic" situation (i.e., the distribution of the inner-product mod 2 of X and r, conditioned on a uniformly selected $r \in \{0,1\}^n$, where X is a random variable with super-logarithmic min-entropy, which represents the "effective" knowledge of x, when given $f(x)$).[4]

[4] The min-entropy of X is defined as $\min_v\{\log_2(1/\Pr[X = v])\}$; that is, if X has min-entropy m then $\max_v\{\Pr[X = v]\} = 2^{-m}$. The Leftover Hashing Lemma (see Appendix D.2) implies that,

Our starting point is a probabilistic polynomial-time algorithm B that satisfies, for some polynomial p and infinitely many n's, $\Pr[B(f(X_n), U_n) = b(X_n, U_n)] > (1/2) + (1/p(n))$, where X_n and U_n are uniformly and independently distributed over $\{0,1\}^n$. Using a simple averaging argument, we focus on a $\varepsilon \stackrel{\text{def}}{=} 1/2p(n)$ fraction of the x's for which $\Pr[B(f(x), U_n) = b(x, U_n)] > (1/2) + \varepsilon$ holds. We will show how to use B in order to invert f, on input $f(x)$, provided that x is in the good set (which has density ε).

As a warm-up, suppose for a moment that, for the aforementioned x's, algorithm B succeeds with probability $p > \frac{3}{4} + 1/\text{poly}(|x|)$ rather than at least $\frac{1}{2} + 1/\text{poly}(|x|)$. In this case, retrieving x from $f(x)$ is quite easy: To retrieve the i^{th} bit of x, denoted x_i, we randomly select $r \in \{0,1\}^{|x|}$, and obtain $B(f(x), r)$ and $B(f(x), r \oplus e^i)$, where $e^i = 0^{i-1}10^{|x|-i}$ and $v \oplus u$ denotes the addition mod 2 of the binary vectors v and u. A key observation underlying the foregoing scheme as well as the rest of the proof is that $b(x, r \oplus s) = b(x, r) \oplus b(x, s)$, which can be readily verified by writing $b(x, y) = \sum_{i=1}^{n} x_i y_i \bmod 2$ and noting that addition modulo 2 of bits corresponds to their XOR. Indeed, note that if both $B(f(x), r) = b(x, r)$ and $B(f(x), r \oplus e^i) = b(x, r \oplus e^i)$ hold, then $B(f(x), r) \oplus B(f(x), r \oplus e^i)$ equals $b(x, r) \oplus b(x, r \oplus e^i) = b(x, e^i) = x_i$. The probability that both $B(f(x), r) = b(x, r)$ and $B(f(x), r \oplus e^i) = b(x, r \oplus e^i)$ hold, for a random r, is at least $1 - 2 \cdot (1 - p) > \frac{1}{2} + \frac{1}{\text{poly}(|x|)}$. Hence, repeating the above procedure sufficiently many times (using independent random choices of such r's) and ruling by majority, we retrieve x_i with very high probability. Similarly, we can retrieve all the bits of x, and hence invert f on $f(x)$. However, the entire analysis was conducted under (the unjustifiable) assumption that $p > \frac{3}{4} + \frac{1}{\text{poly}(|x|)}$, whereas we only know that $p > \frac{1}{2} + \varepsilon$ for $\varepsilon = 1/\text{poly}(|x|)$.

The problem with the foregoing procedure is that it doubles the original error probability of algorithm B on inputs of the form $(f(x), \cdot)$. Under the unrealistic (foregoing) assumption that B's average error on such inputs is non-negligibly smaller than $\frac{1}{4}$, the "error-doubling" phenomenon raises no problems. However, in general (and even in the special case where B's error is exactly $\frac{1}{4}$) the above procedure is unlikely to invert f. Note that the *average* error probability of B (for a fixed $f(x)$, when the average is taken over a random r) can not be decreased by repeating B several times (e.g., for every x, it may be that B always answer correctly on three quarters of the pairs $(f(x), r)$, and always err on the remaining quarter). What is required is an *alternative way of using* the algorithm B, a way that does not double the original error probability of B.

The key idea is generating the r's in a way that allows applying algorithm B only once per each r (and i), instead of twice. Specifically, we will invoke B on $(f(x), r \oplus e^i)$ in order to obtain a "guess" for $b(x, r \oplus e^i)$, and obtain $b(x, r)$ in a different way (which does not involve using B). The good news is that the error probability is no longer doubled, since we only use B to get a "guess" of $b(x, r \oplus e^i)$. The bad news is that we still need to know $b(x, r)$, and it is not clear how we can know $b(x, r)$ without applying B. The answer is that we can

in this case, $\Pr[b(X, U_n) = 1 | U_n] = \frac{1}{2} \pm 2^{-\Omega(m)}$, where U_n denotes the uniform distribution over $\{0,1\}^n$, and $b(u, v)$ denotes the inner-product mod 2 of u and v.

guess $b(x,r)$ by ourselves. This is fine if we only need to guess $b(x,r)$ for one r (or logarithmically in $|x|$ many r's), but the problem is that we need to know (and hence guess) the value of $b(x,r)$ for polynomially many r's. The obvious way of guessing these $b(x,r)$'s yields an exponentially small success probability. Instead, we generate these polynomially many r's such that, on one hand they are "sufficiently random" whereas, on the other hand, we can guess all the $b(x,r)$'s with noticeable success probability.[5] Specifically, generating the r's in a specific *pairwise independent* manner will satisfy both (conflicting) requirements. We stress that in case we are successful (in our guesses for all the $b(x,r)$'s), we can retrieve x with high probability. Hence, we retrieve x with noticeable probability.

A word about the way in which the pairwise independent r's are generated (and the corresponding $b(x,r)$'s are guessed) is indeed in place. To generate $m = \text{poly}(|x|)$ many r's, we uniformly (and independently) select $\ell \stackrel{\text{def}}{=} \log_2(m+1)$ strings in $\{0,1\}^{|x|}$. Let us denote these strings by $s^1, ..., s^\ell$. We then guess $b(x,s^1)$ through $b(x,s^\ell)$. Let us denote these guesses, which are uniformly (and independently) chosen in $\{0,1\}$, by σ^1 through σ^ℓ. Hence, the probability that all our guesses for the $b(x,s^i)$'s are correct is $2^{-\ell} = \frac{1}{\text{poly}(|x|)}$. The different r's correspond to the different *non-empty* subsets of $\{1,2,...,\ell\}$. Specifically, for every such subset J, we let $r^J \stackrel{\text{def}}{=} \oplus_{j\in J} s^j$. The reader can easily verify that the r^J's are pairwise independent and each is uniformly distributed in $\{0,1\}^{|x|}$; see Exercise 7.5. The key observation is that $b(x,r^J) = b(x, \oplus_{j\in J} s^j) = \oplus_{j\in J} b(x,s^j)$. Hence, our guess for $b(x,r^J)$ is $\oplus_{j\in J}\sigma^j$, and with noticeable probability all our guesses are correct. Wrapping-up everything, we obtain the following procedure, where $\varepsilon = 1/\text{poly}(n)$ represents a lower-bound on the advantage of B in guessing $b(x,\cdot)$ for an ε fraction of the x's.

Inverting procedure (on input $y = f(x)$ and parameters n and ε):
Set $\ell = \log_2(n/\varepsilon^2) + O(1)$.
(1) Select uniformly and independently $s^1, ..., s^\ell \in \{0,1\}^n$.
Select uniformly and independently $\sigma^1, ..., \sigma^\ell \in \{0,1\}$.
(2) For every non-empty $J \subseteq [\ell]$, compute $r^J = \oplus_{j\in J} s^j$ and $\rho^J = \oplus_{j\in J}\sigma^j$.
(3) For $i = 1, ..., n$ determine the bit z_i according to the majority vote of the $(2^\ell - 1)$-long sequence of bits $(\rho^J \oplus B(f(x), r^J \oplus e^i))_{\emptyset\neq J\subseteq[\ell]}$.
(4) Output $z_1 \cdots z_n$.

Note that the "voting scheme" employed in Step 3 uses pairwise independent samples (i.e., the r^J's), but works essentially as well as it would have worked with independent samples (i.e., the independent r's).[6] That is, for every i and J, it holds that $\text{Pr}_{s^1,...,s^\ell}[B(f(x), r^J \oplus e^i) = b(x, r^J \oplus e^i)] > (1/2) + \varepsilon$, where $r^J = \oplus_{j\in J} s^j$,

[5] Alternatively, we can try all polynomially many possible guesses. In such a case, we shall output a list of candidates that, with high probability, contains x.

[6] Our focus here is on the accuracy of the approximation obtained by the sample, and not so much on the error probability. We wish to approximate $\text{Pr}[b(x,r) \oplus B(f(x), r\oplus e^i) = 1]$ up to an additive term of ε, because such an approximation allows to correctly determine $b(x,e^i)$. A pairwise independent sample of $O(t/\varepsilon^2)$ points allows for an approximation of a value in $[0,1]$ up to an additive term of ε with error probability $1/t$, whereas a totally random sample of the same size yields error probability $\exp(-t)$. Since we can afford setting $t = \text{poly}(n)$ and having error

and (for every fixed i) the events corresponding to different J's are pairwise independent. It follows that *if for every $j \in [\ell]$ it holds that $\sigma^j = b(x, s^j)$*, then for every i and J we have

$$\begin{aligned} & \mathsf{Pr}_{s^1,\ldots,s^\ell}[\rho^J \oplus B(f(x), r^J \oplus e^i) = b(x, e^i)] \\ & \quad = \mathsf{Pr}_{s^1,\ldots,s^\ell}[B(f(x), r^J \oplus e^i) = b(x, r^J \oplus e^i)] \; > \; \frac{1}{2} + \varepsilon \end{aligned} \tag{7.5}$$

where the equality is due to $\rho^J = \oplus_{j \in J} \sigma^j = b(x, r^J) = b(x, r^J \oplus e^i) \oplus b(x, e^i)$. Note that Eq. (7.5) refers to the correctness of a single vote for $b(x, e^i)$. Using $m = O(n/\varepsilon^2)$ and noting that these (Boolean) votes are pairwise independent, we infer that the probability that the majority of these votes is wrong is upper-bounded by $1/2n$. Using a union bound on all i's, we infer that with probability at least $1/2$, all majority votes are correct and thus x is retrieved correctly. Recall that the foregoing is conditioned on $\sigma^j = b(x, s^j)$ for every $j \in [\ell]$, which in turn holds with probability $2^{-\ell} = (m+1)^{-1} = \Omega(\varepsilon^2/n) = 1/\text{poly}(n)$, Thus, x is retrieved correctly with probability $1/\text{poly}(n)$, and the theorem follows. □

Digest. Looking at the proof of Theorem 7.7, we note that it actually refers to a black-box $B_x(\cdot)$ that approximates $b(x, \cdot)$; specifically, in the case of Theorem 7.7 we used $B_x(r) \stackrel{\text{def}}{=} B(f(x), r)$. In particular, the proof does not use the fact that we can verify the correctness of the preimage recovered by the described process. Furthermore, using the alternative procedure outlined in Footnote 5, the proof extends to establish *the existence of a* $\text{poly}(n/\varepsilon)$*-time oracle machine that, for every $x \in \{0,1\}^n$, given oracle access to any $B_x : \{0,1\}^n \to \{0,1\}$ satisfying*

$$\mathsf{Pr}_{r \in \{0,1\}^n}[B_x(r) = b(x, r)] \geq \frac{1}{2} + \varepsilon \tag{7.6}$$

outputs, with probability at least $1/2$*, a list of n-bit strings that includes x.* Noting that x is merely a string for which Eq. (7.6) holds, and that the procedure may get n and ε as inputs, we derive

Theorem 7.8 (Theorem 7.7, revisited): *There exists a probabilistic oracle machine that, given parameters n, ε and oracle access to any function $B : \{0,1\}^n \to \{0,1\}$, halts after* $\text{poly}(n/\varepsilon)$ *steps and with probability at least* $1/2$ *outputs a list of all strings $x \in \{0,1\}^n$ that satisfy*

$$\mathsf{Pr}_{r \in \{0,1\}^n}[B(r) = b(x, r)] \geq \frac{1}{2} + \varepsilon,$$

where $b(x, r)$ denotes the inner-product mod 2 of x and r.

This machine can be modified such that, with high probability, its output list does not include any string x such that $\mathsf{Pr}_{r \in \{0,1\}^n}[B(r) = b(x, r)] < \frac{1}{2} + \frac{\varepsilon}{2}$. Theorem 7.8

probability $1/2n$, the difference in the error probability between the two approximation schemes is not important here. For a wider perspective see Appendix D.1.2 and D.3.

can be viewed as a list decoding[7] procedure for the Hadamard Code, where the Hadamard encoding of a string $x \in \{0,1\}^n$ is the 2^n-bit long string containing $b(x,r)$ for every $r \in \{0,1\}^n$.

Applications. Hard-core predicates play a central role in the construction of general-purpose pseudorandom generators (see Section 8.3), commitment schemes and zero-knowledge proofs (see Sections 9.2.2 and C.4.3), and encryption schemes (see Appendix C.5).

7.2 Hard Predicates in E

We start again with the assumption $\mathcal{P} \neq \mathcal{NP}$. In fact, we consider the seemingly stronger assumption by which $\mathcal{NP}$ cannot be solved by (non-uniform) families of polynomial-size circuits; that is, $\mathcal{NP}$ is not contained in $\mathcal{P}/\text{poly}$ (even not infinitely often). Our goal is to transform this worst-case assumption into an average-case condition, which is useful for our applications. Since the transformation will not yield a problem in $\mathcal{NP}$ but rather one in $\mathcal{E}$, we might as well take the weaker assumption (see Exercise 7.8). That is, our starting point is actually that *there exists an exponential-time solvable decision problem such that any family of polynomial-size circuit fails to solve it correctly on all but finitely many input lengths.*[8]

Recall that our goal is to obtain a predicate (i.e., a decision problem) that is computable in exponential-time but is inapproximable by small circuits, where small may mean polynomial-size. For sake of later developments, we formulate a general notion of inapproximability.

Definition 7.9 (inapproximability, a general formulation): *We say that* $f : \{0,1\}^* \to \{0,1\}$ *is* (S,ρ)-inapproximable *if for every family of* S*-size circuits* $\{C_n\}_{n\in\mathbb{N}}$ *and all sufficiently large* n *it holds that*

$$\Pr[C_n(U_n) \neq f(U_n)] \geq \frac{\rho(n)}{2} \tag{7.7}$$

We say that f *is* T-inapproximable *if it is* $(T, 1-(1/T))$*-inapproximable.*

We chose the specific form of Eq. (7.7) such that the "level of inapproximability" represented by the parameter ρ will range in $(0,1)$ and increase with the value of ρ. Specifically, (almost-everywhere) worst case hardness for circuits of size S

[7] In contrast to *standard decoding* in which one recovers the unique information that is encoded in the codeword that is closest to the given string, in list decoding one recovers all strings having encoding that is at a specified distance from the given string. We mention that list decoding is applicable and valuable in the case that the specified distance does not allow for unique decoding and/or that the specified distance is greater than half the distance of the code. See further discussion in Appendix E.1.

[8] Note that our starting point is actually stronger than assuming the existence of a function f in $\mathcal{E} \setminus \mathcal{P}/\text{poly}$. Such an assumption would mean that any family of polynomial-size circuit fails to compute f correctly on infinitely many input lengths, whereas our starting point postulates failures on all but finitely many lengths.

is represented by (S, ρ)-inapproximability with $\rho(n) = 2^{-n+1}$ (i.e., in this case $\mathsf{Pr}[C(U_n) \neq f(U_n)] \geq 2^{-n}$ for every circuit C_n of size $S(n)$), whereas no predicate can be (S, ρ)-inapproximability for $\rho(n) = 1 - O(2^{-n})$ even with $S(n) = O(n)$ (i.e., $\mathsf{Pr}[C(U_n) = f(U_n)] \geq 0.5 + O(2^{-n})$ holds for some linear-size circuit; see Exercise 7.9). Indeed, Eq. (7.7) can be interpreted as an upper-bound on the *correlation* of each adequate circuit with f (i.e., $\mathsf{E}[\chi(C(U_n), f(U_n))] \leq 1 - \rho(n)$, where $\chi(\sigma, \tau) = 1$ if $\sigma = \tau$ and $\chi(\sigma, \tau) = -1$ otherwise). Thus, T-inapproximability means that no family of size T circuits can correlate f better than $1/T$.

Comments. Recall that $\mathcal{E}$ denote the class of exponential-time solvable decision problems (equivalently, exponential-time computable Boolean predicates); that is, $\mathcal{E} = \cup_\varepsilon \text{DTIME}(t_\varepsilon)$, where $t_\varepsilon(n) \stackrel{\text{def}}{=} 2^{\varepsilon n}$. We highlight the aforementioned term *almost everywhere*: Our starting point is not merely that $\mathcal{E}$ is not contained in $\mathcal{P}/\text{poly}$ (or in other circuit size classes to be discussed), but rather that this is the case almost everywhere. Note that by saying that f has circuit complexity exceeding S, we merely mean that *there are infinitely many n's* such that no circuit of size $S(n)$ can computes f correctly on all inputs of length n. In contrast, by saying that f has circuit complexity exceeding S almost everywhere, we mean that *for all but finite many n's* no circuit of size $S(n)$ can computes f correctly on all inputs of length n.

We start (in Section 7.2.1) with a treatment of assumptions and hardness amplification regarding polynomial-size circuits, which suffice for non-trivial derandomization of $\mathcal{BPP}$. We then turn (in Section 7.2.2) to assumptions and hardness amplification regarding exponential-size circuits, which yield a "full" derandomization of $\mathcal{BPP}$ (i.e., $\mathcal{BPP} = \mathcal{P}$). In fact, both sections contain material that is applicable to various other circuit-size bounds, but the motivational focus is as stated.

Teaching note: Section 7.2.2 is advanced material, which is best left for independent reading. Furthermore, for one of the central results (i.e., Lemma 7.23) only an outline is provided and the interested reader is referred to the original paper [121].

7.2.1 Amplification wrt polynomial-size circuits

Our goal here is to prove the following result.

Theorem 7.10 *Suppose that for every polynomial p there exists a problem in $\mathcal{E}$ having circuit complexity that is almost-everywhere greater than p. Then there exist polynomial-inapproximable Boolean functions in $\mathcal{E}$; that is, for every polynomial p there exists a p-inapproximable Boolean function in $\mathcal{E}$.*

Theorem 7.10 is used towards deriving a meaningful derandomization of $\mathcal{BPP}$ under the aforementioned assumption (see Part 2 of Theorem 8.19). We present two proofs of Theorem 7.10. The first proof proceeds in two steps:

1. Starting from the worst-case hypothesis, we first establish some mild level of average-case hardness (i.e., a mild level of inapproximability). Specifically,

we show that for every polynomial p there exists a problem in $\mathcal{E}$ that is (p,ε)-inapproximable for $\varepsilon(n) = 1/n^3$.

2. For any polynomial p, we prove that *if for every polynomial q the function f is $(q, 1/p)$-inapproximable, then the function $F(x_1, ..., x_{t(n)}) = \oplus_{i=1}^{t(n)} f(x_i)$, where $x_1, ..., x_{t(n)} \in \{0,1\}^n$ and $t(n) = n \cdot p(n)$, is T-inapproximable for any polynomial T.* This claim is known as Yao's XOR Lemma, and its proof is far more complex than the proof of its information theoretic analogue.

The second proof of Theorem 7.10 consists of showing that the construction employed in the first step, when composed with Theorem 7.8, actually yields the desired end result. This proof will uncover a connection between hardness amplification and coding theory. Our presentation will thus proceed in three corresponding steps (presented in §7.2.1.1-7.2.1.3, and schematically depicted in Figure 7.2).

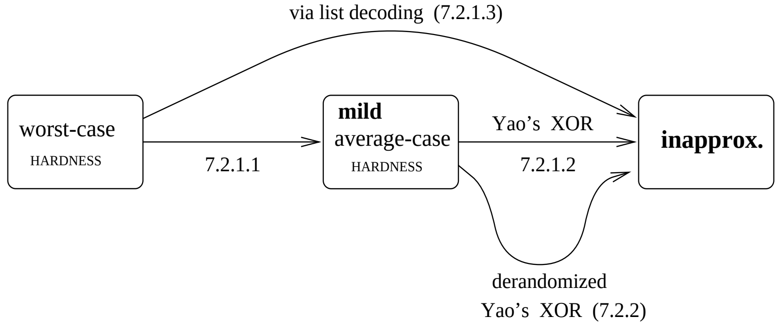


Figure 7.2: Proofs of hardness amplification: organization

7.2.1.1 From worst-case hardness to mild average-case hardness

The transformation of worst-case hardness into average-case hardness (even in a mild sense) is indeed remarkable. Note that worst-case hardness may be due to a relatively small (super-polynomial[9]) number of instances, whereas even mild forms of average-case hardness refer to an exponential number of possible instances. In other words, we should transform hardness that may occur on a negligible fraction of the instances into hardness that occurs on a noticeable fraction of the instances. Intuitively, we should "spread" the hardness of few instances (of the original problem) over all (or most) instances (of the transformed problem). The counter-positive view is that computing the value of typical instances of the transformed problem should enable solving the original problem on every instance.

The aforementioned transformation is based on the *self-correction paradigm* (see also §9.3.2.1), to be reviewed first. The paradigm refers to functions g that can be evaluated at any desired point by using the value of g at a few random points,

[9]Indeed, worst-case hardness for polynomial-size circuits cannot be due to a small (i.e., polynomial) number of instances, because a polynomial number of instances can be hard-wired into such circuits.

where each of these points is uniformly distributed in the function's domain (but indeed the points are not independently distributed). The key observation is that if $g(x)$ can be reconstructed based on the value of g at t such random points, then such a reconstruction can tolerate a $1/3t$ fraction of errors (regarding the values of g). Thus, if we can correctly obtain the value of g on all but at most a $1/3t$ fraction of its domain, then we can probabilistically recover the correct value of g at any point with very high probability. It follows that if no probabilistic polynomial-time algorithm can correctly compute g *in the worst-case sense*, then every probabilistic polynomial-time algorithm must fail to correctly compute g *on at least a $1/3t$ fraction of its domain.*

The archetypical example of a self-correctable function is any m-variate polynomial of individual degree d over a finite field F such that $|F| > dm + 1$. The value of such a polynomial at any desired point x can be recovered based on the values of $dm + 1$ points (other than x) that reside on a random line that passes through x. Note that each of these points is uniformly distributed in F^m, which is the function's domain.

Recall that we are given an arbitrary function $f \in \mathcal{E}$ that is hard to compute in the worst-case. Needless to say, this function is not necessarily self-correctable (based on relatively few points), but it can be extended into such a function. Specifically, we extend $f : [N] \to \{0,1\}$ (viewed as $f : [N^{1/m}]^m \to \{0,1\}$) to an m-variate polynomial of individual degree d over a finite field F such that $|F| > dm+1$ and $(d+1)^m = N$. Intuitively, the extended function is at least as hard on the worst-case as f, and by self-correction the extended function must be mildly hard in the average-case. Details follow.

Construction 7.11 (multi-variate extension)[10]: *For any function $f_n : \{0,1\}^n \to \{0,1\}$, finite field F, $H \subset F$ and integer m such that $|H|^m = 2^n$ and $|F| \geq m|H|$, we consider the function $\hat{f}_n : F^m \to F$ defined as the m-variate polynomial of individual degree $|H| - 1$ that extends $f_n : H^m \to \{0,1\}$. That is, we identify $\{0,1\}^n$ with H^m, and define $\hat{f}_n$ as the unique m-variate polynomial of individual degree $|H| - 1$ that satisfies $\hat{f}_n(x) = f_n(x)$ for every $x \in H^m$, where we view $\{0,1\}$ as a subset of F.*

Note that $\hat{f}_n$ can be evaluated at any desired point, by evaluating f_n on its entire domain, and determining the unique m-variate polynomial of individual degree $|H|-1$ that agrees with f_n on H^m (see Exercise 7.10). Thus, for $f : \{0,1\}^* \to \{0,1\}$ in $\mathcal{E}$, the corresponding $\hat{f}$ (defined by separately extending the restriction of f to each input length) is also in $\mathcal{E}$. For the sake of preserving various complexity measures, we wish to have $|F^m| = \text{poly}(2^n)$, which leads to setting $m = O(n/\log n)$ (yielding $|F| = \text{poly}(n)$, as in §9.3.2.2). In particular, in this case $\hat{f}_n$ is defined over strings of length $O(n)$. The mild average-case hardness of $\hat{f}$ follows by the forgoing discussion. In fact, we state and prove a more general result.

[10]The algebraic fact underlying this construction is that for any function $f : H^m \to F$ there exists a unique m-variate polynomial $\hat{f} : F^m \to F$ of individual degree $|H| - 1$ such that for every $x \in H^m$ it holds that $\hat{f}(x) = f(x)$. This polynomial is called a multi-variate polynomial extension of f, and it can be found in $\text{poly}(|H|^m \log |F|)$-time. For details, see Exercise 7.10.

Theorem 7.12 *Suppose that there exists a Boolean function f in $\mathcal{E}$ having circuit complexity that is almost-everywhere greater than S. Then, there exists an exponential-time computable function $\hat{f} : \{0,1\}^* \to \{0,1\}^*$ such that $|\hat{f}(x)| \leq |x|$ and for every family of circuit $\{C'_{n'}\}_{n' \in \mathbb{N}}$ of size $S'(n') = S(n'/O(1))/\text{poly}(n')$ it holds that $\Pr[C'_{n'}(U_{n'}) \neq \hat{f}(U_{n'})] > (1/n')^2$. Furthermore, $\hat{f}$ does not depend on S.*

Theorem 7.12 completes the first step of the proof of Theorem 7.10, except that we desire a Boolean function rather than one that does not stretch its input. The extra step (of obtaining a Boolean function that is $(\text{poly}(n), n^{-3})$-inapproximable) may be taken by considering the bits in the output of the function (see Exercise 7.11).[11] That is, if $\hat{f}$ is hard to compute on an $(1/n')^2$ fraction of the n'-bit long inputs then the Boolean predicate that returns an indicated bit of $\hat{f}(x)$ must be mildly inapproximable.

Proof: Given f as in the hypothesis and for every $n \in \mathbb{N}$, we consider the restriction of f to $\{0,1\}^n$, denoted f_n, and apply Construction 7.11 to it, while using $m = n/\log n$, $|H| = n$ and $n^2 < |F| = \text{poly}(n)$. Recall that the resulting function $\hat{f}_n$ maps strings of length $n' = \log_2 |F^m| = O(n)$ to strings of length $\log_2 |F| = O(\log n)$. Following the foregoing discussion, we note that by making $m|H| = o(n^2)$ oracle calls to any circuit $C'_{n'}$ that satisfies $\Pr[C'_{n'}(U_{n'}) = \hat{f}_n(U_{n'})] > 1 - (1/n')^2 > 1 - (1/3m|H|)$, we can probabilistically recover the value of ($\hat{f}_n$ and thus) f_n on each input, with probability at least 2/3. Using error-reduction and derandomization as in the proof of Theorem 6.3, we obtain a circuit of size $n^3 \cdot |C'_{n'}|$ that computes f_n. By the hypothesis $n^3 \cdot |C'_{n'}| > S(n)$, and the theorem follows. ∎

Digest. The proof of Theorem 7.12 is actually a worst-case to average-case reduction. That is, the proof consists of a self-correction procedure that allows for the evaluation of f at any desired n-bit long point, using oracle calls to any circuit that computes $\hat{f}$ correctly on a $1 - (1/n')^2$ fraction of the n'-bit long inputs. We note that if $f \in \mathcal{E}$ then $\hat{f} \in \mathcal{E}$, but we do not know how to preserve the complexity of f in case it is in $\mathcal{NP}$. (Various indications to the difficulty of a worst-case to average-case reduction for $\mathcal{NP}$ are known; see, e.g., [40].)

7.2.1.2 Yao's XOR Lemma

Having obtained a mildly inapproximable predicate, we wish to obtain a strongly inapproximable one. The information theoretic context provides an appealing suggestion: Suppose that X is a Boolean random variable (representing the mild inapproximability of the aforementioned predicate) that equals 1 with probability ε. Then XORing the outcome of n/ε independent samples of X yields a bit that equals 1 with probability $0.5 \pm \exp(-\Omega(n))$. It is tempting to think that the same should happen in the computational setting. That is, if f is hard to approximate

[11] A quantitatively stronger bound can be obtained by noting that the proof of Theorem 7.12 actually establishes an error lower-bound of $\Omega((\log n')/(n')^2)$ and that $|\hat{f}(x)| = O(\log |x|)$.

correctly with probability exceeding $1-\varepsilon$ then XORing the output of f on n/ε non-overlapping parts of the input should yield a predicate that is hard to approximate correctly with probability that is non-negligibly higher than $1/2$. The latter assertion turns out to be correct, but (even more than in Section 7.1.2) the proof of the computational phenomenon is considerably more complex than the analysis of the information theoretic analogue.

Theorem 7.13 (Yao's XOR Lemma): *Let p be a polynomial and suppose that the Boolean function f is $(T,1/p)$-inapproximable, for every polynomial T. Then the function $F(x_1,...,x_{t(n)}) = \oplus_{i=1}^{t(n)} f(x_i)$, where $x_1,...,x_{t(n)} \in \{0,1\}^n$ and $t(n) = n \cdot p(n)$, is T'-inapproximable for every polynomial T'.*

Combining Theorem 7.12 (and Exercise 7.11), and Theorem 7.13, we obtain a proof of Theorem 7.10. (Recall that an alternative proof is presented in §7.2.1.3.)

We note that proving Theorem 7.13 seems more difficult than proving Theorem 7.5 (i.e., the amplification of one-way functions), due to two issues. Firstly, unlike in Theorem 7.5, the computational problems are not in $\mathcal{PC}$ and thus we cannot efficiently recognize correct solutions to them. Secondly, unlike in Theorem 7.5, solutions to instances of the transformed problem do not correspond of the concatenation of solutions for the original instances, but are rather a function of the latter that losses almost all the information about the latter. The proof of Theorem 7.13 presented next deals with each of these two difficulties separately.

Several different proofs of Theorem 7.13 are known. We choose using a proof that benefits most from the material already presented in Section 7.1. This proof proceeds in two steps:

1. First we prove that the corresponding "direct product" function $P(x_1,...,x_{t(n)}) = (f(x_1),...,f(x_{t(n)}))$ is difficult to compute in a strong average-case sense.

2. Next we establish the desired result by an application of Theorem 7.8.

Thus, given Theorem 7.8, our main focus is on the first step, which is of independent interest (and is thus generalized from Boolean functions to arbitrary ones).

Theorem 7.14 (The Direct Product Lemma): *Let p be a polynomial and $f : \{0,1\}^* \to \{0,1\}^*$. Suppose that for every family of polynomial-size circuits, $\{C_n\}_{n\in\mathbb{N}}$, and all sufficiently large $n \in \mathbb{N}$, it holds that $\Pr[C_n(U_n) \neq f(U_n)] > 1/p(n)$. Let $P(x_1,...,x_{t(n)}) = (f(x_1),...,f(x_{t(n)}))$, where $x_1,...,x_{t(n)} \in \{0,1\}^n$ and $t(n) = n \cdot p(n)$. Then, for every family of polynomial-size circuits, $\{C'_m\}_{m\in\mathbb{N}}$, it holds that $\mu(m) \stackrel{\text{def}}{=} \Pr[C'_m(U_m) = P(U_m)]$ is a negligible function in m.*

Theorem 7.13 follows from Theorem 7.14 by considering the function $P'(x_1,...,x_{t(n)},r) = b(f(x_1)\cdots f(x_{t(n)}),r)$, where f is a Boolean function, $r \in \{0,1\}^{t(n)}$, and $b(y,r)$ is the inner-product modulo 2 of the $t(n)$-bit long strings y and r. Applying Theorem 7.8, we infer that P' is T'-inapproximable for every polynomial T'. Lastly, we reduce the approximation of P' to the approximation of F (see Exercise 7.12), and Theorem 7.13 follows.

Proof of Theorem 7.14. As in the proof of Theorem 7.5, we show how to converts circuits that violate the theorem's conclusion into circuits that violate the theorem's hypothesis. We note, however, that things were much simpler in the context of Theorem 7.5: There we could (efficiently) check whether or not a value contained in the output of the circuit that solves the direct-product problem constitutes a correct answer for the corresponding instance of the basic problem. Lacking such an ability in the current context, we shall have to use such values more carefully. Loosely speaking, we will take a weighted majority vote among various answers, where the weights reflect our confidence in the correctness of the various answers.

We establish Theorem 7.14 by applying the following lemma that provides quantitative bounds on the feasibility of computing the direct product of two functions. In this lemma, $\{Y_m\}_{m\in\mathbb{N}}$ and $\{Z_m\}_{m\in\mathbb{N}}$ are independent probability ensembles such that $Y_m, Z_m \in \{0,1\}^m$, and $X_n = (Y_{\ell(n)}, Z_{n-\ell(n)})$ for some function $\ell:\mathbb{N}\to\mathbb{N}$. The lemma refers to the success probability of computing the direct product function $F:\{0,1\}^*\to\{0,1\}^*$ defined by $F(yz) = (F_1(y), F_2(z))$, where $|y| = \ell(|yz|)$, when given bounds on the success probability of computing F_1 and F_2 (separately). Needless to say, these probability bounds refer to circuits of certain sizes. We stress that the statement of the lemma is not symmetric with respect to the two functions, guaranteeing a stronger (and in fact lossless) preservation of circuit sizes for one of the functions (which is arbitrarily chosen to be F_1).

Lemma 7.15 (Direct Product, a quantitative two argument version): *For $\{Y_m\}$, $\{Z_m\}$, F_1, F_2, ℓ, $\{X_n\}$ and F as in the foregoing, let $\rho_1(\cdot)$ be an upper-bound on the success probability of $s_1(\cdot)$-size circuits in computing F_1 over $\{Y_m\}$. That is, for every such circuit family $\{C_m\}$*

$$\mathsf{Pr}[C_m(Y_m)=F_1(Y_m)] \leq \rho_1(m).$$

Likewise, suppose that $\rho_2(\cdot)$ is an upper-bound on the probability that $s_2(\cdot)$-size circuits compute F_2 over $\{Z_m\}$. Then, for every function $\varepsilon:\mathbb{N}\to\mathbb{R}$, the function ρ defined as

$$\rho(n) \stackrel{\text{def}}{=} \rho_1(\ell(n)) \cdot \rho_2(n-\ell(n)) + \varepsilon(n)$$

is an upper-bound on the probability that families of $s(\cdot)$-size circuits correctly compute F over $\{X_n\}$, where

$$s(n) \stackrel{\text{def}}{=} \min\left\{ s_1(\ell(n)) \ , \ \frac{s_2(n-\ell(n))}{\text{poly}(n/\varepsilon(n))} \right\}.$$

Theorem 7.14 is derived from Lemma 7.15 by using a careful induction, which capitalizes on the asymmetry of Lemma 7.15. Specifically:

- We write $P(x_1, x_2, ..., x_{t(n)})$ as $P^{(t(n))}(x_1, x_2, ..., x_{t(n)})$, where $P^{(i)}(x_1, ..., x_i) = (f(x_1), ..., f(x_i))$ and $P^{(i)}(x_1, ..., x_i) \equiv (P^{(i-1)}(x_1, ..., x_{i-1}), f(x_i))$.

  For every polynomial s and any non-negligible function ε, we shall prove by induction on i that circuits of size $s(n)$ cannot compute $P^{(i)}(U_{i\cdot n})$ with success probability greater than $(1-(1/p(n))^i + (i-1)\cdot\varepsilon(n)$. Thus, no $s(n)$-size circuit can compute $P^{(t(n))}(U_{t(n)\cdot n})$ with success probability greater than $(1-(1/p(n))^{t(n)} + (t(n)-1)\cdot\varepsilon(n) = \exp(-n) + (t(n)-1)\cdot\varepsilon(n)$. Recalling that this is established for any polynomial s and any non-negligible function ε, Theorem 7.14 follows.

- Turning to the induction itself, we first note that its basis (i.e., $i=1$) is guaranteed by the theorem's hypothesis. The induction step (i.e., from i to $i+1$) is proved using Lemma 7.15 with $F_1 = P^{(i)}$ and $F_2 = f$ (along with $\rho_1(i\cdot n) = (1-(1/p(n))^i + (i-1)\cdot\varepsilon(n)$, $s_1(i\cdot n) = s(n)$, $\rho_2(n) = 1-(1/p(n))$ and $s_2(n) = \mathrm{poly}(n/\varepsilon(n))\cdot s(n)$). In particular, we use again the theorem's hypothesis regarding f, and note that $\rho_1(i\cdot n)\cdot\rho_2(n)+\varepsilon(n)$ is upper-bounded by $(1-(1/p(n))^{i+1} + i\cdot\varepsilon(n)$.

Proof of Lemma 7.15: Proceeding (as usual) by the contrapositive, we consider a family of $s(\cdot)$-size circuits $\{C_n\}_{n\in\mathbb{N}}$ that violates the lemma's conclusion; that is, $\Pr[C_n(X_n) = F(X_n)] > \rho(n)$. We will show how to use such circuits in order to obtain either circuits that violate the lemma's hypothesis regarding F_1 or circuits that violate the lemma's hypothesis regarding F_2. Towards this end, it is instructive to write the success probability of C_n in a conditional form, while denoting the i^{th} output of $C_n(x)$ by $C_n(x)_i$ (i.e., $C_n(x) = (C_n(x)_1, C_n(x)_2)$):

$$\begin{aligned}
&\Pr[C_n(Y_{\ell(n)}, Z_{n-\ell(n)}) = F(Y_{\ell(n)}, Z_{n-\ell(n)})] \\
&\quad = \Pr[C_n(Y_{\ell(n)}, Z_{n-\ell(n)})_1 = F_1(Y_{\ell(n)})] \\
&\qquad \cdot \Pr[C_n(Y_{\ell(n)}, Z_{n-\ell(n)})_2 = F_2(Z_{n-\ell(n)}) \mid C_n(Y_{\ell(n)}, Z_{n-\ell(n)})_1 = F_1(Y_{\ell(n)})].
\end{aligned}$$

The basic idea is that if the first factor is greater than $\rho_1(\ell(n))$ then we immediately derive a circuit (i.e., $C'_n(y) = C_n(y, Z_{n-\ell(n)})_1$) contradicting the lemma's hypothesis regarding F_1, whereas if the second factor is significantly greater than $\rho_2(n-\ell(n))$ then we can obtain a circuit contradicting the lemma's hypothesis regarding F_2. The treatment of the latter case is indeed not obvious. The idea is that a sufficiently large sample of $(Y_{\ell(n)}, F_1(Y_{\ell(n)}))$, which may be hard-wired into the circuit, allows using the conditional probability space (in such a circuit) towards an attempt to approximate F_2. That is, on input z, we select uniformly a string y satisfying $C_n(y,z)_1 = F_1(y)$ (from the aforementioned sample), and output $C_n(y,z)_2$. For a fixed z, sampling of the conditional space (i.e., y's satisfying $C_n(y,z)_1 = F_1(y)$) is possible provided that $\Pr[C_n(Y_{\ell(n)}, z)_1 = F_1(Y_{\ell(n)})]$ holds with noticeable probability. The last caveat motivates a separate treatment of z's having a noticeable value of $\Pr[C_n(Y_{\ell(n)}, z)_1 = F_1(Y_{\ell(n)})]$ and of the rest of z's (which are essentially ignored). Details follow.

Let us first simplify the notations by fixing a generic n and using the abbreviations $C = C_n$, $\varepsilon = \varepsilon(n)$, $\ell = \ell(n)$, $Y = Y_\ell$, and $Z = Y_{n-\ell}$. We call z good if $\Pr[C(Y,z)_1 = F_1(Y)] \geq \varepsilon/2$ and let G be the set of good z's. Next, rather than considering the event $C(Y,Z) = F(Y,Z)$, we consider the event $C(Y,Z) = F(Y,Z) \wedge Z\in G$, which occurs with almost the same probability (up to an additive error term of $\varepsilon/2$). This is the case because, for any $z \notin G$, it holds that

$$\Pr[C(Y,z)=F(Y,z)] \leq \Pr[C(Y,z)_1=F_1(Y)] < \varepsilon/2$$

and thus z's that are not good do not contribute much to $\Pr[C(Y,Z)=F(Y,Z)]$; that is, $\Pr[C(Y,Z)=F(Y,Z) \wedge Z \in G]$ is lower-bounded by $\Pr[C(Y,Z)=F(Y,Z)] - \varepsilon/2$. Using $\Pr[C(Y,z)=F(Y,z)] > \rho(n) = \rho_1(\ell) \cdot \rho_2(n-\ell) + \varepsilon$, we have

$$\Pr[C(Y,Z)=F(Y,Z) \wedge Z \in G] > \rho_1(\ell) \cdot \rho_2(n-\ell) + \frac{\varepsilon}{2}. \tag{7.8}$$

We proceed according to the forgoing outline, first showing that if $\Pr[C(Y,Z)_1 = F_1(Y)] > \rho_1(\ell)$ then we derive circuits violating the hypothesis concerning F_2. Actually, we prove something stronger (which we will actually need for the other case).

Claim 7.15.1: For every z, it holds that $\Pr[C(Y,z)_1=F_1(Y)] \leq \rho_1(\ell)$.

Proof: Otherwise, using any $z \in \{0,1\}^{n-\ell}$ that satisfies $\Pr[C(Y,z)_1 = F_1(Y)] > \rho_1(\ell)$, we obtain a circuit $C'(y) \stackrel{\text{def}}{=} C(y,z)_1$ that contradicts the lemma's hypothesis concerning F_1. □

Using Claim 7.15.1, we show how to obtain a circuit that violates the lemma's hypothesis concerning F_2, and doing so we complete the proof of the lemma.

Claim 7.15.2: There exists a circuit C'' of size $s_2(n-\ell)$ such that

$$\begin{aligned}\Pr[C''(Z)=F_2(Z)] &\geq \frac{\Pr[C(Y,Z)=F(Y,Z) \wedge Z \in G]}{\rho_1(\ell)} - \frac{\varepsilon}{2} \\ &> \rho_2(n-\ell)\end{aligned}$$

Proof: The second inequality is due to Eq. (7.8), and thus we focus on establishing the first inequality. We construct the circuit C'' as suggested in the foregoing outline. Specifically, we take a poly(n/ε)-large sample, denoted S, from the distribution $(Y, F_1(Y))$ and let $C''(z) \stackrel{\text{def}}{=} C(y,z)_2$, where (y,v) is a uniformly selected among the elements of S for which $C(y,z)_1 = v$ holds. Details follow.

Let S be a sequence of $m \stackrel{\text{def}}{=} \text{poly}(n/\varepsilon)$ pairs, generated by taking m independent samples from the distribution $(Y, F_1(Y))$. We stress that we do not assume here that such a sample can be produced by an efficient (uniform) algorithm (but, jumping ahead, we remark that such a sequence can be fixed non-uniformly). For each $z \in G \subseteq \{0,1\}^{n-\ell}$, we denote by S_z the set of pairs $(y,v) \in S$ for which $C(y,z)_1 = v$. Note that S_z is a random sample of the residual probability space defined by $(Y, F_1(Y))$ conditioned on $C(Y,z)_1 = F_1(Y)$. Also, with overwhelmingly high probability, $|S_z| = \Omega(n/\varepsilon^2)$, because $z \in G$ implies $\Pr[C(Y,z)_1=F_1(Y)] \geq \varepsilon/2$ and $m = \Omega(n^2/\varepsilon^3)$. Thus, for each $z \in G$, with overwhelming probability taken over the choices of S, the sample S_z provides a good approximation to the conditional probability space. In particular, with probability greater than $1-2^{-n}$, it holds that

$$\frac{|\{(y,v) \in S_z : C(y,z)_2=F_2(z)\}|}{|S_z|} \geq \Pr[C(Y,z)_2=F_2(z) \mid C(Y,z)_1=F_1(Y)] - \frac{\varepsilon}{2}. \tag{7.9}$$

Thus, with positive probability, Eq. (7.9) holds for all $z \in G \subseteq \{0,1\}^{n-\ell}$. The circuit C'' computing F_2 is now defined as follows. A set $S = \{(y_i, v_i) : i = 1, ..., m\}$ satisfying Eq. (7.9) for all good z's is "hard-wired" into the circuit C''. (In particular, S_z is not empty for any good z.) On input z, the circuit C'' first determines the set S_z, by running C for m times and checking, for each $i = 1, ..., m$, whether or not $C(y_i, z) = v_i$. In case S_z is empty, the circuit returns an arbitrary value. Otherwise, the circuit selects uniformly a pair $(y, v) \in S_z$ and outputs $C(y, z)_2$. (The latter random choice can be eliminated by a standard averaging argument.) Using the definition of C'' and Eq. (7.9), we have:

$$\begin{aligned}
\mathsf{Pr}[C''(Z) = F_2(Z)] &\geq \sum_{z \in G} \mathsf{Pr}[Z = z] \cdot \mathsf{Pr}[C''(z) = F_2(z)] \\
&= \sum_{z \in G} \mathsf{Pr}[Z = z] \cdot \frac{|\{(y,v) \in S_z : C(y,z)_2 = F_2(z)\}|}{|S_z|} \\
&\geq \sum_{z \in G} \mathsf{Pr}[Z = z] \cdot \left(\mathsf{Pr}[C(Y,z)_2 = F_2(z) \,|\, C(Y,z)_1 = F_1(Y)] - \frac{\varepsilon}{2} \right) \\
&= \sum_{z \in G} \mathsf{Pr}[Z = z] \cdot \left(\frac{\mathsf{Pr}[C(Y,z)_2 = F_2(z) \wedge C(Y,z)_1 = F_1(Y)]}{\mathsf{Pr}[C(Y,z)_1 = F_1(Y)]} - \frac{\varepsilon}{2} \right)
\end{aligned}$$

Next, using Claim 7.15.1, we have:

$$\begin{aligned}
\mathsf{Pr}[C''(Z) = F_2(Z)] &\geq \left(\sum_{z \in G} \mathsf{Pr}[Z = z] \cdot \frac{\mathsf{Pr}[C(Y,z) = F(Y,z)]}{\rho_1(\ell)} \right) - \frac{\varepsilon}{2} \\
&= \frac{\mathsf{Pr}[C(Y,Z) = F(Y,Z) \wedge Z \in G]}{\rho_1(\ell)} - \frac{\varepsilon}{2}
\end{aligned}$$

Finally, using Eq. (7.8), the claim follows. □

This completes the proof of the lemma. ■

Comments. Firstly, we wish to call attention to the care with which an inductive argument needs to be carried out in the computational setting, especially when a non-constant number of inductive steps is concerned. Indeed, our inductive proof of Theorem 7.14 involves invoking a quantitative lemma (i.e., Lemma 7.15) that allows to keep track of the relevant quantities (e.g., success probability and circuit size) throughout the induction process. Secondly, we mention that Lemma 7.15 (as well as Theorem 7.14) has a uniform complexity version that assumes that one can efficiently sample the distribution $(Y_{\ell(n)}, F_1(Y_{\ell(n)}))$ (resp., $(U_n, f(U_n))$). For details see [98]. Indeed, a good lesson from the proof of Lemma 7.15 is that non-uniform circuits can "effectively sample" any distribution. Lastly, we mention that Theorem 7.5 (the amplification of one-way functions) and Theorem 7.13 (Yao's XOR Lemma) also have (tight) quantitative versions (see, e.g., [87, Sec. 2.3.2] and [98, Sec. 3], respectively).

7.2.1.3 List decoding and hardness amplification

Recall that Theorem 7.10 was proved in §7.2.1.1-7.2.1.2, by first constructing a mildly inapproximable predicate via Construction 7.11, and then amplifying its hardness via Yao's XOR Lemma. In this subsection we show that the construction used in the first step (i.e., Construction 7.11) actually yields a strongly inapproximable predicate. Thus, we provide an alternative proof of Theorem 7.10. Specifically, we show that a strongly inapproximable predicate (as asserted in Theorem 7.10) can be obtained by combining Construction 7.11 (with a suitable choice of parameters) and the inner-product construction (of Theorem 7.8). The main ingredient of this argument is captured by the following result.

Proposition 7.16 *Suppose that there exists a Boolean function f in $\mathcal{E}$ having circuit complexity that is almost-everywhere greater than S, and let $\varepsilon : \mathbb{N} \to [0,1]$ satisfying $\varepsilon(n) > 2^{-n}$. Let f_n be the restriction of f to $\{0,1\}^n$, and let $\hat{f}_n$ be the function obtained from f_n when applying Construction 7.11*[12] *with $|H| = n/\varepsilon(n)$ and $|F| = |H|^3$. Then, the function $\hat{f} : \{0,1\}^* \to \{0,1\}^*$, defined by $\hat{f}(x) = \hat{f}_{|x|/3}(x)$, is computable in exponential-time and for every family of circuit $\{C'_{n'}\}_{n'\in\mathbb{N}}$ of size $S'(n') = \text{poly}(\varepsilon(n'/3)/n') \cdot S(n'/3)$ it holds that $\Pr[C'_{n'}(U_{n'}) = \hat{f}(U_{n'})] < \varepsilon'(n') \stackrel{\text{def}}{=} \varepsilon(n'/3)$.*

Before turning to the proof of Proposition 7.16, let us describe how it yields an alternative proof of Theorem 7.10. Firstly, for some $\gamma > 0$, Proposition 7.16 yields an exponential-time computable function $\hat{f}$ such that $|\hat{f}(x)| \leq |x|$ and for every family of circuit $\{C'_{n'}\}_{n'\in\mathbb{N}}$ of size $S'(n') = S(n'/3)^{\gamma}/\text{poly}(n')$ it holds that $\Pr[C'_{n'}(U_{n'}) = \hat{f}(U_{n'})] < 1/S'(n')$. Combining this with Theorem 7.8, we infer that $P(x,r) = b(\hat{f}(x), r)$, where $|r| = |\hat{f}(x)| \leq |x|$, is S''-inapproximable for $S''(n'') = S(n''/2)^{\Omega(1)}/\text{poly}(n'')$. In particular, for every polynomial p, we obtain a p-inapproximable predicate in $\mathcal{E}$ by applying the foregoing with $S(n) = \text{poly}(n, p(n))$. Thus, Theorem 7.10 follows.

Proposition 7.16 is proven by observing that the transformation of f_n to $\hat{f}_n$ constitutes a "good" code (see §E.1.1.4) and that any such code provides a worst-case to (strongly) average-case reduction. We start by defining the class of codes that suffices for the latter reduction, while noting that the code underlying the mapping $f_n \mapsto \hat{f}_n$ is actually stronger than needed.

Definition 7.17 (efficient codes supporting implicit decoding): *For fixed functions $q, \ell : \mathbb{N} \to \mathbb{N}$ and $\alpha : \mathbb{N} \to [0,1]$, the mapping $\Gamma : \{0,1\}^* \to \{0,1\}^*$ is* efficient and supports implicit decoding with parameters *q, ℓ, α if it satisfies the following two conditions:*

1. Encoding: *The mapping Γ is polynomial-time computable.*

   It is instructive to view Γ as mapping N-bit long strings to sequences of length $\ell(N)$ over $[q(N)]$, and to view $\Gamma(x) \in [q(|x|)]^{\ell(|x|)}$ as a mapping from $[\ell(|x|)]$ to $[q(|x|)]$.

[12] Recall that in Construction 7.11 we have $|H|^m = 2^n$. Here we relax this condition allowing for $2^n \leq |H|^m < 2^{2m}$.

2. Decoding: *There exists a polynomial p such that the following holds. For every $w:[\ell(N)]\to[q(N)]$ and $x\in\{0,1\}^N$ such that $\Gamma(x)$ is $(1-\alpha(N))$-close to w, there exists an oracle-aided*[13] *circuit C of size $p((\log N)/\alpha(N))$ such that, for every $i\in[N]$, it holds that $C^w(i)$ equals the i^{th} bit of x.*

The encoding condition implies that ℓ is polynomially bounded. The decoding condition refers to any Γ-codeword that agrees with the oracle $w:[\ell(N)]\to[q(N)]$ on an $\alpha(N)$ fraction of the $\ell(N)$ coordinates, *where $\alpha(N)$ may be very small.* We highlight the non-triviality of the decoding condition: There are N bits of information in x, while the size of the circuit C is only $p((\log N)/\alpha(N))$ and yet C should be able to recover any desired entry of x by making queries to w, which may be a highly corrupted version of $\Gamma(x)$. Needless to say, the number of queries made by C is upper-bounded by its size (i.e.,$p((\log N)/\alpha(N))$). On the other hand, the decoding condition does not refer to the complexity of obtaining the aforementioned oracle-aided circuits.

We mention that the transformation of f_n to $\hat{f}_n$ underlying Proposition 7.16 (where $N=2^n$) is efficient and supports implicit decoding with parameters q,ℓ,α such that $\ell(2^n)=\ell(|\langle f_n\rangle|)=|\langle f_n\rangle|^3=2^{3n}$, $\alpha(2^n)=\varepsilon(n)$, and $q(2^n)=(n/\alpha(2^n))^3$. Furthermore, there are at most $O(1/\alpha(2^n)^2)$ codewords (i.e., $\hat{f}_n$'s) that are $(1-\alpha(2^n))$-close to any fixed $w:[\ell(2^n)]\to[q(2^n)]$, and the corresponding oracle-aided circuits can be constructed in probabilistic $p(n/\alpha(2^n))$-time.[14] These results are termed "list decoding" (with implicit representations). We stress that the fact that $f_n\mapsto\hat{f}_n$ satisfies there properties (e.g., constitutes an efficient code that supports implicit decoding) is highly non-trivial, but establishing this fact is beyond the scope of the current text (and the interested reader is referred to [205]). Our focus is on showing that efficient codes that supports implicit decoding suffice for worst-case to (strongly) average-case reductions. We state and prove a general result, noting that in the special case of Proposition 7.16 $g_n=\hat{f}_n$.

Theorem 7.18 *Suppose that there exists a Boolean function f in $\mathcal{E}$ having circuit complexity that is almost-everywhere greater than S, and let $\varepsilon:\mathbb{N}\to[0,1]$. Consider $\ell:\mathbb{N}\to\mathbb{N}$ such that $n\mapsto\log_2\ell(2^n)$ is a 1-1 map of the integers, and let $m(n)=\log_2\ell(2^n)$. Suppose that the mapping $\Gamma:\{0,1\}^*\to\{0,1\}^*$ is efficient and supports implicit decoding with parameters q,ℓ,α such that $\alpha(N)=\varepsilon(\lfloor\log_2 N\rfloor)$. Define $g_n:[\ell(2^n)]\to[q(2^n)]$ such that $g_n(i)=\Gamma(\langle f_n\rangle)(i)$, where $\langle f_n\rangle$ denotes the 2^n-bit long description of the truth-table of f_n. Then, the function $g:\{0,1\}^*\to\{0,1\}^*$, defined by $g(z)=g_{m^{-1}(|z|)}(z)$, is computable in exponential-time and for*

[13] Oracle-aided circuits are defined analogously to oracle Turing machines. Alternatively, we may consider here oracle machines that take advice such that both the advice length and the machine's running time are upper-bounded by $p((\log N)/\alpha(N))$. The relevant oracles may be viewed either as blocks of binary strings that encode sequences over $[q(N)]$ or as sequences over $[q(N)]$. Indeed, in the latter case we consider non-binary oracles, which return elements in $[q(N)]$.

[14] The construction may yield also oracle-aided circuits that compute the decoding of codewords that are almost $(1-\alpha(2^n))$-close to w. That is, there exists a probabilistic $p(n/\alpha(2^n))$-time algorithm that outputs a list of circuits that, with high probability, contains an oracle-aided circuit for the decoding of each codeword that is $(1-\alpha(2^n))$-close to w. Furthermore, with high probability, the list contains only circuits that decode codewords that are $(1-\alpha(2^n)/2)$-close to w.

every family of circuit $\{C'_{n'}\}_{n'\in\mathbb{N}}$ *of size* $S'(n') = \text{poly}(\varepsilon(m^{-1}(n'))/n')\cdot S(m^{-1}(n'))$ *it holds that* $\Pr[C'_{n'}(U_{n'}) = g(U_{n'})] < \varepsilon'(n') \stackrel{\text{def}}{=} \varepsilon(m^{-1}(n'))$.

Proof Sketch: First note that we can generate the truth-table of f_n in exponential-time, and by the encoding condition of Γ it follows that g_n can be evaluated in exponential-time. Regarding g's average-case hardness, consider a circuit $C' = C'_{n'}$ violating the conclusion of the theorem, let $n = m^{-1}(n')$, and recall that $\varepsilon'(n') = \varepsilon(n) = \alpha(2^n)$. Then, C' is $(1-\alpha(2^n))$-close to $g_n = \Gamma(\langle f_n\rangle)$, and the decoding condition of Γ asserts that we can recover each bit of $\langle f_n\rangle$ (i.e., evaluate f_n) by a circuit of size $p(n/\alpha(2^n))\cdot S'(n') < S(n)$, in contradiction to the hypothesis. □

Comment. For simplicity, we formulated Definition 7.17 in a crude manner that suffices for the foregoing application. A more careful formulation of the decoding condition refers to codewords that are $(1-((1/q(N))+\alpha(N)))$-close to the oracle $w:[\ell(N)]\to[q(N)]$ rather than being $(1-\alpha(N))$-close to it.[15] Needless to say, the difference is insignificant in the case that $\alpha(N) \gg 1/q(N)$ (as in Proposition 7.16, where we used $q(N) = ((\log_2 N)/\alpha(N))^3$), but it is significant in case we care about binary codes (i.e., $q(N) = 2$, or codes over other small alphabets). We mention that Theorem 7.18 can be adapted to this context (of $q(N) = 2$), and directly yields strongly inapproximable predicates. For details, see Exercise 7.13.

7.2.2 Amplification wrt exponential-size circuits

For the purpose of stronger derandomization of $\mathcal{BPP}$, we start with a stronger assumption regarding the worst-case circuit complexity of $\mathcal{E}$ and turn it to a stronger inapproximability result.

Theorem 7.19 *Suppose that there exists a decision problem* $L \in \mathcal{E}$ *having almost-everywhere exponential circuit complexity; that is, there exists a constant* $b > 0$ *such that, for all but finitely many* n*'s, any circuit that correctly decides* L *on* $\{0,1\}^n$ *has size at least* 2^{bk}. *Then, for some constant* $c > 0$ *and* $T(n) \stackrel{\text{def}}{=} 2^{c\cdot n}$, *there exists a* T*-inapproximable Boolean function in* $\mathcal{E}$.

Theorem 7.19 can be used for deriving a full derandomization of $\mathcal{BPP}$ (i.e., $\mathcal{BPP} = \mathcal{P}$) under the aforementioned assumption (see Part 1 of Theorem 8.19).

Theorem 7.19 follows as a special case of Proposition 7.16 (combined with Theorem 7.8; see Exercise 7.14). An alternative proof, which uses different ideas that are of independent interest, will be briefly reviewed next. The starting point of the latter proof is a mildly inapproximable predicate, as provided by Theorem 7.12. However, here we cannot afford to apply Yao's XOR Lemma (i.e., Theorem 7.13),

[15] Note that this is the "right" formulation, because in the case that $\alpha(N) < 1/q(n)$ it seems impossible to satisfy the decoding condition (as stated in Definition 7.17). Specifically, a random $\ell(N)$-sequence over $[q(N)]$ is expected to be $(1-(1/q(N)))$-close to any fixed codeword, and with overwhelmingly high probability it will be $(1-((1-o(1))/q(N)))$-close to almost all the codewords, provided $\ell(N) \gg q(n)^2$. But in case $N \gg \log q(N)$, we cannot hope to recover almost all N-bit long strings based on $\text{poly}(q(N)\log N)$ bits of advice (per each of them).

because the latter relates the size of circuits that *strongly* fail to approximate a predicate defined over poly(n)-bit long strings to the size of circuits that fail to *mildly* approximate a predicate defined over n-bit long strings. That is, Yao's XOR Lemma asserts that if $f : \{0,1\}^n \to \{0,1\}$ is mildly inapproximable by S_f-size circuits then $F : \{0,1\}^{\mathrm{poly}(n)} \to \{0,1\}$ is strongly inapproximable by S_F-size circuits, where $S_F(\mathrm{poly}(n))$ is polynomially related to $S_f(n)$. In particular, $S_F(\mathrm{poly}(n)) < S_f(n)$ seems inherent in this reasoning. For the case of polynomial lower-bounds, this is good enough (i.e., if S_f can be an arbitrarily large polynomial then so can S_F), but for $S_F(n) = \exp(\Omega(n))$ we cannot obtain $S_F(m) = \exp(\Omega(m))$ (but rather only obtain $S_F(m) = \exp(m^{\Omega(1)})$).

The source of trouble is that amplification of inapproximability was achieved by taking a polynomial number of independent instances. Indeed, we cannot hope to amplify hardness without applying f on many instances, but these instances need not be independent. Thus, the idea is to define $F(r) = \oplus_{i=1}^{\mathrm{poly}(n)} f(x_i)$, where $x_1, ..., x_{\mathrm{poly}(n)} \in \{0,1\}^n$ are generated from r and still $|r| = O(n)$. That is, we seek a "derandomized" version of Yao's XOR Lemma. In other words, we seek a "pseudorandom generator" of a type appropriate for expanding r to dependent x_i's such that the XOR of the $f(x_i)$'s is as inapproximable as it would have been for independent x_i's.[16]

Teaching note: In continuation to Footnote 16, we note that there is a strong connection between the rest of this section and Chapter 8. On top of the aforementioned conceptual aspects, we will refer to pairwise independence generators (see Section 8.6.1), random walks on expanders (see Section 8.6.3), and even to the Nisan-Wigderson Construction (Construction 8.17).

The pivot of the proof is the notion of a hard region. Loosely speaking, S is a hard region of a Boolean function f if f is *strongly inapproximable on a random input in* S; that is, for every (relatively) small circuit C_n, it holds that $\mathsf{Pr}[C_n(U_n) = f(U_n)|U_n \in S] \approx 1/2$. By definition, $\{0,1\}^*$ is a hard region of any *strongly* inapproximable predicate. One important (and non-trivial) observation is that any *mildly* inapproximable predicate has a hard region of density related to its inapproximability parameter. Loosely speaking, hardness amplification will proceed via methods for generating related instances that hit the hard region with sufficiently high probability. But, first let us study the notion of a hard region.

7.2.2.1 Hard regions

We actually generalize the notion of hard regions to arbitrary distributions. The important special case of uniform distributions is obtained by taking X_n to be U_n (i.e., the uniform distribution over $\{0,1\}^n$). In general, we only assume that $X_n \in \{0,1\}^n$.

[16]Indeed, this falls within the general paradigm discussed in Section 8.2. Furthermore, this suggestion provides another perspective on the connection between randomness and computational difficulty, which is the focus of much discussion in Chapter 8 (see, e.g., §8.3.7.2).

Definition 7.20 (hard region relative to arbitrary distribution): *Let* $f:\{0,1\}^* \to \{0,1\}$ *be a Boolean predicate,* $\{X_n\}$ *be a probability ensemble,* $s:\mathbb{N}\to\mathbb{N}$ *and* $\varepsilon:\mathbb{N}\to[0,1]$.

- *We say that a set* S *is a* hard region of f relative to $\{X_n\}$ *with respect to* $s(\cdot)$*-size circuits and advantage* $\varepsilon(\cdot)$ *if for every* n *and every circuit* C_n *of size at most* $s(n)$*, it holds that*

$$\mathsf{Pr}[C_n(X_n)=f(X_n)|X_n\in S] \le \frac{1}{2}+\varepsilon(n).$$

- *We say that* f *has a* hard region of density $\rho(\cdot)$ *relative to* $\{X_n\}$ (with respect to $s(\cdot)$-size circuits and advantage $\varepsilon(\cdot)$) *if there exists a set* S *that is a hard region of* f *relative to* $\{X_n\}$ (with respect to the foregoing parameters) *such that* $\mathsf{Pr}[X_n\in S_n] \ge \rho(n)$.

Note that a Boolean function f is $(s,1-2\varepsilon)$-inapproximable if and only if $\{0,1\}^*$ is a hard region of f relative to $\{U_n\}$ with respect to $s(\cdot)$-size circuits and advantage $\varepsilon(\cdot)$. Thus, *strongly* inapproximable predicates (e.g., S-inapproximable predicates for super-polynomial S) have a hard region of density 1 (with respect to a negligible advantage).[17] But this trivial observation does not provide hard regions (with respect to a small (i.e., close to zero) advantage) for *mildly* inapproximable predicates. Providing such hard regions is the contents of the following theorem.

Theorem 7.21 (hard regions for mildly inapproximable predicates): *Let* $f:\{0,1\}^*\to\{0,1\}$ *be a Boolean predicate,* $\{X_n\}$ *be a probability ensemble,* $s:\mathbb{N}\to\mathbb{N}$*, and* $\rho:\mathbb{N}\to[0,1]$ *such that* $\rho(n) > 1/\mathrm{poly}(n)$*. Suppose that, for every circuit* C_n *of size at most* $s(n)$*, it holds that* $\mathsf{Pr}[C_n(X_n)=f(X_n)] \le 1-\rho(n)$*. Then, for every* $\varepsilon:\mathbb{N}\to[0,1]$*, the function* f *has a hard region of density* $\rho'(\cdot)$ *relative to* $\{X_n\}$ *with respect to* $s'(\cdot)$*-size circuits and advantage* $\varepsilon(\cdot)$*, where* $\rho'(n) \stackrel{\text{def}}{=} (1-o(1))\cdot\rho(n)$ *and* $s'(n) \stackrel{\text{def}}{=} s(n)/\mathrm{poly}(n/\varepsilon(n))$.

In particular, if f is $(s,2\rho)$-inapproximable then f has a hard region of density $\rho'(\cdot)\approx\rho(\cdot)$ relative to the uniform distribution (with respect to $s'(\cdot)$-size circuits and advantage $\varepsilon(\cdot)$).

Proof Sketch:[18] The proof proceeds by first establishing that $\{X_n\}$ is "related" to (or rather "dominates") an ensemble $\{Y_n\}$ such that f is strongly inapproximable on $\{Y_n\}$, and next showing that this implies the claimed hard region.

For $\rho:\mathbb{N}\to[0,1]$, we say that $\{X_n\}$ ρ-dominates $\{Y_n\}$ if for every x it holds that $\mathsf{Pr}[X_n=x] \ge \rho(n)\cdot\mathsf{Pr}[Y_n=x]$. In this case we also say that $\{Y_n\}$ is ρ-dominated by $\{X_n\}$. We say that $\{Y_n\}$ is critically ρ-dominated by $\{X_n\}$ if for every x either $\mathsf{Pr}[Y_n=x]=(1/\rho(n))\cdot\mathsf{Pr}[X_n=x]$ or $\mathsf{Pr}[Y_n=x]=0$.

The notions of domination and critical domination play a central role in the proof, which consists of two parts. In the first part (Claim 7.21.1), we prove the

[17] Likewise, *mildly* inapproximable predicates have a hard region of density 1 with respect to an advantage that is close to 1/2.

[18] See details in [98, Apdx. A].

existence of a ensemble $\{Y_n\}$ that is ρ-dominated by $\{X_n\}$ such that f is strongly inapproximable on $\{Y_n\}$. In the second part (Claim 7.21.2), we prove that the existence of such a dominated ensemble implies the existence of an ensemble $\{Z_n\}$ that is *critically* ρ'-dominated by $\{X_n\}$ such that f is strongly inapproximable on $\{Z_n\}$. Finally, we note that such a critically dominated ensemble yields a hard region of f relative to $\{X_n\}$, and the theorem follows.

Claim 7.21.1: Under the hypothesis of the theorem it holds that there exists a probability ensemble $\{Y_n\}$ that is ρ-dominated by $\{X_n\}$ such that, for every $s'(n)$-size circuit C_n, it holds that

$$\mathsf{Pr}[C_n(Y_n)=f(Y_n)] \leq \frac{1}{2} + \frac{\varepsilon(n)}{2}. \tag{7.10}$$

Proof: We employ von Neumann's Min-Max Principle (cf. [219]) to a "game" that corresponds to the set of critically dominated (by X_n) probability distributions on one side and the set of $s'(n)$-size circuits on the other side.[19] We start by assuming, towards the contradiction, that for every distribution Y_n that is ρ-dominated by X_n there exists a $s'(n)$-size circuits C_n such that $\mathsf{Pr}[C_n(Y_n) = f(Y_n)] > 0.5+\varepsilon'(n)$, where $\varepsilon'(n) = \varepsilon(n)/2$. One key observation there is a correspondence between the set of distributions that are each ρ-dominated by X_n and the set of all convex combinations of critically ρ-dominated (by X_n) distributions (cf., a special case in §D.4.1.1). Thus, considering an enumeration $Y_n^{(1)}, ..., Y_n^{(t)}$ of the critically ρ-dominated (by X_n) distributions, we conclude that for every distribution π on $[t]$ there exists a $s'(n)$-size circuits C_n such that

$$\sum_{i=1}^{t} \pi(i) \cdot \mathsf{Pr}[C_n(Y_n^{(i)}) = f(Y_n^{(i)})] > 0.5 + \varepsilon'(n). \tag{7.11}$$

Now, consider a finite game between two players, where the first player selects a critically ρ-dominated (by X_n) distribution, and the second player selects a $s'(n)$-size circuit and obtains a payoff as determined by the corresponding success probability; that is, if the first player selects the i^{th} critically dominated distribution and the second player selects the circuit C then the payoff equals $\mathsf{Pr}[C(Y_n^{(i)}) = f(Y_n^{(i)})]$. Eq. (7.11) may be interpreted as saying that for any randomized strategy for the first player there exists a deterministic strategy for the second player yielding average payoff greater than $0.5 + \varepsilon'(n)$. The min-max principle asserts that in such a case there exists a randomized strategy for the second player that yields average payoff greater than $0.5 + \varepsilon'(n)$ no matter what strategy is employed by the first player. This means that there exists a distribution, denoted D_n, on $s'(n)$-size circuits such that for every i it holds that

$$\mathsf{Pr}[D_n(Y_n^{(i)}) = f(Y_n^{(i)})] > 0.5 + \varepsilon'(n), \tag{7.12}$$

where the probability refers both to the choice of the circuit D_n and to the random variable Y_n. Let $B_n = \{x : \mathsf{Pr}[D_n(x) = f(x)] \leq 0.5 + \varepsilon'(n)\}$. Then, $\mathsf{Pr}[X_n \in$

[19]We warn that this application of the min-max principle is somewhat non-straightforward.

$B_n] < \rho(n)$, because otherwise we reach a contradiction to Eq. (7.12) by defining Y_n such that $\Pr[Y_n = x] = \Pr[X_n = x]/\Pr[X_n \in B_n]$ if $x \in B_n$ and $\Pr[Y_n = x] = 0$ otherwise.[20] By employing standard amplification to D_n, we obtain a distribution D'_n over $\mathrm{poly}(n/\varepsilon'(n)) \cdot s'(n)$-size circuits such that for every $x \in \{0,1\}^n \setminus B_n$ it holds that $\Pr[D'_n(x) = f(x)] > 1 - 2^{-n}$. It follows that there exists a $s(n)$-sized circuit C_n such that $C_n(x) = f(x)$ for every $x \in \{0,1\}^n \setminus B_n$, and it follows that $\Pr[C_n(X_n) = f(X_n)] \geq \Pr[X_n \in \{0,1\}^n \setminus B_n] > 1 - \rho(n)$, in contradiction to the theorem's hypothesis. The claim follows. □

We next show that the conclusion of Claim 7.21.1 (which was stated for for ensembles that are ρ-dominated by $\{X_n\}$) essentially holds also for some critically ρ-dominated (by $\{X_n\}$) ensembles. The following precise statement involves some loss in the domination parameter ρ (as well as in the advantage ε).

Claim 7.21.2: If there exists a probability ensemble $\{Y_n\}$ that is ρ-dominated by $\{X_n\}$ such that for every $s'(n)$-size circuit C_n it holds that $\Pr[C_n(Y_n) = f(Y_n)] \geq 0.5 + (\varepsilon(n)/2)$, then there exists a probability ensemble $\{Z_n\}$ that is critically ρ'-dominated by $\{X_n\}$ such that for every $s'(n)$-size circuit C_n it holds that $\Pr[C_n(Z_n) = f(Z_n)] \geq 0.5 + \varepsilon(n)$.

In other words, Claim 7.21.2 asserts that the function f has a hard region of density $\rho'(\cdot)$ relative to $\{X_n\}$ with respect to $s'(\cdot)$-size circuits and advantage $\varepsilon(\cdot)$, thus establishing the theorem. The proof of Claim 7.21.2 uses the Probabilistic Method (cf. [10]). Specifically, we select a set S_n at random by including each n-bit long string x with probability

$$p(x) \stackrel{\text{def}}{=} \frac{\rho(n) \cdot \Pr[Y_n = x]}{\Pr[X_n = x]} \leq 1 \tag{7.13}$$

independently of the choice of all other strings. It can be shown that, with high probability over the choice of S_n, it holds that $\Pr[X_n \in S_n] \approx \rho(n)$ and that $\Pr[C_n(X_n) = f(X_n) | X_n \in S_n] < 0.5 + \varepsilon(n)$ for every circuit C_n of size $s'(n)$. The latter assertion is proved by a union bound on all relevant circuits, showing that for each such circuit C_n, with probability $1 - \exp(-s'(n)^2)$ over the choice of S_n, it holds that $|\Pr[C_n(X_n) = f(X_n) | X_n \in S_n] - \Pr[C_n(Y_n) = f(Y_n)]| < \varepsilon(n)/2$. For details see [98, Apdx. A]. ■

7.2.2.2 Hardness amplification via hard regions

Before showing how to use the notion of a hard region in order to prove a derandomized version of Yao's XOR Lemma, we show how to use it in order to prove the original version of Yao's XOR Lemma (i.e., Theorem 7.13).

---

[20] Note that Y_n is ρ-dominated by X_n, whereas by the hypothesis $\Pr[D_n(Y_n) = f(Y_n)] \leq 0.5 + \varepsilon'(n)$. Using the fact that any ρ-dominated distribution is a convex combination of critically ρ-dominated distributions, it follows that $\Pr[D_n(Y_n^{(i)}) = f(Y_n^{(i)})] \leq 0.5 + \varepsilon'(n)$ holds for some critically ρ-dominated $Y_n^{(i)}$.

An alternative proof of Yao's XOR Lemma. Let f, p, and T be as in Theorem 7.13. Then, by Theorem 7.21, for $\rho'(n) = 1/3p(n)$ and $s'(n) = T(n)^{\Omega(1)}/\text{poly}(n)$, the function f has a hard region S of density ρ' (relative to $\{U_n\}$) with respect to $s'(\cdot)$-size circuits and advantage $1/s'(\cdot)$. Thus, for $t(n) = n \cdot p(n)$ and F as in Theorem 7.13, with probability at least $1-(1-\rho'(n))^{t(n)} = 1-\exp(-\Omega(n))$, one of the $t(n)$ random n-bit blocks of F resides in S (i.e., the hard region of f). Intuitively, this suffices for establishing the strong inapproximability of F. Indeed, suppose towards the contradiction that a small circuit C_n can approximate F with advantage $\varepsilon(n)+\exp(-\Omega(n))$, where $\varepsilon(n) > 1/s'(n)$. Then, the $\varepsilon(n)$ term must be due to $t(n) \cdot n$-bit long inputs that contain a block in S. Using an averaging argument, we can first fix the index of this block and then the contents of the other blocks, and infer the following: for some $i \in [t(n)]$ and $x_1, ..., x_{t(n)} \in \{0,1\}^n$ it holds that

$$\mathsf{Pr}[C_n(x', U_n, x'') = F(x', U_n, x'') \,|\, U_n \in S] \;\geq\; \frac{1}{2} + \varepsilon(n)$$

where $x' = (x_1, ..., x_{i-1}) \in \{0,1\}^{(i-1)\cdot n}$ and $x'' = (x_{i+1}, ..., x_{t(n)}) \in \{0,1\}^{(t(n)-i)\cdot n}$. Hard-wiring $i \in [t(n)]$, $x' = (x_1, ..., x_{i-1})$ and $x'' = (x_{i+1}, ..., x_{t(n)})$ as well as $\sigma \stackrel{\text{def}}{=} \oplus_{j \neq i} f(x_j)$ in C_n, we obtain a contradiction to the (established) fact that S is a hard region of f (by using the circuit $C'_n(z) = C_n(x', z, x'') \oplus \sigma$), and the theorem follows. Actually, we derive a generalization of Theorem 7.13 asserting that *for any function T such that f is $(T, 1/p)$-inapproximable it holds that F is T'-inapproximable for $T'(t(n) \cdot n) = s'(n) = T(n)^{\Omega(1)}/\text{poly}(n)$*.[21]

Derandomized versions of Yao's XOR Lemma. We first show how to use the notion of a hard region in order to amplify very mild inapproximability to a constant level of inapproximability. This amplification utilizes a pairwise independence generator (see Section 8.6.1), denoted G, that stretches $2n$-bit long seeds to sequences of n strings, each of length n.

Lemma 7.22 (derandomized XOR Lemma up to constant inapproximability): *Suppose that $f : \{0,1\}^* \to \{0,1\}$ is (T, ρ)-inapproximable, for $\rho(n) > 1/\text{poly}(n)$, and assume for simplicity that $\rho(n) \leq 1/n$. Let b denote the inner-product mod 2 predicate, and G be the aforementioned pairwise independence generator. Then $F_1(s, r) = b(f(x_1) \cdots f(x_n), r)$, where $|r| = n = |s|/2$ and $(x_1, ..., x_n) = G(s)$, is (T', ρ')-inapproximable for $T'(n') = T(n'/3)/\text{poly}(n')$ and $\rho'(n') = \Omega(n' \cdot \rho(n'/3))$.*

Needless to say, if $f \in \mathcal{E}$ then $F_1 \in \mathcal{E}$. By applying Lemma 7.22 for a constant number of times, we may transform an $(T, 1/\text{poly})$-inapproximable predicate into an $(T'', \Omega(1))$-inapproximable one, where $T''(n'') = T(n''/O(1))/\text{poly}(n'')$.

Proof Sketch: As in the foregoing proof (of the original version of Yao's XOR Lemma), we first apply Theorem 7.21 (this time) inferring that, for $\alpha(n) = \rho(n)/3$ and $s'(n) = T(n)/\text{poly}(n)$, the function f has a hard region S of density α (relative

[21] This generalization can also be established using the proof techniques presented in §7.2.1.2.

to $\{U_n\}$) with respect to $s'(\cdot)$-size circuits and advantage 0.01. Next, as in §7.2.1.2, we shall consider the corresponding (derandomized) direct product problem; that is, the function $P_1(s) = (f(x_1), ..., f(x_n))$, where $|s| = 2n$ and $(x_1, ..., x_n) = G(s)$. We will first show that P_1 is hard to compute on an $\Omega(n \cdot \alpha(n))$ fraction of the domain, and the quantified inapproximality of F_1 will follow.

One key observation is that, by Exercise 7.15, with probability at least $\beta(n) \stackrel{\text{def}}{=} n \cdot \alpha(n)/2$, at least one of the n strings output by $G(U_{2n})$ resides in S. Intuitively, we expect every $s'(n)$-sized circuit to fail in computing $P_1(U_{2n})$ with probability at least $0.49\beta(n)$, because with probability $\beta(n)$ the sequence $G(U_{2n})$ contains an element in the hard region of f. Things are somewhat more involved (than in the non-derandomized case) because it is not clear what is the conditional distribution of the element(s) residing in the hard region.

For technical reasons[22], we assume (without loss of generality) that $\alpha(n) < 1/2n$ and note that in this case Exercise 7.15 implies that, with probability at least $\beta(n) \stackrel{\text{def}}{=} 0.75 \cdot n \cdot \alpha(n)$, at least one of the n strings output by $G(U_{2n})$ resides in S. We claim that every $(s'(n) - \text{poly}(n))$-sized circuit fails to compute P_1 correctly with probability at least $\gamma(n) = 0.3\beta(n)$. As usual, the claim is proved by a reducibility argument. Let $G(s)_i$ denote the i^{th} string in the sequence $G(s)$ (i.e., $G(s) = (G(s)_1, ..., G(s)_n)$), and note that given i and x we can efficiently sample $G_i^{-1}(x) \stackrel{\text{def}}{=} \{s \in \{0,1\}^{2n} : G(s)_i = x\}$. Given a circuit C_n that computes $P_1(U_{2n})$ correctly with probability $1 - \gamma(n)$, we consider the circuit C'_n that, on input x, uniformly selects $i \in [n]$ and $s \in G_i^{-1}(x)$, and outputs the i^{th} bit in $C_n(s)$. Then, by the construction (of C'_n) and the hypothesis regarding C_n, it holds that

$$\begin{aligned}
\mathsf{Pr}[C'_n(U_n) = f(U_n) | U_n \in S] &\geq \sum_{i=1}^{n} \frac{1}{n} \cdot \mathsf{Pr}[C_n(U_{2n}) = P_1(U_{2n}) | G(U_{2n})_i \in S] \\
&\geq \frac{1}{n} \cdot \frac{\mathsf{Pr}[C_n(U_{2n}) = P_1(U_{2n}) \wedge \exists i\, G_i(U_{2n})_i \in S]}{\max_i \{\mathsf{Pr}[G(U_{2n})_i \in S]\}} \\
&\geq \frac{1}{n} \cdot \frac{(1-\gamma(n)) - (1-\beta(n))}{\alpha(n)} \\
&= \frac{0.7\beta(n)}{n \cdot \alpha(n)} \;>\; 0.52\,.
\end{aligned}$$

This contradicts the fact that S is a hard region of f with respect to $s'(\cdot)$-size circuits and advantage 0.01. Thus, we have established that every $(s'(n) - \text{poly}(n))$-sized circuit fails to compute P_1 correctly with probability at least $\gamma(n) = 0.3\beta(n)$. Employing the simple (warm-up) case discussed at the beginning of the proof of Theorem 7.7 (where the predictor errs with probability less than 1/4), it follows that, for $s''(n') = s(n'/3)/\text{poly}(n')$, every $s''(|s|+|r|)$-sized circuits fails to compute $(s,r) \mapsto b(P_1(s), r)$ with probability at least $\delta(|s|+|r|) \stackrel{\text{def}}{=} 0.24 \cdot \gamma(|r|)$. Thus, F_1 is $(s'', 2\delta)$-inapproximable, and the lemma follows. □

[22]The following argument will rely on the fact that $\beta(n) - \gamma(n) > 0.51\alpha(n)$, where $\gamma(n) = \Omega(\beta(n))$.

The next lemma offers an amplification of constant inapproximability to strong inapproximability. Indeed, combining Theorem 7.12 with Lemmas 7.22 and 7.23, yields Theorem 7.19 (as a special case).

Lemma 7.23 (derandomized XOR Lemma starting with constant inapproximability): *Suppose that $f : \{0,1\}^* \to \{0,1\}$ is (T,ρ)-inapproximable, for some constant ρ, and let b denote the inner-product mod 2 predicate. Then there exists a exponential-time computable function G such that $F_2(s,r) = b(f(x_1)\cdots f(x_n),r)$, where $(x_1,...,x_n) = G(s)$ and $n = \Omega(|s|) = |r| = |x_1| = \cdots = |x_n|$, is T'-inapproximable for $T'(n') = T(n'/O(1))^{\Omega(1)}/\mathrm{poly}(n')$.*

Again, if $f \in \mathcal{E}$ then $F_2 \in \mathcal{E}$.

Proof Outline:[23] As in the proof of Lemma 7.22, we start by establishing a hard region of density $\rho/3$ for f (this time with respect to circuits of size $T(n)^{\Omega(1)}/\mathrm{poly}(n)$ and advantage $T(n)^{-\Omega(1)}$), and focus on the analysis of the (derandomized) direct product problem corresponding to computing the function $P_2(s) = (f(x_1),...,f(x_n))$, where $|s| = O(n)$ and $(x_1,...,x_n) = G(s)$. The "generator" G is defined such that $G(s's'') = G_1(s') \oplus G_2(s'')$, where $|s'| = |s''|$, $|G_1(s')| = |G_2(s'')|$, and the following conditions hold:

1. G_1 is the Expander Random Walk Generator discussed in Section 8.6.3. It can be shown that $G_1(U_{O(n)})$ outputs a sequence of n strings such that for any set S of density ρ, with probability $1 - \exp(-\Omega(\rho n))$, at least $\Omega(\rho n)$ of the strings hit S. Note that this property is inherited by G, provided $|G_1(s')| = |G_2(s'')|$ for any $|s'| = |s''|$. It follows that, with probability $1 - \exp(-\Omega(\rho n))$, a constant fraction of the x_i's in the definition of P_2 hit the hard region of f.

   It is tempting to say that small circuits cannot compute P_2 better than with probability $\exp(-\Omega(\rho n))$, but this is clear only in case the the x_i's that hit the hard region are distributed independently (and uniformly) in it, which is hardly the case here. Indeed, G_2 is used to handle this problem.

2. G_2 is the "set projection" system underlying Construction 8.17; specifically, $G_2(s) = (s_{S_1},...,s_{S_n})$, where each S_i is an n-subset of $[|s|]$ and the S_i's have pairwise intersections of size at most $n/O(1)$.[24] An analysis as in the proof of Theorem 8.18 can be employed for showing that the dependency among the x_i's does not help for computing a particular $f(x_i)$ when given x_i as well as all the other $f(x_j)$'s. (Note that the relevant property of G_2 is inherited by G.)

The actual analysis of the construction (via a guessing game presented in [121, Sec. 3]), links the success probability of computing P_2 to the advantage of guessing f on its hard region. The interested reader is referred to [121]. $\square$

[23] For details, see [121].

[24] Recall that s_S denotes the projection of s on coordinates $S \subseteq [|s|]$; that is, for $s = \sigma_1 \cdots \sigma_k$ and $S = \{i_j : j = 1,...,n\}$, we have $s_S = \sigma_{i_1} \cdots \sigma_{i_n}$.

Digest. Both Lemmas 7.22 and 7.23 are proved by first establishing corresponding "direct product" versions (i.e., derandomized versions of Theorem 7.14). We call the reader's attention to the seemingly crucial role of this step (especially in the proof of Lemma 7.23): We cannot treat the values $f(x_1), ...f(x_n)$ as independent (at least not for the generator G as postulated in these lemmas), and so we seek to avoid analyzing the probability of correctly computing the XOR of *all these values*. In contrast, we have established that it is very hard to correctly compute all n values, and thus *XORing a random subset of these values* yields a strongly inapproximable predicate. Note that the argument used in Exercise 7.12 fails here, because the x_i's are not independent.

Chapter Notes

The notion of a one-way function was suggested by Diffie and Hellman [62]. The notion of weak one-way functions as well as the amplification of one-way functions (Theorem 7.5) were suggested by Yao [223]. A proof of Theorem 7.5 has first appeared in [83].

The concept of hard-core predicates was suggested by Blum and Micali [37]. They also proved that a particular predicate constitutes a hard-core for the "DLP function" (i.e., exponentiation in a finite field), provided that the latter function is one-way. The generic hard-core predicate (Theorem 7.7) was suggested by Levin, and proven as such by Goldreich and Levin [95]. The proof presented here was suggested by Rackoff. We comment that the original proof has its own merits (cf., e.g., [101]).

The construction of canonical derandomizers and, specifically, the Nisan-Wigderson framework (Construction 8.17) has been the driving force behind the study of inapproximable predicates in $\mathcal{E}$. Theorem 7.10 is due to [19], whereas Theorem 7.19 is due to [121]. Both results rely heavily of variants of Yao's XOR Lemma, to be reviewed next.

Like several other fundamental insights attributed to Yao's paper [223], Yao's XOR Lemma (Theorem 7.13) is not even stated in [223] but is rather due to Yao's oral presentations of his paper. The first published proof of Yao's XOR Lemma was given by Levin (see [98, Sec. 3]). Levin's proof is the only one known giving a tight quantitative analysis (on the decrease in the level of approximability), and the interested reader is referred to it (via the non-laconic presentation of [98, Sec. 3]). The proof presented in §7.2.1.2 is due to Goldreich, Nisan and Wigderson [98, Sec. 5].

The notion of a hard region and its applications to proving the original version of Yao's XOR Lemma as well as the first derandomization of it (Lemma 7.22) are due to Impagliazzo [119]. The second derandomization (Lemma 7.23) as well as Theorem 7.19 are due to Impagliazzo and Wigderson [121].

The connection between list decoding and hardness amplification (§7.2.1.3), yielding an alternative proof of Theorem 7.19, is due to Sudan, Trevisan, and Vadhan [205].

Hardness amplification for $\mathcal{NP}$ has been the subject of recent attention: An amplification of mild inapproximability to strong inapproximability is provided in [115], an indication to the impossibility of a worst-case to average-case reductions (at least non-adaptive ones) is provided in [40].

Exercises

Exercise 7.1 Prove that if one way-functions exist then there exists one-way functions that are length preserving (i.e., $|f(x)| = |x|$ for every $x \in \{0,1\}^n$).

Guideline: Clearly, for some polynomial p, it holds that $|f(x)| \leq p(|x|)$ for all x. Assume, without loss of generality that $n \mapsto p(n)$ is 1-1, and let $p^{-1}(m) = n$ if $p(n) \leq m < p(n+1)$. Define $f'(z) = f(x)0^{|z|-|f(x)|}$, where x is the $p^{-1}(|z|)$-bit long prefix of z.

Exercise 7.2 Prove that if a function f is hard to invert in the sense of Definition 7.3 then it is hard to invert in the sense of Definition 7.1.
(Hint: consider a sequence of internal coin tosses that maximizes the probability in Eq. (7.1).)

Exercise 7.3 Assuming the existence of one-way functions, prove that there exists a weak one-way function that is not strongly one-way.

Exercise 7.4 (a universal one-way function) Using the notion of a universal machine, present a polynomial-time computable function that is hard to invert (in the sense of Definition 7.1) if and only if there exist one-way functions.

Guideline: Consider the function F that parses its input into a pair (M, x) and emulates $|x|^3$ steps of M on input x. Note that if there exists a one-way function that can be evaluated in cubic time then F is a weak one-way function. Using padding, prove that there exists a one-way function that can be evaluated in cubic time if and only if there exist one-way functions.

Exercise 7.5 For $\ell > 1$, prove that the following $2^\ell - 1$ samples are pairwise independent and uniformly distributed in $\{0,1\}^n$. The samples are generated by uniformly and independently selecting ℓ strings in $\{0,1\}^n$. Denoting these strings by $s^1, ..., s^\ell$, we generate $2^\ell - 1$ samples corresponding to the different *non-empty* subsets of $\{1, 2, ..., \ell\}$ such that for subset J we let $r^J \stackrel{\text{def}}{=} \oplus_{j \in J} s^j$.

Guideline: For $J \neq J'$, it holds that $r^J \oplus r^{J'} = \oplus_{j \in K} s^j$, where K denotes the symmetric difference of J and J'. See related material in Section 8.6.1.

Exercise 7.6 Prove Theorem 7.8. In particular, provide a detailed presentation of the alternative procedure outlined in Footnote 5.

Exercise 7.7 A polynomial-time computable predicate $b : \{0,1\}^* \to \{0,1\}$ is called a universal hard-core predicate if for every one-way function f, the predicate b is a hard-core of f. Note that the predicate presented in Theorem 7.7 is "almost universal" (i.e., for every one-way function f, that predicate is a hard-core of $f'(x,r) = (f(x), r)$, where $|x| = |r|$). Prove that there exist no universal hard-core predicate.

Guideline: Let b be a candidate universal hard-core predicate, and let f be an arbitrary one-way function. Then consider the function $f'(x) = (f(x), b(x))$.

Exercise 7.8 Prove that if $\mathcal{NP}$ is not contained in $\mathcal{P}$/poly then neither is $\mathcal{E}$. Furthermore, for every $S : \mathbb{N} \to \mathbb{N}$, if some problem in $\mathcal{NP}$ does not have circuits of size S then for some constant $\varepsilon > 0$ there exists a problem in $\mathcal{E}$ that does not have circuits of size S', where $S'(n) = S(n^{\varepsilon})$.

Guideline: Although $\mathcal{NP}$ is not known to be in $\mathcal{E}$, it is the case that SAT is in $\mathcal{E}$, which implies that $\mathcal{NP}$ is reducible to a problem in $\mathcal{E}$.

Exercise 7.9 For every function $f : \{0,1\}^n \to \{0,1\}$, present a linear-size circuit C_n such that $\Pr[C(U_n) = f(U_n)] \geq 0.5 + O(2^{-n})$. Furthermore, for every $t \leq 2^{n-1}$, present a circuit C_n of size $O(t \cdot n)$ such that $\Pr[C(U_n) = f(U_n)] \geq 0.5 + t \cdot 2^{-n}$. Warning: you may not assume that $\Pr[f(U_n) = 1] = 0.5$.

Exercise 7.10 (low degree extension) Prove that for any $H \subset F$ and function $f : H^m \to F$ there exists an m-variate polynomial $\hat{f} : F^m \to F$ of individual degree $|H| - 1$ such that for every $x \in H^m$ it holds that $\hat{f}(x) = f(x)$.

Guideline: Define $\hat{f}(x) = \sum_{a \in H^m} \delta_a(x) \cdot f(a)$, where δ_a is an m-variate of individual degree $|H| - 1$ such that $\delta_a(a) = 1$ whereas $\delta_a(x) = 0$ for every $x \in H^m \setminus \{a\}$. Specifically, $\delta_{a_1,\dots,a_m}(x_1, \dots, x_m) = \prod_{i=1}^{m} \prod b \in H \setminus \{a_i\}((x_i - b)/(a_i - b))$.

Exercise 7.11 Let $\hat{f}$ be as in the conclusion of Theorem 7.12. Prove that there exists a Boolean function g in $\mathcal{E}$ that is (p, ε)-inapproximable for every polynomial p and for $\varepsilon(n) = 1/n^3$.

Guideline: Consider the function g defined such that $g(x, i)$ equals the i^{th} bit of $\hat{f}(x)$.

Exercise 7.12 Let f be a Boolean function, and $b(y, r)$ denote the inner-product modulo 2 of the equal-length strings y and r. Suppose that $F'(x_1, \dots, x_{t(n)}, r) \stackrel{\text{def}}{=} b(f(x_1) \cdots f(x_{t(n)}), r)$, where $x_1, \dots, x_{t(n)} \in \{0,1\}^n$ and $r \in \{0,1\}^{t(n)}$, is T-inapproximable for every polynomial T. Assuming that $n \mapsto t(n) \cdot n$ is 1-1, prove that $F(x) \stackrel{\text{def}}{=} F'(x, 1^{t'(|x|)})$, where $t'(t(n) \cdot n) = t(n)$, is T-inapproximable for every polynomial T.

Guideline: Reduce the approximation of F' to the approximation of F. An important observation is that for any $x = (x_1, \dots, x_{t(n)})$, $x' = (x'_1, \dots, x'_{t(n)})$, and $r = r_1 \cdots r_{t(n)}$ such that $x'_i = x_i$ if $r_i = 1$, it holds that $F'(x, r) = F(x') \oplus \oplus_{i:r_i=0} f(x'_i)$. This suggests a non-uniform reduction of F' to F, which uses "adequate" $z_1, \dots, z_{t(n)} \in \{0,1\}^n$ as well as the corresponding values $f(z_i)$'s as advice. On input $x_1, \dots, x_{t(n)}, r_1 \cdots r_{t(n)}$, the reduction sets $x'_i = x_i$ if $r_i = 1$ and $x'_i = z_i$ otherwise, makes the query $x' = (x'_1, \dots, x'_{t(n)})$ to F, and returns $F(x') \oplus_{i:r_i=0} f(z_i)$. Analyze this reduction in the case that $z_1, \dots, z_{t(n)} \in \{0,1\}^n$ are uniformly distributed, and infer that they can be set to some fixed values.

Exercise 7.13 Consider a modification of Definition 7.17, in which the decoding condition reads as follows (where p is a fixed polynomial): *For every* $w : [\ell(N)] \to$

$[q(N)]$ and $x \in \{0,1\}^N$ such that $\Gamma(x)$ is $(1-((1/q(N))+\alpha(N)))$-close to w, there exists an oracle-aided circuit C of size $p((\log N)/\alpha(N))$ such that $C^w(i)$ yields the i^{th} bit of x for every $i \in [N]$.

1. Formulate and prove a version of Theorem 7.18 that refers to the modified definition (rather than to the original one).

   (Hint: the modified version should refer to computing $g(U_{m(n)})$ with success probability greater than $(1/q(n))+\varepsilon(n)$.)

2. Prove that, when applied to binary codes (i.e., $q \equiv 2$), the version in Item 1 yields S''-inapproximable predicates, for $S''(n') = S(m^{-1}(n'))^{\Omega(1)}/\text{poly}(n')$.

3. Prove that the Hadamard Code allows implicit decoding under the modified definition (but not according to the original one).[25]

   (Hint: this is the actual contents of Theorem 7.8.)

Note that if $\Gamma : \{0,1\}^N \to [q(N)]^{\ell(N)}$ is a (non-binary) code that allows implicit decoding then encoding its symbols by the Hadamard code yields a binary code ($\{0,1\}^N \to \{0,1\}^{\ell(N)\cdot 2^{\lceil \log_2 q(N) \rceil}}$) that allows implicit decoding. Note that efficient encoding is preserved only if $q(N) = \text{poly}(N)$.

Exercise 7.14 (using Proposition 7.16 to prove Theorem 7.19) Prove Theorem 7.19 by combining Proposition 7.16 and Theorem 7.8.

Guideline: Note that, for some $\gamma > 0$, Proposition 7.16 yields an exponential-time computable function $\hat{f}$ such that $|\hat{f}(x)| \leq |x|$ and for every family of circuit $\{C'_{n'}\}_{n' \in \mathbb{N}}$ of size $S'(n') = S(n'/3)^{\gamma}/\text{poly}(n')$ it holds that $\mathsf{Pr}[C'_{n'}(U_{n'}) = \hat{f}(U_{n'})] < 1/S'(n')$. Combining this with Theorem 7.8, infer that $P(x,r) = b(\hat{f}(x),r)$, where $|r| = |\hat{f}(x)| \leq |x|$, is S''-inapproximable for $S''(n'') = S(n''/2)^{\Omega(1)}/\text{poly}(n'')$. Note that if $S(n) = 2^{\Omega(n)}$ then $S''(n'') = 2^{\Omega(n'')}$.

Exercise 7.15 Let G be a pairwise independent generator (i.e., as in Lemma 7.22), $S \subset \{0,1\}^n$ and $\alpha \stackrel{\text{def}}{=} |S|/2^n$. Prove that, with probability at least $\min(n \cdot \alpha, 1)/2$, at least one of the n strings output by $G(U_{2n})$ resides in S.

Guideline: Using the pairwise independence property and employing the Inclusion-Exclusion formula, we lower-bound the aforementioned probability by $n \cdot p - \binom{n}{2} \cdot p^2$. If $p \leq 1/n$ then the claim follows, otherwise we employ the same reasoning to the first $1/p$ elements in the output of $G(U_{2n})$.

Exercise 7.16 (one-way functions versus inapproximable predicates) Prove that the existence of a non-uniformly hard one-way function (as in Definition 7.3) implies the existence of an exponential-time computable predicate that is T-inapproximable (as per Definition 7.9), for every polynomial T.

Guideline: Suppose first that the one-way function f is length-preserving and 1-1. Consider the corresponding function g and hard-core predicate b guaranteed by Theorem 7.7,

[25] Needless to say, the Hadamard Code is not efficient (for the trivial reason that its codewords have exponential length).

and show that the Boolean function h such that $h(z) = b(g^{-1}(z))$ is polynomially inapproximable. For the general case a different approach seems needed. Specifically, given a (length preserving) one-way function f, consider the Boolean function h defined as $h(z, i, \sigma) = 1$ if and only if the i^{th} bit of the lexicographically first element in $f^{-1}(z) = \{x : f(x) = z\}$ equals σ. (In particular, if $f^{-1}(z) = \emptyset$ then $h(z, i, \sigma) = 0$ for every i and σ.)[26] Note that h is computable in exponential-time, but is not (worst-case) computable in polynomial-time. Applying Theorem 7.10, we are done.

[26] Thus, h may be easy to computed in the average-case sense (e.g., if $f(x) = 0^{|x|} f'(x)$ for some one-way function f').

Chapter 8

Pseudorandom Generators

Indistinguishable things are identical.[1]

G.W. Leibniz (1646–1714)

A fresh view at the *question of randomness* has been taken in the theory of computing: It has been postulated that a distribution is random (or rather pseudorandom) if it cannot be told apart from the uniform distribution by any efficient procedure. Thus, (pseudo)randomness is not an inherent property of an object, but is rather subjective to the observer.

At the extreme, this approach says that the question of whether the world is deterministic or allows for some free choice (which may be viewed as sources of randomness) is irrelevant. *What matters is how the world looks to us and to various computationally bounded devices.* That is, if some phenomenon looks random then we may just treat it as if it were random. Likewise, if we can generate sequences that cannot be told apart from the uniform distribution by any efficient procedure, then we can use these sequences in any efficient randomized application instead of the ideal random bits that are postulated in the design of this application.

The pivot of this approach is the notion of computational indistinguishability, which refers to pairs of distributions that cannot be told apart by efficient procedures. The most fundamental variant of this notion associates efficient procedures with polynomial-time algorithms, but other variants that restrict attention to other classes of distinguishing procedures also lead to interesting insights. Likewise, the generation of pseudorandom objects is actually a general paradigm with numerous useful incarnations.

Summary: A generic formulation of pseudorandom generators consists of specifying three fundamental aspects – the *stretch measure* of the

[1] This is Leibniz's *Principle of Identity of Indiscernibles.* Leibniz admits that counterexamples to this principle are conceivable but will not occur in real life because God is much too benevolent. We thus believe that he would have agreed to the theme of this chapter, which asserts that *indistinguishable things should be considered as identical.*

generators; the class of distinguishers that the generators are supposed to fool (i.e., the algorithms with respect to which the *computational indistinguishability* requirement should hold); and the resources that the generators are allowed to use (i.e., their own *computational complexity*).

The archetypical case of pseudorandom generators refers to efficient generators that fool any feasible procedure; that is, the potential distinguisher is any probabilistic polynomial-time algorithm, which may be more complex than the generator itself (which, in turn, has time-complexity bounded by a fixed polynomial). These generators are called general-purpose, because their output can be safely used in an efficient application. Such (general-purpose) pseudorandom generators exist if and only if one-way functions exist.

For purposes of derandomization one may use pseudorandom generators that are somewhat more complex than the potential distinguisher (which represents the algorithm to be derandomized). Following this approach, suitable pseudorandom generators, which can be constructed assuming the existence of problems in $\mathcal{E}$ that have no sub-exponential size circuits, yield a full derandomization of $\mathcal{BPP}$ (i.e., $\mathcal{BPP} = \mathcal{P}$).

It is also beneficial to consider pseudorandom generators that fool space-bounded distinguishers and generators that exhibit some limited random behavior (e.g., outputting a pair-wise independent or a small-bias sequence).

8.1 Introduction

The second half of this century has witnessed the development of three theories of randomness, a notion which has been puzzling thinkers for ages. The first theory (cf., [60]), initiated by Shannon [190], is rooted in probability theory and is focused at distributions that are not perfectly random. Shannon's Information Theory characterizes perfect randomness as the extreme case in which the *information contents* is maximized (i.e., there is no redundancy at all). Thus, perfect randomness is associated with a unique distribution – the uniform one. In particular, by definition, one cannot (deterministically) generate such perfect random strings from shorter random seeds.

The second theory (cf., [144, 147]), due to Solomonov [197], Kolmogorov [138] and Chaitin [48], is rooted in computability theory and specifically in the notion of a universal language (equiv., universal machine or computing device; see §1.2.3.3). It measures the complexity of objects in terms of the shortest program (for a fixed universal machine) that generates the object. Like Shannon's theory, Kolmogorov Complexity is quantitative and perfect random objects appear as an extreme case. However, in this approach one may say that a single object, rather than a distribution over objects, is perfectly random. Still, Kolmogorov's approach is inherently intractable (i.e., Kolmogorov Complexity is uncomputable), and – by definition –

one cannot (deterministically) generate strings of high Kolmogorov Complexity from short random seeds.

The third theory is rooted in complexity theory and is the focus of this chapter. This approach is explicitly aimed at providing a notion of randomness that nevertheless allows for an efficient (and deterministic) generation of random strings from shorter random seeds. The heart of this approach is the suggestion to view objects as equal if they cannot be told apart by any efficient procedure. Consequently, a distribution that cannot be efficiently distinguished from the uniform distribution will be considered as being random (or rather called pseudorandom). Thus, randomness is not an "inherent" property of objects (or distributions) but is rather relative to an observer (and its computational abilities). To demonstrate this approach, let us consider the following mental experiment.

> Alice and Bob play "head or tail" in one of the following four ways. In each of them Alice flips an unbiased coin and Bob is asked to guess its outcome *before* the coin hits the floor. The alternative ways differ by the knowledge Bob has before making his guess.
>
> In the first alternative, Bob has to announce his guess before Alice flips the coin. Clearly, in this case Bob wins with probability 1/2.
>
> In the second alternative, Bob has to announce his guess while the coin is spinning in the air. Although the outcome is *determined in principle* by the motion of the coin, Bob does not have accurate information on the motion and thus we believe that also in this case Bob wins with probability 1/2.
>
> The third alternative is similar to the second, except that Bob has at his disposal sophisticated equipment capable of providing accurate *information* on the coin's motion as well as on the environment effecting the outcome. However, Bob cannot process this information in time to improve his guess.
>
> In the fourth alternative, Bob's recording equipment is directly connected to a *powerful computer* programmed to solve the motion equations and output a prediction. It is conceivable that in such a case Bob can improve substantially his guess of the outcome of the coin.

We conclude that the randomness of an event is relative to the information and computing resources at our disposal. At the extreme, deterministic events that are fully determined by some rule may be perceived as random events by observer that lack relevant information and/or ability to process it. Our focus will be on the lack of processing power, which may be due either to the formidable amount of computation required for analyzing the event at question or to the fact that the observer is very limited.

A natural notion of pseudorandomness arises – a distribution is *pseudorandom* if no efficient procedure can distinguish it from the uniform distribution, where efficient procedures are associated with (probabilistic) polynomial-time algorithms. This specific notion of pseudorandomness is indeed the most fundamental one, and much of this chapter is focused on it. Weaker notions of pseudorandomness arise as well – they refer to indistinguishability by weaker procedures such as space-bounded algorithms, constant-depth circuits, etc. Stretching this approach even further one may consider algorithms that are designed on purpose so not to distinguish even weaker forms of "pseudorandom" sequences from random ones (such algorithms arise naturally when trying to convert some natural randomized algorithm into deterministic ones; see Section 8.6).

The foregoing discussion has focused at one aspect of the pseudorandomness question – the resources or type of the observer (or potential distinguisher). Another important aspect is whether such pseudorandom sequences can be generated from much shorter ones, and at what cost (or complexity). A natural approach is that the generation process has to be at least as efficient as the distinguisher (equiv., that the distinguisher is allowed at least as much resources as the generator). Coupled with the aforementioned strong notion of pseudorandomness, this yields the archetypical notion of pseudorandom generators – these operating in polynomial-time and producing sequences that are indistinguishable from uniform ones by *any* polynomial-time observer. Such (general-purpose) pseudorandom generators allow to reduced the randomness complexity of *any efficient application*, and are thus of great relevance to randomized algorithms and cryptography (see Section 8.3).

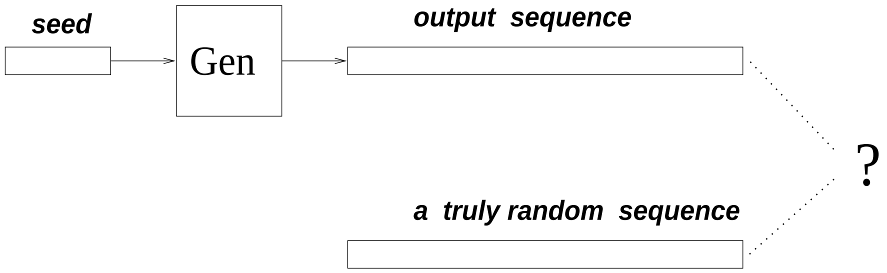


Figure 8.1: Pseudorandom generators – an illustration.

We stress that there are important reasons for considering also an alternative that seems less natural; that is, allowing the pseudorandom generator to use more resources (e.g., time or space) than the observer it tries to fool. This alternative is natural in the context of derandomization (i.e., converting randomized algorithms to deterministic ones), where the crucial step is replacing the random input of an algorithm by a pseudorandom input, which in turn can be generated based on a much shorter random seed. In particular, when derandomizing a probabilistic polynomial-time algorithm, the observer (to be fooled by the generator) is a fixed algorithm. In this case employing a more complex generator merely means that the complexity of the derived deterministic algorithm is dominated by the complexity of the generator (rather than by the complexity of the original randomized algorithm). Needless to say, allowing the generator to use more resources than the observer that it tries to fool makes the task of designing pseudorandom generators potentially easier, and enables derandomization results that are not known when using general-purpose pseudorandom generators. The usefulness of this approach is demonstrated in Sections 8.4 through 8.6.

We note that the goal of all types of pseudorandom generators is to allow the generation of "sufficiently random" sequences based on much shorter random seeds. Thus, pseudorandom generators offer significant saving in the randomness complexity of various applications. This saving is valuable because many applications are severely limited in their ability to generate or obtain truly random bits. Furthermore, typically, generating truly random bits is significantly more expensive than standard computation steps. Thus, randomness is a computational resource that should be considered on top of time complexity (analogously to the consideration of space complexity).

Organization. In Section 8.2 we present the general paradigm underlying the various notions of pseudorandom generators. The archetypical case of general-purpose pseudorandom generators is presented in Section 8.3. We then turn to the alternative notions of pseudorandom generators: Generators that suffice for the derandomization of complexity classes such as $\mathcal{BPP}$ are discussed in Section 8.4; Pseudorandom generators in the domain of space-bounded computations are discussed in Section 8.5; and special-purpose generators are discussed in Section 8.6. (For an alternative presentation, which focuses on general-purpose pseudorandom generators and provides more details on it, the reader is referred to [87, Chap. 3].)

Teaching note: If you can afford teaching only one of the alternative notions of pseudorandom generators, then we suggest teaching the notion of general-purpose pseudorandom generators (presented in Section 8.3). This notion is more relevant to computer science at large and the technical material is relatively simpler. The chapter is organized to facilitate this option.

Prerequisites: We assume a basic familiarity with elementary probability theory (see Appendix D.1) and randomized algorithms (see Section 6.1). In particular, standard conventions regarding random variables (presented in Appendix D.1.1) will be extensively used.

8.2 The General Paradigm

Teaching note: We advocate a unified view of various notions of pseudorandom generators. That is, we view these notions as incarnations of a general abstract paradigm, to be presented in this section. A teacher that wishes to focus on one of the special cases may still use this section as a general motivation towards the specific definitions used later.

A generic formulation of pseudorandom generators consists of specifying three fundamental aspects – the *stretch measure* of the generators; the class of distinguishers that the generators are supposed to fool (i.e., the algorithms with respect to which the *computational indistinguishability* requirement should hold); and the resources that the generators are allowed to use (i.e., their own *computational complexity*).

Stretch function: A necessary requirement from any notion of a pseudorandom generator is that it is a *deterministic algorithm* that stretches short strings, called seeds, into longer output sequences. Specifically, it stretches k-bit long seeds into $\ell(k)$-bit long outputs, where $\ell(k) > k$. The function $\ell : \mathbb{N} \to \mathbb{N}$ is called the stretch measure (or stretch function). In some settings the specific stretch measure is immaterial (e.g., see Section 8.3.4).

Computational Indistinguishability: A necessary requirement from any notion of a pseudorandom generator is that it "fools" some non-trivial algorithms. That is, any algorithm taken from a predetermined class of interest cannot distinguish the output produced by the generator (when the generator is fed with a uniformly chosen seed) from a uniformly chosen sequence. Typically, we consider a class $\mathcal{D}$ of distinguishers and a class $\mathcal{F}$ of (threshold) functions, and require that the generator G satisfies the following: For any $D \in \mathcal{D}$, any $f \in \mathcal{F}$, and for all sufficiently large k's

$$|\,\mathsf{Pr}[D(G(U_k)) = 1] \; - \; \mathsf{Pr}[D(U_{\ell(k)}) = 1]\,| \; < \; f(k) \tag{8.1}$$

where U_n denotes the uniform distribution over $\{0,1\}^n$ and the probability is taken over U_k (resp., $U_{\ell(k)}$) as well as over the coin tosses of algorithm D in case it is probabilistic.[2] The reader may think of such a distinguisher, D, as trying to tell whether the "tested string" is a random output of the generator (i.e., distributed as $G(U_k)$) or is a truly random string (i.e., distributed as $U_{\ell(k)}$). The condition in Eq. (8.1) requires that D cannot make a meaningful decision; that is, ignoring a negligible difference (represented by $f(k)$), D's verdict is the same in both cases. The archetypical choice is that $\mathcal{D}$ is the set of all probabilistic polynomial-time algorithms, and $\mathcal{F}$ is the set of all functions that are the reciprocal of some positive polynomial.

Complexity of Generation: The archetypical choice is that the generator has to work in polynomial-time (in length of its input – the seed). Other choices will be discussed as well. We note that placing no computational requirements on the generator (or, alternatively, putting very mild requirements such as a double-exponential running-time upper bound), yields "generators" that can fool any subexponential-size circuit family (see Exercise 8.1).

[2] The class of threshold functions $\mathcal{F}$ should be viewed as determining the class of noticeable probabilities (as a function of k). Thus, we require certain functions (i.e., the absolute difference between the above probabilities), to be smaller than any noticeable function *on all but finitely many integers*. We call the former functions negligible. Note that a function may be neither noticeable nor negligible (e.g., it may be smaller than any noticeable function on infinitely many values and yet larger than some noticeable function on infinitely many other values).

Notational conventions. We will consistently use k to denote the length of the seed of a pseudorandom generator, and $\ell(k)$ to denote the length of the corresponding output. In some cases, this makes our presentation a little more cumbersome (as a natural presentation may specify some other parameters and let the seed-length be a function of these). However, our choice has the advantage of focusing attention on the fundamental parameter of pseudorandom generation – the length of the random seed. We note that whenever a pseudorandom generator is used to "derandomize" an algorithm, n will denote the length of the input to this algorithm, and k will be selected as a function of n.

Some instantiations of the general paradigm. Two important instantiations of the notion of pseudorandom generators relate to probabilistic polynomial-time distinguishers.

1. General-purpose pseudorandom generators correspond to the case that the generator itself runs in polynomial time and needs to withstand *any probabilistic polynomial-time distinguisher*, including distinguishers that run for more time than the generator. Thus, the same generator may be used safely in any efficient application. (This notion is treated in Section 8.3.)

2. In contrast, pseudorandom generators intended for derandomization may run more time than the distinguisher, which is viewed as a fixed circuit having size that is upper-bounded by a fixed polynomial. (This notion is treated in Section 8.4.)

In addition, the general paradigm may be instantiated by focusing on the space complexity of the potential distinguishers (and the generator), rather than on their time complexity. Furthermore, one may also consider distinguishers that merely reflect probabilistic properties such as pair-wise independence, small-bias, and hitting frequency.

8.3 General-Purpose Pseudorandom Generators

Randomness is playing an increasingly important role in computation: It is frequently used in the design of sequential, parallel and distributed algorithms, and it is of course central to cryptography. Whereas it is convenient to design such algorithms making free use of randomness, it is also desirable to minimize the usage of randomness in real implementations. Thus, general-purpose pseudorandom generators (as defined next) are a key ingredient in an "algorithmic tool-box" – they provide an automatic compiler of programs written with free usage of randomness into programs that make an economical use of randomness.

8.3.1 The basic definition

Loosely speaking, general-purpose pseudorandom generators are efficient (i.e., polynomial-time) deterministic programs that expand short randomly selected seeds into longer pseudorandom bit sequences, where the latter are defined as computationally indistinguishable from truly random sequences by *any* efficient (i.e., polynomial-time) algorithm. Thus, the distinguisher is more complex than the generator: The generator is a fixed algorithm working within *some fixed* polynomial-time, whereas a potential distinguisher is *any* algorithm that runs in polynomial-time. Thus, for example, the distinguisher *may* always run in time cubic in the running-time of the generator. Furthermore, to facilitate the development of this theory, we allow the distinguisher to be probabilistic (whereas the generator remains deterministic as stated above). We require that such distinguishers cannot tell the output of the generator from a truly random string of similar length, or rather that the difference that such distinguishers may detect (or "sense") is negligible. Here a negligible function is one that vanishes faster than the reciprocal of any positive polynomial.

Definition 8.1 (general-purpose pseudorandom generator): *A deterministic polynomial-time algorithm G is called a* pseudorandom generator *if there exists a* stretch function, *$\ell : \mathbb{N} \to \mathbb{N}$ (satisfying $\ell(k) > k$ for all k), such that for any probabilistic polynomial-time algorithm D, for any positive polynomial p, and for all sufficiently large k's*

$$|\,\mathsf{Pr}[D(G(U_k)) = 1] - \mathsf{Pr}[D(U_{\ell(k)}) = 1]\,| < \frac{1}{p(k)} \tag{8.2}$$

where U_n denotes the uniform distribution over $\{0,1\}^n$ and the probability is taken over U_k (resp., $U_{\ell(k)}$) as well as over the internal coin tosses of D.

Thus, Definition 8.1 is derived from the generic framework (presented in Section 8.2) by taking the class of distinguishers to be the set of all probabilistic polynomial-time algorithms, and taking the class of (noticeable) threshold functions to be the set of all functions that are the reciprocals of some positive polynomial.[3] The latter choice is naturally coupled with the association of efficient computation with polynomial-time algorithms: An event that occurs with noticeable probability occurs almost always when the experiment is repeated a "feasible" (i.e., polynomial) number of times.

We note that Definition 8.1 does not make any requirement regarding the stretch function $\ell : \mathbb{N} \to \mathbb{N}$, except for the generic requirement that $\ell(k) > k$ for all k. Needless to say, the larger ℓ is the more useful is the pseudorandom generator. In Section 8.3.4 we show how to use any pseudorandom generator (even one with minimal stretch $\ell(k) = k + 1$) in order to obtain a pseudorandom generator of any desired polynomial stretch function. But before going so, we rigorously discuss the "reduction in randomness" offered by pseudorandom generators, and provide a wider perspective on the notion of computational indistinguishability that underlies Definition 8.1.

[3]Definition 8.1 requires that the distinguishing gap of certain algorithms must be smaller than the reciprocal of any positive polynomial for all but finitely many k's. Such functions are called *negligible*; see Footnote 2. The notion of negligible probability is robust in the sense that an event which occurs with negligible probability occurs with negligible probability also when the experiment is repeated a "feasible" (i.e., polynomial) number of times.

8.3.2 The archetypical application

We note that "pseudo-random number generators" appeared with the first computers. However, typical implementations use generators that are not pseudorandom according to Definition 8.1. Instead, at best, these generators are shown to pass *some* ad-hoc statistical test (cf., [137]). We warn that the fact that a "pseudo-random number generator" passes some statistical tests, does not mean that it will pass a new test and that it will be good for a future (untested) application. Furthermore, the approach of subjecting the generator to some ad-hoc tests fails to provide general results of the form "for *all* practical purposes using the output of the generator is as good as using truly unbiased coin tosses." In contrast, the approach encompassed in Definition 8.1 aims at such generality, and in fact is tailored to obtain it: The notion of computational indistinguishability, which underlines Definition 8.1, covers all possible efficient applications guaranteeing that for all of them pseudorandom sequences are as good as truly random ones. Indeed, any efficient randomized algorithm maintains its performance when its internal coin tosses are substituted by a sequence generated by a pseudorandom generator. This substitution is spell-out next.

Construction 8.2 (typical application of pseudorandom generators): *Let G be a pseudorandom generator with stretch function $\ell:\mathbb{N}\to\mathbb{N}$. Let A be a probabilistic algorithm, and $\rho(n)$ denote a* (polynomial) *upper bound on its randomness complexity. Denote by $A(x,r)$ the output of A on input x and coin tosses sequence $r \in \{0,1\}^{\rho(|x|)}$. Consider the following randomized algorithm, denoted A_G:*

> *On input x, set $k = k(|x|)$ to be the smallest integer such that $\ell(k) \geq \rho(|x|)$, uniformly select $s \in \{0,1\}^k$, and output $A(x,r)$, where r is the $\rho(|x|)$-bit long prefix of $G(s)$.*

That is, $A_G(x,s) = A(x,G'(s))$, for $|s| = k(|x|) = \operatorname{argmin}_i\{\ell(i) \geq \rho(|x|)\}$, where $G'(s)$ is the $\rho(|x|)$-bit long prefix of $G(s)$.

Thus, using A_G instead of A, the randomness complexity is reduced from ρ to $\ell^{-1} \circ \rho$, while (as we show next) it is infeasible to find inputs (i.e., x's) on which the *noticeable behavior* of A_G is different from the one of A. For example, if $\ell(k) = k^2$, then the randomness complexity is reduced from ρ to $\sqrt{\rho}$. We stress that the pseudorandom generator G is *universal*; that is, it can be applied to reduce the randomness complexity of *any* probabilistic polynomial-time algorithm A.

Proposition 8.3 *Let A, ρ and G be as in Construction 8.2, and suppose that $\rho:\mathbb{N}\to\mathbb{N}$ is 1-1. Then, for every pair of probabilistic polynomial-time algorithms,* a finder *F and* a tester *T, every positive polynomial p and all sufficiently long n's*

$$\sum_{x\in\{0,1\}^n} \mathsf{Pr}[F(1^n)=x] \cdot |\,\Delta_{A,T}(x)\,| \;<\; \frac{1}{p(n)} \tag{8.3}$$

where $\Delta_{A,T}(x) \stackrel{\text{def}}{=} \mathsf{Pr}[T(x,A(x,U_{\rho(|x|)}))=1] - \mathsf{Pr}[T(x,A_G(x,U_{k(|x|)}))=1]$, and the probabilities are taken over the U_m's as well as over the coin tosses of F and T.

Algorithm F represents a potential attempt to find an input x on which the output of A_G is distinguishable from the output of A. This "attempt" may be benign as in the case that a user employs algorithm A_G on inputs that are generated by some probabilistic polynomial-time application. However, the attempt may also be adversarial as in the case that a user employs algorithm A_G on inputs that are provided by a potentially malicious party. The potential tester, denoted T, represents the potential use of the output of algorithm A_G, and captures the requirement that this output be as good as a corresponding output produced by A. Thus, T is given x as well as the corresponding output produced either by $A_G(x) \stackrel{\rm def}{=} A(x, U_{k(n)})$ or by $A(x) = A(x, U_{\rho(n)})$, and it is required that T cannot tell the difference. In the case that A is a probabilistic polynomial-time *decision procedure*, this means that it is infeasible to find an x on which A_G decides incorrectly (i.e., differently than A). In the case that A is a *search procedure for some NP-relation*, it is infeasible to find an x on which A_G outputs a wrong solution. For details, see Exercise 8.2.

Proof: The proposition is proven by showing that any triplet (A, F, T) violating the claim can be converted into an algorithm D that distinguishes the output of G from the uniform distribution, in contradiction to the hypothesis. The key observation is that $\Delta_{A,T}(x)$ equals $\Pr[T(x, A(x, U_{\rho(n)})) = 1] - \Pr[T(x, A(x, G'(U_{k(n)}))) = 1]$, where $G'(s)$ is the $\rho(n)$-bit long prefix of $G(s)$. Details follow.

As a warm-up, consider the following algorithm D. On input r (taken from either $U_{\ell(k(n))}$ or $G(U_{k(n)})$), algorithm D first obtains $x \leftarrow F(1^n)$, where n can be obtained easily from $|r|$ (because ρ is 1-1 and $1^n \mapsto \rho(n)$ is computable via A). Next, D obtains $y = A(x, r')$, where r' is the $\rho(|x|)$-bit long prefix of r. Finally D outputs $T(x, y)$. Note that D is implementable in probabilistic polynomial-time, and that

$$
\begin{aligned}
D(U_{\rho(n)}) &= T(X_n, A(X_n, U_{\rho(n)}))\,, \text{ where } X_n \stackrel{\rm def}{=} F(1^n) \\
D(G'(U_{k(n)})) &= T(X_n, A(X_n, G'(U_{k(n)})))\,, \text{ where } X_n \stackrel{\rm def}{=} F(1^n)
\end{aligned}
$$

It follows that $\Pr[D(U_{\ell(k(n))}) = 1] - \Pr[D(G(U_{k(n)})) = 1]$ equals $\mathsf{E}[\Delta_{A,T}(F(1^n))]$, which implies a weaker version of the proposition (referring to $\mathsf{E}[\Delta_{A,T}(F(1^n))]$ rather than to $\mathsf{E}[|\Delta_{A,T}(F(1^n))|]$).

In order to prove that $\mathsf{E}[|\Delta_{A,T}(F(1^n))|]$ (rather than to $\mathsf{E}[\Delta_{A,T}(F(1^n))]$) is negligible, we need to modify D a little. We start by assuming, towards the contradiction, that $\mathsf{E}[|\Delta_{A,T}(F(1^n))|] > \varepsilon(n)$ for some non-negligible function ε. On input r (taken from either $U_{\ell(k(n))}$ or $G(U_{k(n)})$), the modified algorithm D first obtains $x \leftarrow F(1^n)$, as before. Next, using a sample of size $\mathrm{poly}(n/\varepsilon(n))$, it approximates $p_U(x) \stackrel{\rm def}{=} \Pr[T(x, A(x, U_{\rho(n)}) = 1]$ and $p_G(x) \stackrel{\rm def}{=} \Pr[T(x, A(x, G'(U_{k(n)})) = 1]$ such that each probability is approximated to within a deviation of $\varepsilon(n)/8$ with negligible error probability (say, $\exp(-n)$). (Note that, so far, the actions of D only depend on the length of its input r, which determines n.) If these approximations indicate that $p_U(x) \geq p_G(x)$ (equiv., that $\Delta_{A,T} \geq 0$) then D outputs $T(x, A(x, r'))$ else it outputs $1 - T(x, A(x, r'))$, where r' is the $\rho(|x|)$-bit long prefix of r and we assume without loss of generality that the output of T is in $\{0, 1\}$. The reader may verify that, for every x, it holds that

$$\begin{aligned}\mathsf{Pr}[D(U_{\rho(n)}) = 1|F(1^n) = x] &- \mathsf{Pr}[D(G'(U_{k(n)})) = 1|F(1^n) = x] \\ &\geq |p_U(x) - p_G(x)| - \frac{\varepsilon(n)}{2} - \exp(-n),\end{aligned}$$

where the error terms are due to possible errors in the approximation of $p_U(x) - p_G(x)$ (which may cause us to flip its sign and incur an error of $2|p_U(x) - p_G(x)|$ in the case that $|p_U(x) - p_G(x)|$ is smaller than our typical approximation error for $p_U(x) - p_G(x)$).[4] Thus, $\mathsf{Pr}[D(U_{\ell(k(n))}) = 1] - \mathsf{Pr}[D(G(U_{k(n)})) = 1]$ is lower-bounded by $\mathsf{E}[|\Delta_{A,T}(F(1^n))|] - (\varepsilon(n)/2) - \exp(-n) > \varepsilon(n)/3$, and the proposition follows. ■

Conclusion. Analogous arguments are applied whenever one wishes to prove that an efficient randomized process (be it an algorithm as above or a multi-party computation) preserves its behavior when one replaces true randomness (assumed in the analysis) by pseudorandomness (used in the implementation). Thus, given a pseudorandom generator with a large stretch function, *one can considerably reduce the randomness complexity in any efficient application.*

8.3.3 Computational Indistinguishability

In this section we spell-out (and study) the definition of computational indistinguishability that underlies Definition 8.1. The general definition of computational indistinguishability refers to *arbitrary* probability ensembles, where a probability ensemble is an infinite sequence of random variables $\{Z_n\}_{n\in\mathbb{N}}$ such that each Z_n ranges over strings of length bounded by a polynomial in n. We say that $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$ are computationally indistinguishable if for every feasible algorithm A the difference $d_A(n) \stackrel{\rm def}{=} |\mathsf{Pr}[A(X_n)=1] - \mathsf{Pr}[A(Y_n)=1]|$ is a negligible function in n. That is:

Definition 8.4 (computational indistinguishability): *The probability ensembles $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$ are* computationally indistinguishable *if for every probabilistic polynomial-time algorithm D, every positive polynomial p, and all sufficiently large n,*

$$|\mathsf{Pr}[D(X_n)=1] - \mathsf{Pr}[D(Y_n)=1]| < \frac{1}{p(n)} \tag{8.4}$$

where the probabilities are taken over the relevant distribution (i.e., either X_n or Y_n) *and over the internal coin tosses of algorithm D. The l.h.s. of Eq.* (8.4)*, when viewed as a function of n, is often called the* distinguishing gap of D, *where $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$ are understood from the context.*

[4]Specifically, the $\varepsilon(n)/2$ term is due to the maximal typical deviation (i.e., $\varepsilon(n)/4$) of our approximation of $p_U(x) - p_G(x)$ and the $\exp(-n)$ term is due to the rare case that our approximation of $p_U(x) - p_G(x)$ errs by more than $\varepsilon(n)/4$. Note that if $|p_U(x) - p_G(x)| \geq \varepsilon(n)/4$ and our approximation of $p_U(x) - p_G(x)$ deviates from its true value by less than $\varepsilon(n)/4$ then we gain the full gap due to x (i.e., $|p_U(x) - p_G(x)|$).

That is, we can think of D as somebody who wishes to distinguish two distributions (based on a sample given to it), and think of the output "1" as D's verdict that the sample was drawn according to the first distribution. Saying that the two distributions are computationally indistinguishable means that if D is a feasible procedure then its verdict is not really meaningful (because the verdict is almost as often 1 when the input is drawn from the first distribution as when the input is drawn from the second distribution). We comment that the absolute value in Eq. (8.4) can be omitted without affecting the definition (see Exercise 8.3), and we will often do so without warning.

In Definition 8.1, we required that the probability ensembles $\{G(U_k)\}_{k\in\mathbb{N}}$ and $\{U_{\ell(k)}\}_{k\in\mathbb{N}}$ be computationally indistinguishable. Indeed, an important special case of Definition 8.4 is when one ensemble is uniform, and in such a case we call the other ensemble pseudorandom.

Non-triviality of Computational Indistinguishability. Clearly, any two probability ensembles that are statistically close[5] are computationally indistinguishable. Needless to say, this is a trivial case of computational indistinguishability, which is due to information theoretic reasons. In contrast, as noted in Section 8.2, there exist probability ensembles that are statistically far apart and yet are computationally indistinguishable (see Exercise 8.1). However, at least one of the probability ensembles in Exercise 8.1 is *not* polynomial-time constructible. One non-trivial case of computational indistinguishability in which both ensembles are polynomial-time constructible is provided by the definition of pseudorandom generators (see Exercise 8.4). As we shall see (in Theorem 8.11), the existence of one-way functions implies the existence of pseudorandom generators, which in turn implies the existence of *polynomial-time constructible* probability ensembles that are statistically far apart and yet are computationally indistinguishable. We mention that this sufficient condition is also necessary (see Exercise 8.5).

Indistinguishability by Multiple Samples

The definition of computational indistinguishability (i.e., Definition 8.4) refers to distinguishers that obtain a single sample from one of the two probability ensembles (i.e., $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$). A more general definition refers to distinguishers that obtain several independent samples from such an ensemble.

Definition 8.5 (indistinguishability by multiple samples): *Let $s:\mathbb{N}\to\mathbb{N}$ be polynomially-bounded. Two probability ensembles, $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$, are* computationally indistinguishable by $s(\cdot)$ samples *if for every probabilistic polynomial-time algorithm, D, every positive polynomial $p(\cdot)$, and all sufficiently large n's*

$$\left|\Pr\left[D(X_n^{(1)},...,X_n^{(s(n))})=1\right] - \Pr\left[D(Y_n^{(1)},...,Y_n^{(s(n))})=1\right]\right| < \frac{1}{p(n)}$$

[5] Two probability ensembles, $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$, are said to be statistically close if for every positive polynomial p and sufficient large n the variation distance between X_n and Y_n (i.e., $\frac{1}{2}\sum_z|\Pr[X_n=z]-\Pr[Y_n=z]|$) is bounded above by $1/p(n)$.

where $X_n^{(1)}$ *through* $X_n^{(s(n))}$ *and* $Y_n^{(1)}$ *through* $Y_n^{(s(n))}$ *are independent random variables, with each* $X_n^{(i)}$ *identical to* X_n *and each* $Y_n^{(i)}$ *identical to* Y_n.

It turns out that in the most interesting cases, computational indistinguishability by a single sample implies computational indistinguishability by any polynomial number of samples. One such case is the case of polynomial-time constructible ensembles. We say that the ensemble $\{Z_n\}_{n\in\mathbb{N}}$ is polynomial-time constructible if there exists a polynomial-time algorithm S so that $S(1^n)$ and Z_n are identically distributed.

Proposition 8.6 *Suppose that* $X \stackrel{\rm def}{=} \{X_n\}_{n\in\mathbb{N}}$ *and* $Y \stackrel{\rm def}{=} \{Y_n\}_{n\in\mathbb{N}}$ *are both polynomial-time constructible, and* s *be a polynomial. Then,* X *and* Y *are computationally indistinguishable by a single sample if and only if they are computationally indistinguishable by* $s(\cdot)$ *samples.*

Clearly, for every polynomial s, computational indistinguishability by $s(\cdot)$ samples implies computational indistinguishability by a single sample. We now prove that, for efficiently constructible ensembles, indistinguishability by a single sample implies indistinguishability by multiple samples. [6] The proof provides a simple demonstration of a central proof technique, known as the *hybrid technique*.

Proof Sketch:[7] To prove that a sequence of independently drawn samples of one distribution is indistinguishable from a sequence of independently drawn samples from the other distribution, we consider *hybrid* sequences such that the $i^{\rm th}$ hybrid consists of i samples taken from the first distribution and the rest taken from the second distribution. The "homogeneous" sequences (which we wish to prove to be computational indistinguishable) are the extreme hybrids (i.e., the first and last hybrids considered above). The key observation is that distinguishing the extreme hybrids (towards the contradiction hypothesis) means distinguishing neighboring hybrids, which in turn yields a procedure for distinguishing single samples of the two original distributions (contradicting the hypothesis that these two distributions are indistinguishable by a single sample). Details follow.

Suppose that D distinguishes $s(n)$ samples of one distribution from $s(n)$ samples of the other, with a distinguishing gap of $\delta(n)$. Denoting the $i^{\rm th}$ hybrid by H_n^i (i.e., $H_n^i = (X_n^{(1)}, ..., X_n^{(i)}, Y_n^{(i+1)}, ..., Y_n^{(s(n))})$), this means that D distinguishes the extreme hybrids (i.e., H_n^0 and $H_n^{s(n)}$) with gap $\delta(n)$. Then D distinguishes a random pair of neighboring hybrids (i.e., D distinguishes the $i^{\rm th}$ hybrid from the $i+1^{\rm st}$ hybrid, for a randomly selected i) with gap at least $\delta(n)/s(n)$. The reason being that

$$\begin{aligned}
&\mathsf{E}_{i\in\{0,\ldots,s(n)-1\}}\left[\mathsf{Pr}[D(H_n^i)=1] - \mathsf{Pr}[D(H_n^{i+1})=1]\right] \\
&= \frac{1}{s(n)} \cdot \sum_{i=0}^{s(n)-1} \left(\mathsf{Pr}[D(H_n^i)=1] - \mathsf{Pr}[D(H_n^{i+1})=1]\right) \qquad (8.5)
\end{aligned}$$

[6] The requirement that both ensembles are polynomial-time constructible is essential; see, Exercise 8.8.

[7] For more details see [87, Sec. 3.2.3].

$$= \frac{1}{s(n)} \cdot \left(\Pr[D(H_n^0) = 1] - \Pr[D(H_n^{s(n)}) = 1] \right) = \frac{\delta(n)}{s(n)}$$

Using D, we obtain a distinguisher D' of single samples: Given a single sample, algorithm D' selects $i \in \{0, ..., s(n) - 1\}$ at random, generates i samples from the first distribution and $s(n) - i - 1$ samples from the second distribution, and invokes D with the $s(n)$-samples sequence obtained when placing the input sample in location $i + 1$. Thus, the construction of D' relies on the hypothesis that both probability ensembles are polynomial-time constructible. In analyzing D', observe that when the single sample (i.e., the input to D') is taken from the first (resp., second) distribution, algorithm D' invokes D on the $i+1^{\text{st}}$ hybrid (resp., i^{th} hybrid). Thus, the distinguishing gap of D' is captured by Eq. (8.5), and the claim follows. □

The hybrid technique – a digest: The hybrid technique constitutes a special type of a "reducibility argument" in which the computational indistinguishability of *complex* ensembles is proven using the computational indistinguishability of *basic* ensembles. The actual reduction is in the other direction: efficiently distinguishing the basic ensembles is reduced to efficiently distinguishing the complex ensembles, and *hybrid* distributions are used in the reduction in an essential way. The following three properties of the construction of the hybrids play an important role in the argument:

1. *The extreme hybrids collide with the complex ensembles*: this property is essential because what we want to prove (i.e., the indistinguishability of the complex ensembles) relates to the complex ensembles.

2. *Neighboring hybrids are easily related to the basic ensembles*: this property is essential because what we know (i.e., the indistinguishability of the basic ensembles) relates to the basic ensembles. We need to be able to translate our knowledge (i.e., computational indistinguishability) of the basic ensembles to knowledge (i.e., computational indistinguishability) of any pair of neighboring hybrids. Typically, it is required to efficiently transform strings in the range of a basic distribution into strings in the range of a hybrid, so that the transformation maps the first basic distribution to one hybrid and the second basic distribution to the neighboring hybrid. (In the proof of Proposition 8.6, the hypothesis that both X and Y are polynomial-time constructible is instrumental for such an efficient transformation.)

3. *The number of hybrids is small* (i.e., polynomial): this property is essential in order to deduce the computational indistinguishability of extreme hybrids from the computational indistinguishability of each pair of neighboring hybrids. Typically, the provable "distinguishability gap" is inversely proportional to the number of hybrids. Indeed, see Eq. (8.5).

We remark that in the course of an hybrid argument, a distinguishing algorithm referring to the complex ensembles is being analyzed and even invoked on arbitrary hybrids. The reader may be annoyed of the fact that the algorithm "was not designed to work on such hybrids" (but rather only on the extreme hybrids). However, *an algorithm is an algorithm*: once it exists we can invoke it on inputs of our choice, and analyze its performance on arbitrary input distributions.

8.3.4 Amplifying the stretch function

Recall that the definition of pseudorandom generators (i.e., Definition 8.1) makes a minimal requirement regarding their stretch; that is, it is only required that the length of the output of such generators is longer than their input. Needless to say, we seek pseudorandom generators with a significant stretch. It turns out (see Construction 8.7) that pseudorandom generators of any stretch function and in particular of stretch $\ell_1(k) \stackrel{\text{def}}{=} k+1$, are easily converted into pseudorandom generators of any desired (polynomially bounded) stretch function, ℓ. (On the other hand, since pseudorandom generators are required (in Definition 8.1) to run in polynomial time, their stretch must be polynomially bounded.) Thus, when talking about the existence of pseudorandom generators, as in Definition 8.1, we may ignore the stretch function.

Construction 8.7 *Let G_1 be a pseudorandom generator with stretch function $\ell_1(k) = k+1$, and ℓ be any polynomially bounded stretch function that is polynomial-time computable. Let*

$$G(s) \stackrel{\text{def}}{=} \sigma_1\sigma_2\cdots\sigma_{\ell(|s|)} \tag{8.6}$$

where $x_0 = s$ and $x_i\sigma_i = G_1(x_{i-1})$, for $i = 1, ..., \ell(|s|)$. (That is, σ_i is the last bit of $G_1(x_{i-1})$ and x_i is the $|s|$-bit long prefix of $G_1(x_{i-1})$.)

Needless to say, G is polynomial-time computable and has stretch ℓ. An alternative construction is considered in Exercise 8.9.

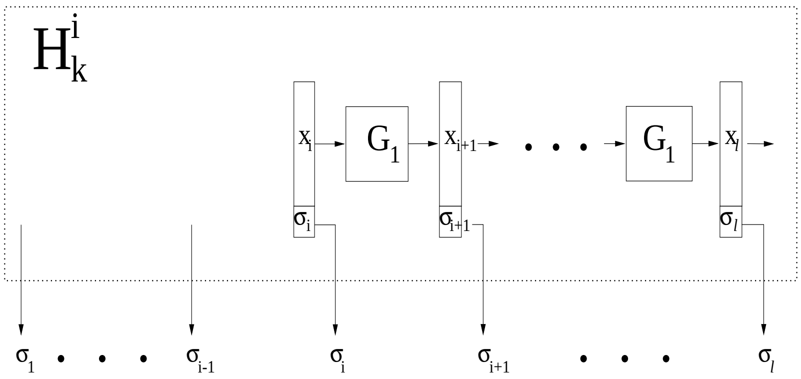


Figure 8.2: Analysis of stretch amplification – the i^{th} hybrid.

Proposition 8.8 *Let G_1 and G be as in Construction 8.7. Then G constitutes a pseudorandom generator.*

Proof Sketch:[8] The proposition is proven using the *hybrid technique*, presented and discussed in Section 8.3.3. Here (for $i = 0, ..., \ell(k)$) we consider the hybrid distributions H_k^i, defined by

$$H_k^i \stackrel{\text{def}}{=} U_i^{(1)} \cdot g_{\ell(k)-i}(U_k^{(2)}),$$

where $U_i^{(1)}$ and $U_k^{(2)}$ are independent uniform distributions (over $\{0,1\}^i$ and $\{0,1\}^k$, respectively), and $g_j(x)$ denotes the j-bit long prefix of $G(x)$. (See Figure 8.2.) The extreme hybrids (i.e., H_k^0 and H_k^k) correspond to $G(U_k)$ and $U_{\ell(k)}$, whereas distinguishability of neighboring hybrids can be worked into distinguishability of $G_1(U_k)$ and U_{k+1}. Details follow.

Suppose that algorithm D distinguishes H_k^i from H_k^{i+1} (with some gap $\delta(k)$). Denoting the first $|x|-1$ bits (resp., last bit) of x by $F(x)$ (resp., $L(x)$), we may write $g_j(s) \equiv (L(G_1(s)), g_{j-1}(F(G_1(s))))$ and

$$\begin{aligned}
H_k^i &= U_i^{(1)} \cdot g_{\ell(k)-i}(U_k^{(2)}) \\
&\equiv (U_i^{(1)}, L(G_1(U_k^{(2)})), g_{(\ell(k)-i)-1}(F(G_1(U_k^{(2)})))) \\
H_k^{i+1} &= U_{i+1}^{(1')} \cdot g_{\ell(k)-i-1}(U_k^{(2)}) \\
&\equiv (U_i^{(1)}, L(U_{k+1}^{(2')}), g_{(\ell(k)-i)-1}(F(U_{k+1}^{(2')}))).
\end{aligned}$$

Then, incorporating the generation of $U_i^{(1)}$ and the evaluation of $g_{\ell(k)-i-1}$ into the distinguisher D, we distinguish $(F(G_1(U_k^{(2)})), L(G_1(U_k^{(2)}))) \equiv G_1(U_k)$ from $(F(U_{k+1}^{(2')}), L(U_{k+1}^{(2')})) \equiv U_{k+1}$, in contradiction to the pseudorandomness of G_1. Specifically, on input $x \in \{0,1\}^{k+1}$, we uniformly select $r \in \{0,1\}^i$ and output $D(r \cdot L(x) \cdot g_{\ell(k)-i-1}(F(x)))$. Thus, the probability we output 1 on input $G_1(U_k)$ (resp., U_{k+1}) equals $\mathsf{Pr}[D(H_k^i) = 1]$ (resp., $\mathsf{Pr}[D(H_k^{i+1}) = 1]$). A final detail refers to the question which i to use. As usual (when the hybrid technique is used), a random i (in $\{0, ..., k-1\}$) will do. $\square$

8.3.5 Constructions

The constructions surveyed in this section "transform" computational difficulty, in the form of one-way functions, into generators of pseudorandomness. Recall that a *polynomial-time computable function* is called one-way if any efficient algorithm can invert it only with negligible success probability (see Definition 7.1 and Section 7.1 for further discussion). We will actually use hard-core predicates of such functions, and refer the reader to their treatment in Section 7.1.3. Loosely speaking, a *polynomial-time computable* predicate b is called a hard-core of a function f if any efficient algorithm, given $f(x)$, can guess $b(x)$ only with success probability that is negligible better than half. Recall that, for any one-way function f, the inner-product mod 2 of x and r is a hard-core of $f'(x,r) = (f(x), r)$. Finally, we get to the construction of pseudorandom generators.

[8] For more details see [87, Sec. 3.3.3].

Proposition 8.9 (A simple construction of pseudorandom generators): *Let b be a hard-core predicate of a polynomial-time computable 1-1 and length-preserving function f. Then, $G(s) \stackrel{\text{def}}{=} f(s) \cdot b(s)$ is a pseudorandom generator.*

Proof Sketch:[9] The $|s|$-bit long prefix of $G(s)$ is uniformly distributed, because f is 1-1 and onto $\{0,1\}^{|s|}$. Hence, the proof boils down to showing that distinguishing $f(s)b(s)$ from $f(s)\cdot\sigma$, where σ is a random bit, yields contradiction to the hypothesis that b is a hard-core of f (i.e., that $b(s)$ is *unpredictable* from $f(s)$). Intuitively, such a distinguisher also distinguishes $f(s)b(s)$ from $f(s) \cdot \overline{b(s)}$, where $\overline{\sigma} = 1 - \sigma$, and distinguishing $f(s) \cdot b(s)$ from $f(s) \cdot \overline{b(s)}$ yields an algorithm for predicting $b(s)$ based on $f(s)$. Details follow.

We start with any potential distinguisher D, and let

$$\delta(k) \stackrel{\text{def}}{=} \mathsf{Pr}[D(G(U_k)) = 1] - \mathsf{Pr}[D(U_{k+1}) = 1].$$

We may assume, without loss of generality, that $\delta(k)$ is non-negative (for infinitely many k's). Using $G(U_k) = f(U_k) \cdot b(U_k)$ and $U_{k+1} \equiv f(U_k) \cdot Z$, where $Z = b(U_k)$ with probability $1/2$ and $Z = \overline{b(U_k)}$ otherwise, we have

$$\mathsf{Pr}[D(f(U_k)b(U_k)) = 1] - \mathsf{Pr}[D(f(U_k)\overline{b(U_k)}) = 1] \; = \; 2\delta(k).$$

Consider an algorithm A that, on input y, uniformly selects $\sigma \in \{0,1\}$, invokes $D(y\sigma)$, and outputs σ if $D(y\sigma) = 1$ and $\overline{\sigma}$ otherwise. Then

$$\begin{aligned} \mathsf{Pr}[A(f(U_k)) = b(U_k)] &= \mathsf{Pr}[D(f(U_k) \cdot \sigma) = 1 \wedge \sigma = b(U_k)] \\ &\quad + \mathsf{Pr}[D(f(U_k) \cdot \sigma) = 0 \wedge \sigma = \overline{b(U_k)}] \\ &= \frac{1}{2} \cdot \Big(\mathsf{Pr}[D(f(U_k) \cdot b(U_k)) = 1] \\ &\qquad + 1 \; - \; \mathsf{Pr}[D(f(U_k) \cdot \overline{b(U_k)}) = 1]\Big) \end{aligned}$$

which equals $(1 + 2\delta(k))/2$. The proposition follows. $\square$

Combining Theorem 7.7, Proposition 8.9 and Construction 8.7, we obtain the following corollary.

Theorem 8.10 (A sufficient condition for the existence of pseudorandom generators): *If there exists 1-1 and length-preserving one-way function then, for every polynomially bounded stretch function ℓ, there exists a pseudorandom generator of stretch ℓ.*

Digest. The key point in the proof of Proposition 8.9 is showing that the (rather obvious) unpredictability of the output of G implies its pseudorandomness. The fact that (next bit) unpredictability and pseudorandomness are equivalent, in general, is proven explicitly in the alternative proof of Theorem 8.10 provided next.

[9] For more details see [87, Sec. 3.3.4].

An alternative presentation. Let us take a closer look at the pseudorandom generators obtained by combining Construction 8.7 and Proposition 8.9. For a stretch function $\ell:\mathbb{N}\to\mathbb{N}$, a 1-1 one-way function f with a hard-core b, we obtain

$$G(s) \stackrel{\rm def}{=} \sigma_1\sigma_2\cdots\sigma_{\ell(|s|)}\,, \tag{8.7}$$

where $x_0 = s$ and $x_i\sigma_i = f(x_{i-1})b(x_{i-1})$ for $i = 1, ..., \ell(|s|)$. Denoting by $f^i(x)$ the value of f iterated i times on x (i.e., $f^i(x) = f^{i-1}(f(x))$ and $f^0(x) = x$), we rewrite Eq. (8.7) as follows

$$G(s) \stackrel{\rm def}{=} b(s)\cdot b(f(s))\cdots b(f^{\ell(|s|)-1}(s))\,. \tag{8.8}$$

The pseudorandomness of G is established in two steps, using the notion of (next bit) unpredictability. An ensemble $\{Z_k\}_{k\in\mathbb{N}}$ is called unpredictable if any probabilistic polynomial-time machine obtaining a (random)[10] prefix of Z_k fails to predict the next bit of Z_k with probability non-negligibly higher than $1/2$. Specifically, we need to establish the following two results.

1. A **general result** asserting that *an ensemble is pseudorandom if and only if it is unpredictable.* Recall that an ensemble is pseudorandom if it is computationally indistinguishable from a uniform distribution (over bit strings of adequate length).

   Clearly, pseudorandomness implies polynomial-time unpredictability, but here we actually need the other direction, which is less obvious. Still, using a hybrid argument, one can show that (next-bit) unpredictability implies indistinguishability from the uniform ensemble. For details see Exercise 8.10.

2. A **specific result** asserting that the ensemble $\{G(U_k)\}_{k\in\mathbb{N}}$ is unpredictable *from right to left.* Equivalently, $G'(U_n)$ is polynomial-time unpredictable (from left to right (as usual)), where $G'(s) = b(f^{\ell(|s|)-1}(s))\cdots b(f(s))\cdot b(s)$ is the reverse of $G(s)$.

   Using the fact that f induces a permutation over $\{0,1\}^n$, observe that the $(j+1)$-bit long prefix of $G'(U_k)$ is distributed identically to $b(f^j(U_k))\cdots b(f(U_k))\cdot b(U_k)$. Thus, an algorithm that predicts the $j+1^{\rm st}$ bit of $G'(U_n)$ based on the j-bit long prefix of $G'(U_n)$ yields an algorithm that guesses $b(U_n)$ based on $f(U_n)$. For details see Exercise 8.12.

Needless to say, G is a pseudorandom generator if and only if G' is a pseudorandom generator (see Exercise 8.11). We mention that Eq. (8.8) is often referred to as the Blum-Micali Construction.[11]

[10] For simplicity, we define unpredictability as referring to prefices of a random length (distributed uniformly in $\{0, ..., |Z_k| - 1\}$).

[11] Given the popularity of the term, we deviate from our convention of not specifying credits in the main text. Indeed, this construction originates in [37].

A general condition for the existence of pseudorandom generators. Recall that given any one-way 1-1 length-preserving function, we can easily construct a pseudorandom generator. Actually, the 1-1 (and length-preserving) requirement may be dropped, but the currently known construction – for the general case – is quite complex.

Theorem 8.11 (On the existence of pseudorandom generators): *Pseudorandom generators exist if and only if one-way functions exist.*

To show that the existence of pseudorandom generators imply the existence of one-way functions, consider a pseudorandom generator G with stretch function $\ell(k) = 2k$. For $x, y \in \{0,1\}^k$, define $f(x,y) \stackrel{\text{def}}{=} G(x)$, and so f is polynomial-time computable (and length-preserving). It must be that f is one-way, or else one can distinguish $G(U_k)$ from U_{2k} by trying to invert and checking the result: Inverting f on its range distribution refers to the distribution $G(U_k)$, whereas the probability that U_{2k} has inverse under f is negligible.

The interesting direction of the proof of Theorem 8.11 is the construction of pseudorandom generators based on any one-way function. In general (when f may not be 1-1) the ensemble $f(U_k)$ may not be pseudorandom, and so Construction 8.9 (i.e., $G(s) = f(s)b(s)$, where b is a hard-core of f) cannot be used *directly*. One idea underlying the known construction is to hash $f(U_k)$ to an almost uniform string of length related to its entropy, using Universal Hash Functions. (This is done after guaranteeing, that the logarithm of the probability mass of a value of $f(U_k)$ is typically close to the entropy of $f(U_k)$.)[12] But "hashing $f(U_k)$ down to length comparable to the entropy" means shrinking the length of the output to, say, $k' < k$. This foils the entire point of stretching the k-bit seed. Thus, a second idea underlying the construction is to compensate for the $k - k'$ loss by extracting these many bits from the seed U_k itself. This is done by hashing U_k, and the point is that the $(k - k')$-bit long hash value does not make the inverting task any easier. Implementing these ideas turns out to be more difficult than it seems, and indeed an alternative construction would be most appreciated.

8.3.6 Non-uniformly strong pseudorandom generators

Recall that we said that truly random sequences can be replaced by pseudorandom ones without affecting any efficient computation. The specific formulation of this assertion, presented in Proposition 8.3, refers to randomized algorithms that take a "primary input" and use a secondary "random input" in their computation. Proposition 8.3 asserts that it is infeasible to find a primary input for which the replacement of a truly random secondary input by a pseudorandom one affects the final output of the algorithm in a noticeable way. This, however, does not mean that such primary inputs do not exist (but rather that they are hard to find).

[12] Specifically, given an arbitrary one-way function f', one first constructs f by taking a "direct product" of sufficiently many copies of f'. For example, for $x_1, ..., x_{k^{2/3}} \in \{0,1\}^{k^{1/3}}$, we let $f(x_1, ..., x_{k^{2/3}}) \stackrel{\text{def}}{=} f'(x_1), ..., f'(x_{k^{2/3}})$.

Consequently, Proposition 8.3 falls short of yielding a (worst-case)[13] "derandomization" of a complexity class such as $\mathcal{BPP}$. To obtain such results, we need a stronger notion of pseudorandom generators, presented next. Specifically, we need pseudorandom generators that can fool all polynomial-size circuits (cf. §1.2.4.1), and not merely all probabilistic polynomial-time algorithms.[14]

Definition 8.12 (strong pseudorandom generator – fooling circuits): *A deterministic polynomial-time algorithm G is called a* non-uniformly strong pseudorandom generator *if there exists a* stretch function, $\ell : \mathbb{N} \to \mathbb{N}$, *such that for any family $\{C_k\}_{k\in\mathbb{N}}$ of polynomial-size circuits, for any positive polynomial p, and for all sufficiently large k's*

$$|\,\mathsf{Pr}[C_k(G(U_k)) = 1] \; - \; \mathsf{Pr}[C_k(U_{\ell(k)}) = 1]\,| \;<\; \frac{1}{p(k)}$$

An alternative formulation is obtained by referring to polynomial-time machines that take advice (Section 3.1.2). Using such pseudorandom generators, we can "derandomize" $\mathcal{BPP}$.

Theorem 8.13 (Derandomization of $\mathcal{BPP}$): *If there exists non-uniformly strong pseudorandom generators then $\mathcal{BPP}$ is contained in $\cap_{\varepsilon>0}\mathrm{D{\small TIME}}(t_\varepsilon)$, where $t_\varepsilon(n) \stackrel{\rm def}{=} 2^{n^\varepsilon}$.*

Proof Sketch: For any $S \in \mathcal{BPP}$ and any $\varepsilon > 0$, we let A denote the decision procedure for L and G denote a non-uniformly strong pseudorandom generator stretching n^ε-bit long seeds into poly(n)-long sequences (to be used by A as secondary input when processing a primary input of length n). We thus obtain an algorithm $A' = A_G$ (as in Construction 8.2). We claim that A and A' may significantly differ in their (expected probabilistic) decision on at most finitely many inputs, because otherwise we can use these inputs (together with A) to derive a (non-uniform) family of polynomial-size circuits that distinguishes $G(U_{n^\varepsilon})$ and $U_{\mathrm{poly}(n)}$, contradicting the the hypothesis regarding G. Specifically, an input x on which A and A' differ significantly yields a circuit C_x that distinguishes $G(U_{|x|^\varepsilon})$ and $U_{\mathrm{poly}(|x|)}$, by letting $C_x(r) = A(x,r)$.[15] Incorporating the finitely many "bad"

[13]Indeed, Proposition 8.3 yields an *average-case derandomization of $\mathcal{BPP}$*. In particular, for every polynomial-time constructible ensemble $\{X_n\}_{n\in\mathbb{N}}$, every Boolean function $f \in \mathcal{BPP}$, and every $\varepsilon > 0$, there exists a randomized algorithm A' of randomness complexity $r_\varepsilon(n) = n^\varepsilon$ such that the probability that $A'(X_n) \neq f(X_n)$ is negligible. A corresponding deterministic ($\exp(r_\varepsilon)$-time) algorithm A'' can be obtained, as in the proof of Theorem 8.13, and again the probability that $A''(X_n) \neq f(X_n)$ is negligible, where here the probability is taken only over the distribution of the primary input (represented by X_n). In contrast, worst-case derandomization, as captured by the assertion $\mathcal{BPP} \subseteq \mathrm{D{\small TIME}}(2^{r_\varepsilon})$, requires that the probability that $A''(X_n) \neq f(X_n)$ is zero.

[14]Needless to say, strong pseudorandom generators in the sense of Definition 8.12 satisfy the basic definition of a pseudorandom generator (i.e., Definition 8.1); see Exercise 8.13. We comment that the underlying notion of computational indistinguishability (by circuits) is strictly stronger than Definition 8.4, and that it is invariant under multiple samples (regardless of the constructibility of the underlying ensembles); for details, see Exercise 8.14.

[15]Indeed, in terms of the proof of Proposition 8.3, the finder F consists of a non-uniform family of polynomial-size circuits that print the "problematic" primary inputs that are hard-wired in them, and the corresponding distinguisher D is thus also non-uniform.

inputs into A', we derive a probabilistic polynomial-time algorithm that decides S while using randomness complexity n^ε.

Finally, emulating A' on each of the 2^{n^ε} possible random choices (i.e., seeds to G) and ruling by majority, we obtain a deterministic algorithm A'' as required. That is, let $A'(x,r)$ denote the output of algorithm A' on input x when using coins $r \in \{0,1\}^{n^\varepsilon}$. Then $A''(x)$ invokes $A'(x,r)$ on every $r \in \{0,1\}^{n^\varepsilon}$, and outputs 1 if and only if the majority of these 2^{n^ε} invocations have returned 1. $\square$

We comment that stronger results regarding derandomization of $\mathcal{BPP}$ are presented in Section 8.4.

On constructing non-uniformly strong pseudorandom generators. Non-uniformly strong pseudorandom generators (as in Definition 8.12) can be constructed using any one-way function that is hard to invert by any non-uniform family of polynomial-size circuits (as in Definition 7.3), rather than by probabilistic polynomial-time machines. In fact, the construction in this case is simpler than the one employed in the uniform case (i.e., the construction underlying the proof of Theorem 8.11).

8.3.7 Other variants and a conceptual discussion

We first mention two stronger variants on the definition of pseudorandom generators, and conclude this section by highlighting various conceptual issues.

8.3.7.1 Stronger notions

The following two notions represent strengthening of the standard definition of pseudorandom generators (as presented in Definition 8.1). Non-uniform versions of these variants (strengthening Definition 8.12) are also of interest.

Fooling stronger distinguishers. One strengthening of Definition 8.1 amounts to explicitly quantifying the resources (and success gaps) of distinguishers. We chose to bound these quantities as a function of the length of the seed (i.e., k), rather than as a function of the length of the string that is being examined (i.e., $\ell(k)$). For a class of time bounds $\mathcal{T}$ (e.g., $\mathcal{T} = \{t(k) \stackrel{\text{def}}{=} 2^{c\sqrt{k}}\}_{c\in\mathbb{N}}$) and a class of noticeable functions (e.g., $\mathcal{F} = \{f(k) \stackrel{\text{def}}{=} 1/t(k) : t \in \mathcal{T}\}$), we say that a pseudorandom generator, G, is $(\mathcal{T},\mathcal{F})$-strong if for any probabilistic algorithm D having running-time bounded by a function in $\mathcal{T}$ (applied to k)[16], for any function f in $\mathcal{F}$, and for all sufficiently large k's, it holds that

$$|\Pr[D(G(U_k)) = 1] - \Pr[D(U_{\ell(k)}) = 1]| < f(k).$$

An analogous strengthening may be applied to the definition of one-way functions. Doing so reveals the weakness of the known construction that underlies the proof

[16] That is, when examining a sequence of length $\ell(k)$ algorithm D makes at most $t(k)$ steps, where $t \in \mathcal{T}$.

of Theorem 8.11: It only implies that for some $\varepsilon > 0$ ($\varepsilon = 1/8$ will do), for any $\mathcal{T}$ and $\mathcal{F}$, the existence of "$(\mathcal{T}, \mathcal{F})$-strong one-way functions" implies the existence of $(\mathcal{T}', \mathcal{F}')$-strong pseudorandom generators, where $\mathcal{T}' = \{t'(k) \stackrel{\text{def}}{=} t(k^\varepsilon)/\text{poly}(k) : t \in \mathcal{T}\}$ and $\mathcal{F}' = \{f'(k) \stackrel{\text{def}}{=} \text{poly}(k) \cdot f(k^\varepsilon) : f \in \mathcal{F}\}$. What we *would like* to have is an analogous result with $\mathcal{T}' = \{t'(k) \stackrel{\text{def}}{=} t(\Omega(k))/\text{poly}(k) : t \in \mathcal{T}\}$ and $\mathcal{F}' = \{f'(k) \stackrel{\text{def}}{=} \text{poly}(k) \cdot f(\Omega(k)) : f \in \mathcal{F}\}$.

Pseudorandom Functions. Pseudorandom generators allow to efficiently generate long pseudorandom sequences from short random seeds. Pseudorandom functions (defined in Appendix C.3.3) are even more powerful: They allow *efficient direct access* to a huge pseudorandom sequence, which is not even feasible to scan bit-by-bit. Put in other words, pseudorandom functions can replace truly random functions in any efficient application (e.g., most notably in cryptography). We mention that pseudorandom functions can be constructed from any pseudorandom generator (see Appendix C.3.3), and that they found many applications in cryptography (see Appendix C.3.3, C.5.2, and C.6.2). Pseudorandom functions have been used to derive negative results in computational learning theory [216] and in the study of circuit complexity (cf., Natural Proofs [177]).

8.3.7.2 Conceptual Discussion

Whoever does not value preoccupation with thoughts, can skip this chapter.

Robert Musil, The Man without Qualities, Chap. 28

We highlight several conceptual aspects of the foregoing computational approach to randomness. Some of these aspects are common to other instantiation of the general paradigm (esp., the one presented in Section 8.4).

Behavioristic versus Ontological. The behavioristic nature of the computational approach to randomness is best demonstrated by confronting this approach with the Kolmogorov-Chaitin approach to randomness. Loosely speaking, a string is *Kolmogorov-random* if its length equals the length of the shortest program producing it. This shortest program may be considered the "true explanation" to the phenomenon described by the string. A Kolmogorov-random string is thus a string that does not have a substantially simpler (i.e., shorter) explanation than itself. Considering the simplest explanation of a phenomenon may be viewed as an ontological approach. In contrast, considering the effect of phenomena on certain devices (or observations), as underlying the definition of pseudorandomness, is a behavioristic approach. Furthermore, there exist probability distributions that are not uniform (and are not even statistically close to a uniform distribution) and nevertheless are indistinguishable from a uniform distribution (by any efficient device). Thus, distributions that are ontologically very different, are considered equivalent by the behavioristic point of view taken in the definition of computational indistinguishability.

A relativistic view of randomness. We have defined pseudorandomness in terms of its observer. Specifically, we have considered the class of efficient (i.e., polynomial-time) observers and defined as pseudorandom objects that look random to any observer in that class. In subsequent sections, we shall consider restricted classes of such observers (e.g., space-bounded polynomial-time observers and even very restricted observers that merely apply specific tests such as linear tests or hitting tests). Each such class of observers gives rise to a different notion of pseudorandomness. Furthermore, the general paradigm (of pseudorandomness) explicitly aims at distributions that are not uniform and yet are considered as such from the point of view of certain observers. Thus, our entire approach to pseudorandomness is relativistic and subjective (i.e., depending on the abilities of the observer).

Randomness and Computational Difficulty. Pseudorandomness and computational difficulty play dual roles: The general paradigm of pseudorandomness relies on the fact that putting computational restrictions on the observer gives rise to distributions that are not uniform and still cannot be distinguished from uniform. Thus, the pivot of the entire approach is the computational difficulty of distinguishing pseudorandom distributions from truly random ones. Furthermore, many of the constructions of pseudorandom generators rely either on conjectures or on facts regarding computational difficulty (i.e., that certain computations that are hard for certain classes). For example, one-way functions were used to construct general-purpose pseudorandom generators (i.e., those working in polynomial-time and fooling all polynomial-time observers). Analogously, as we shall see in §8.4.3.1, the fact that parity function is hard for polynomial-size constant-depth circuits can be used to generate (highly non-uniform) sequences that fool such circuits.

Randomness and Predictability. The connection between pseudorandomness and unpredictability (by efficient procedures) plays an important role in the analysis of several constructions (cf. Sections 8.3.5 and 8.4.2). We wish to highlight the intuitive appeal of this connection.

8.4 Derandomization of time-complexity classes

Let us take a second look at the proof of Theorem 8.13: A pseudorandom generator was used to shrink the randomness complexity of a BPP-algorithm, and derandomization was achieved by scanning all possible seeds to the generator. A key observation regarding this process is that there is no point in insisting that the pseudorandom generator runs in time polynomial in its seed length. Instead, it suffices to require that the generator runs in time exponential in its seed length, because we are incurring such an overhead anyhow due to the scanning of all possible seeds. Furthermore, in this context, the running-time of the generator may be larger than the running time of the algorithm, which means that the generator need only fool distinguishers that take less steps than the generator. These considerations motivate the following definition.

8.4.1 Definition

Recall that in order to "derandomize" a probabilistic polynomial-time algorithm A, we first obtain a functionally equivalent algorithm A_G (as in Construction 8.2) that has (significantly) smaller randomness complexity. Algorithm A_G has to maintain A's input-output behavior on all (but finitely many) inputs. Thus, the set of the relevant distinguishers (considered in the proof of Theorem 8.13) is the set of all possible circuits obtained from A by hard-wiring each of the possible inputs. Such a circuit, denoted C_x, emulates the execution of algorithm A on input x, when using the circuit's input as the algorithm's internal coin tosses (i.e., $A(x,r) = C_x(r)$). Furthermore, the size of C_x is quadratic in the running-time of A on input x, and the length of the input to C_x is linear in the running-time of A (on input x).[17] Thus, the size of C_x is quadratic in the length of its own input, and the pseudorandom generator in use (i.e., G) needs to fool each such circuit. Recalling that we may allow the generator to run in exponential time (in the length of its own input)[18], we arrive at the following definition.

Definition 8.14 (pseudorandom generator for derandomizing $\mathrm{BP}\textsc{time}(\cdot)$)[19]: *Let $\ell : \mathbb{N} \to \mathbb{N}$ be a 1-1 function. A* canonical derandomizer of stretch ℓ *is a deterministic algorithm G of time complexity upper-bounded by* $\mathrm{poly}(2^k \cdot \ell(k))$ *such that for every circuit D_k of size $\ell(k)^2$ it holds that*

$$|\,\mathsf{Pr}[D_k(G(U_k)) = 1] \;-\; \mathsf{Pr}[D_k(U_{\ell(k)}) = 1]\,| \;<\; \frac{1}{6}\,. \tag{8.9}$$

The circuits D_k are potential distinguishers, which are given inputs of length $\ell(k)$. When seeking to derandomize an algorithm A of time-complexity t, the aforementioned $\ell(k)$-bit long inputs represent possible random-inputs of A when invoked on a generic (primary) input of length $n = t^{-1}(\ell(k))$. That is, letting $D_k(r) = A(x,r)$ for some choice of $x \in \{0,1\}^n$, where $|r| = t(n) = \ell(k)$, Eq. (8.9) implies that $A_G(x)$ *maintains the majority vote of* $A(x)$. The straightforward deterministic emulation of A_G takes time $2^k \cdot (\mathrm{poly}(2^k \cdot \ell(k)) + t(n))$, which is upper-bounded by $\mathrm{poly}(2^k \cdot \ell(k)) = \mathrm{poly}(2^{\ell^{-1}(t(n))} \cdot t(n))$. The following proposition is easy to establish.

[17] Indeed, we assume that algorithm A is represented as a Turing machine and refer to the standard emulation of Turing machines by circuits (as underlying the proof of Theorem 2.20). Thus, the aforementioned circuit C_x has size that is at most quadratic (and in fact even almost-linear [168]) in the running-time of A on input x, which in turn means that C_x has size that is at most quadratic (or almost linear) in the length of its own input. We note that most sources use the fictitious convention by which the circuit size equals the length of its input, which can be justified by considering a suitably padded input.

[18] Actually, in Definition 8.14 we allow the generator to run in time $\mathrm{poly}(2^k\ell(k))$, rather than $\mathrm{poly}(2^k)$. This is done in order not to rule out trivially generators of super-exponential stretch (i.e., $\ell(k) = 2^{\omega(k)}$). However (see Exercise 8.15), the condition in Eq. (8.9) does not allow for super-exponential stretch, and so in retrospect the two formulations are equivalent (because $\mathrm{poly}(2^k\ell(k)) = \mathrm{poly}(2^k)$ for $\ell(k) = 2^{O(k)}$).

[19] Fixing a model of computation, we denote by $\mathrm{BP}\textsc{time}(t)$ the class of decision problems that are solvable by a randomized algorithm of time complexity t that has two-sided error $1/3$. Using $1/6$ as the "threshold distinguishing gap" (in Eq. (8.9)) guarantees that if $\mathsf{Pr}[D_k(U_{\ell(k)}) = 1] \geq 2/3$ (resp., $\mathsf{Pr}[D_k(U_{\ell(k)}) = 1] \leq 1/3$) then $\mathsf{Pr}[D_k(G(U_k)) = 1] > 1/2$ (resp., $\mathsf{Pr}[D_k(G(U_k)) = 1] < 1/2$). Note that $|G(s)| = \ell(|s|)$ is implied by Eq. (8.9).

Proposition 8.15 *If there exists a canonical derandomizer of stretch ℓ then, for every time-constructible $t : \mathbb{N} \to \mathbb{N}$, it holds that* $\mathrm{BP}\textsc{time}(t) \subseteq \mathrm{D}\textsc{time}(T)$*, where* $T(n) = \mathrm{poly}(2^{\ell^{-1}(t(n))} \cdot t(n))$.

Proof Sketch: Just follow the proof of Theorem 8.13, noting that the adequate value of k (i.e., $k = \ell^{-1}(t(n))$) can be determined easily (e.g., by invoking $G(1^i)$ for $i = 1, ..., k$, using the fact that $\ell : \mathbb{N} \to \mathbb{N}$ is 1-1). Note that the complexity of the deterministic procedure is dominated by the 2^k invocations of $A_G(x, s) = A(x, G(s))$, where $s \in \{0,1\}^{\ell^{-1}(t(|x|))}$, and each of these invocations takes time $\mathrm{poly}(2^k \cdot \ell(k)) + t(n) = \mathrm{poly}(2^{\ell^{-1}(t(n))} \cdot t(n))$. Using the hypothesis $|\mathsf{Pr}[A(x, U_{\ell(k)})=1] - (1/2)| \geq 1/6$, it follows that the majority vote of A_G equals 1 (equiv., $\mathsf{Pr}[A_G(x, U_k)=1] > 1/2$) if and only if $\mathsf{Pr}[A(x, U_{\ell(k)})=1] > 1/2$ (equiv., $\mathsf{Pr}[A(x, U_{\ell(k)})=1] > 1/2$). Indeed, the implication is due to Eq. (8.9), when applied to the circuit $C_x(r) = A(x, r)$ (which has size at most $|r|^2$). $\square$

The goal. In light of Proposition 8.15, we seek canonical derandomizers with stretch that is as big as possible. The stretch cannot be super-exponential (i.e., it most hold that $\ell(k) = O(2^k)$), because there exists a circuit of size $O(2^k \cdot \ell(k))$ that violates Eq. (8.9) (see Exercise 8.15) whereas for $\ell(k) = \omega(2^k)$ it holds that $O(2^k \cdot \ell(k)) < \ell(k)^2$. Thus, our goal is to construct canonical derandomizer with stretch $\ell(k) = 2^{\Omega(k)}$. Such canonical derandomizers will allow for a "full derandomization of $\mathcal{BPP}$":

Theorem 8.16 *If there exists a canonical derandomizer of stretch $\ell(k) = 2^{\Omega(k)}$, then $\mathcal{BPP} = \mathcal{P}$.*

Proof: Using Proposition 8.15, we get $\mathrm{BP}\textsc{time}(t) \subseteq \mathrm{D}\textsc{time}(T)$, where $T(n) = \mathrm{poly}(2^{\ell^{-1}(t(n))} \cdot t(n)) = \mathrm{poly}(t(n))$. $\blacksquare$

Reflections. We stress that a canonical derandomizer G was defined in a way that allows it to have time complexity t_G that is larger than the size of the circuits that it fools (i.e., $t_G(k) > \ell(k)^2$ is allowed). Furthermore, $t_G(k) > 2^k$ was also allowed. Thus, if indeed $t_G(k) = 2^{\Omega(k)}$ (as is the case in Section 8.4.2) then $G(U_k)$ *can be distinguished from $U_{\ell(k)}$ in time $2^k \cdot t_G(k) = \mathrm{poly}(t_G(k))$* by trying all possible seeds.[20] In contrast, for a general-purpose pseudorandom generator G (as discussed in Section 8.3) it holds that $t_G(k) = \mathrm{poly}(k)$, while *for every polynomial p it holds that $G(U_k)$ is indistinguishable from $U_{\ell(k)}$ in time $p(t_G(k))$.*

8.4.2 Construction

The fact that canonical derandomizers are allowed to be more complex than the corresponding distinguisher makes *some* of the techniques of Section 8.3 inapplicable in the current context. For example, the stretch function cannot be amplified

[20] Note that this does not contradict the hypothesis that G is a canonical derandomizer because in this case $2^k \cdot t_G(k) > \ell(k)^2$.

as in Section 8.3.4. On the other hand, the techniques developed below are inapplicable to Section 8.3. Amazingly enough, the pseudorandomness (or rather the next-bit unpredictability) of the following generators hold even when the "observer" is given the seed itself. (This fact capitalizes on the fact that the observer's time-complexity does not allow for running the generator.)

As in Section 8.3.5, the construction presented next transforms computational difficulty into pseudorandomness, except that here both computational difficulty and pseudorandomness are of a somewhat different form than in Section 8.3.5. Specifically, here we use Boolean predicates that are computable in exponential-time but are T-inapproximable for some exponential function T (see Definition 7.9 in Section 7.2). That is, for constants $c,\varepsilon > 0$ and all but finitely many m, the (residual) predicate $f:\{0,1\}^m \to \{0,1\}$ is computable in time 2^{cm} but for any circuit C of size $2^{\varepsilon m}$ it holds that $\mathsf{Pr}[C(U_m) = f(U_m)] < \frac{1}{2} + 2^{-\varepsilon m}$. (Needless to say, $\varepsilon < c$.) Recall that such predicates exist under the assumption that $\mathcal{E}$ has (almost-everywhere) exponential circuit complexity (see Theorem 7.19 for an exact formulation). With these preliminaries, we turn to the construction of canonical derandomizers with exponential stretch.

Construction 8.17 (The Nisan-Wigderson Construction):[21] *Let $f:\{0,1\}^m \to \{0,1\}$ and $S_1,...,S_\ell$ be a sequence of m-subsets of $\{1,...,k\}$. Then, for $s \in \{0,1\}^k$, we let*

$$G(s) \stackrel{\text{def}}{=} f(s_{S_1}) \cdots f(s_{S_\ell}) \tag{8.10}$$

where s_S denotes the projection of s on the bit locations in $S \subseteq \{1,...,|s|\}$; that is, for $s = \sigma_1 \cdots \sigma_k$ and $S = \{i_1,...,i_m\}$, we have $s_S = \sigma_{i_1} \cdots \sigma_{i_m}$.

Letting k vary and $\ell, m : \mathbb{N} \to \mathbb{N}$ be functions of k, we wish G to be a canonical derandomizer and $\ell(k) = 2^{\Omega(k)}$. One (obvious) necessary condition for this to happen is that the sets must be distinct, and hence $m(k) = \Omega(k)$; consequently, f must be computable in exponential-time. Furthermore, the sequence of sets $S_1,...,S_{\ell(k)}$ must be constructible in $\text{poly}(2^k)$ time. Intuitively, it is desirable to use a set system with small pairwise intersections (because this restricts the overlap among the various inputs to which f is applied), and a function f that is strongly inapproximable (i.e., T-inapproximable for some exponential function T). Interestingly, these conditions are essentially sufficient.

Theorem 8.18 (analysis of Construction 8.17): *Let $\alpha,\beta,\gamma,\varepsilon > 0$ be constants satisfying $\varepsilon > (2\alpha/\beta) + \gamma$, and $\ell, m, T : \mathbb{N} \to \mathbb{N}$ satisfy $\ell(k) = 2^{\alpha k}$, $m(k) = \beta k$, and $T(n) = 2^{\varepsilon n}$. Suppose that the following two conditions hold:*

1. *There exists an exponential-time computable function $f:\{0,1\}^* \to \{0,1\}$ that is T-inapproximable.* (See Definition 7.9.)

2. *There exists an exponential-time computable function $S : \mathbb{N} \times \mathbb{N} \to 2^{\mathbb{N}}$ such that*

[21] Given the popularity of the term, we deviate from our convention of not specifying credits in the main text. This construction originates in [161, 164].

(a) *For every k and $i = 1, ..., \ell(k)$, it holds that $S(k,i) \subseteq [k]$ and $|S(k,i)| = m(k)$.*

(b) *For every k and $i \neq j$, it holds that $|S(k,i) \cap S(k,j)| \leq \gamma \cdot m(k)$.*

Then using G as defined in Construction 8.17, with $S_i = S(k,i)$, yields a canonical derandomizer with stretch ℓ.

Before proving Theorem 8.18 we note that, for any $\gamma > 0$, a function S as in Condition 2 does exist with some $m(k) = \Omega(k)$ and $\ell(k) = 2^{\Omega(k)}$; see Exercise 8.16. Combining such S with Theorems 7.19 and 8.18, we obtain a canonical derandomizer with exponential stretch based on the assumption that $\mathcal{E}$ has (almost-everywhere) exponential circuit complexity.[22] Combining this with Theorem 8.16, we get the first item of the following theorem.

Theorem 8.19 (Derandomization of BPP, revisited):

1. *Suppose that there exists a set $S \in \mathcal{E}$ having almost-everywhere exponential circuit complexity* (i.e., there exists a constant $\varepsilon > 0$ such that, for all but finitely many m's, any circuit that correctly decides S on $\{0,1\}^m$ has size at least $2^{\varepsilon m}$). *Then, $\mathcal{BPP} = \mathcal{P}$.*

2. *Suppose that for every polynomial p there exists a set $S \in \mathcal{E}$ having circuit complexity that is almost-everywhere greater than p. Then $\mathcal{BPP}$ is contained in $\cap_{\varepsilon>0}\mathrm{DTIME}(t_\varepsilon)$, where $t_\varepsilon(n) \stackrel{\mathrm{def}}{=} 2^{n^\varepsilon}$.*

Part 2 is proved (in Exercise 8.20) by using a generalization of Theorem 8.18, which in turn is provided in Exercise 8.19. We note that Part 2 of Theorem 8.19 supersedes Theorem 8.13 (see Exercise 7.16). The two parts of Theorem 8.19 exhibit two extreme cases: Part 1 (often referred to as the "high end") assumes an extremely strong circuit lower-bound and yields "full derandomization" (i.e., $\mathcal{BPP} = \mathcal{P}$), whereas Part 2 (often referred to as the "low end") assumes an extremely weak circuit lower-bound and yields weak but meaningful derandomization. Intermediate results (relying on intermediate lower-bound assumptions) can be obtained analogous to Exercise 8.20, but tight trade-offs are obtained differently (cf., [212]).

Proof of Theorem 8.18: Using the time complexity upper-bounds on f and S, it follows that G can be computed in exponential time. Our focus is on showing that $\{G(U_k)\}$ cannot be distinguished from $\{U_{\ell(k)}\}$ by circuits of size $\ell(k)^2$; specifically, that G satisfies Eq. (8.9). In fact, we will prove that this holds for $G'(s) = s \cdot G(s)$; that is, G fools such circuits even if they are given the seed as auxiliary input. (Indeed, these circuits are smaller than the running time of G, and so they cannot just evaluate G on the given seed.)

[22] Specifically, starting with a function having circuit complexity at least $\exp(\varepsilon_0 m)$, we apply Theorem 7.19 and obtain a T-inapproximable predicate for $T(m) = 2^{\varepsilon m}$, where the constant $\varepsilon \in (0, \varepsilon_0)$ depends on the constant ε_0. Next, we set $\gamma = \varepsilon/2$ and invoke Exercise 8.16, which determines $\alpha, \beta > 0$ such that $\ell(k) = 2^{\alpha k}$ and $m(k) = \beta k$. Note that (by possibly decreasing α) we get $(2\alpha/\beta) + \gamma < \varepsilon$.

We start by presenting the intuition underlying the proof. As a warm-up suppose that the sets (i.e., $S(k,i)$'s) used in the construction are disjoint. In such a case (which is indeed impossible because $k < \ell(k) \cdot m(k)$), the pseudorandomness of $G(U_k)$ would follow easily from the inapproximability of f, because in this case G consists of applying f to non-overlapping parts of the seed (see Exercise 8.18). In the actual construction being analyzed here, the sets (i.e., $S(k,i)$'s) are not disjoint but have relatively small pairwise intersection, which means that G applies f on parts of the seed that have relatively small overlap. Intuitively, such small overlaps guarantee that the values of f on the corresponding inputs are "computationally independent" (i.e., having the value of f at some inputs $x_1, ..., x_i$ does not help in approximating the value of f at another input x_{i+1}). This intuition will be backed by showing that, when fixing all bits that do not appear in the target input (i.e., in x_{i+1}), the former values (i.e., $f(x_1), ..., f(x_i)$) can be computed at a relatively small computational cost. With this intuition in mind, we now turn to the actual proof.

The proof that G' fools circuits of size $\ell(k)^2$ utilizes the relation between pseudorandomness and unpredictability. Specifically, as detailed in Exercise 8.17, any circuit that distinguishes $G'(U_k)$ from $U_{\ell(k)+k}$ with gap $1/6$, yields a next-bit predictor of similar size that succeeds in predicting the next bit with probability at least $\frac{1}{2} + \frac{1}{6\ell'(k)} > \frac{1}{2} + \frac{1}{7\ell(k)}$, where the factor of $\ell'(k) = \ell(k) + k < (1 + o(1))\ell(k)$ is introduced by the hybrid technique (cf. Eq. (8.5)). Furthermore, given the non-uniform setting of the current proof, we may fix a bit location $i + 1$ for prediction, rather than analyzing the prediction at a random bit location. Indeed, $i \geq k$ must hold, because the first k bits of $G'(U_k)$ are uniformly distributed. In the rest of the proof, we transform such a predictor into a circuit that approximates f better than allowed by the hypothesis (regarding the inapproximability of f).

Assuming that a small circuit C' can predict the $i+1^{\text{st}}$ bit of $G'(U_k)$, when given the previous i bits, we construct a small circuit C for approximating $f(U_{m(k)})$ on input $U_{m(k)}$. The point is that the $i+1^{\text{st}}$ bit of $G'(s)$ equals $f(s_{S(k,j+1)})$, where $j = i - k \geq 0$, and so C' approximates $f(s_{S(k,j+1)})$ based on $s, f(s_{S(k,1)}), ..., f(s_{S(k,j)})$, where $s \in \{0,1\}^k$ is uniformly distributed. Note that this is the type of thing that we are after, except that the circuit we seek may only get $s_{S(k,j+1)}$ as input.

The first observation is that C' maintains its advantage when we fix the best choice for the bits of s that are not at bit locations $S_{j+1} = S(k, j+1)$ (i.e., the bits $s_{[k]\setminus S_{j+1}}$). That is, by an averaging argument, it holds that

$$\begin{aligned} &\max_{s' \in \{0,1\}^{k-m(k)}} \{\mathsf{Pr}_{s\in\{0,1\}^k}[C'(s, f(s_{S_1}), ..., f(s_{S_j})) = f(s_{S_{j+1}}) \,|\, s_{[k]\setminus S_{j+1}} = s']\} \\ &\geq \; p' \stackrel{\text{def}}{=} \mathsf{Pr}_{s\in\{0,1\}^k}[C'(s, f(s_{S_1}), ..., f(s_{S_j})) = f(s_{S_{j+1}})]. \end{aligned}$$

Recall that by the hypothesis $p' > \frac{1}{2} + \frac{1}{7\ell(k)}$. Hard-wiring the fixed string s' into C', and letting $\pi(x)$ denote the (unique) string s satisfying $s_{S_{j+1}} = x$ and $s_{[k]\setminus S_{j+1}} = s'$, we obtain a circuit C'' that satisfies

$$\mathsf{Pr}_{x\in\{0,1\}^m}[C''(x, f(\pi(x)_{S_1}), ..., f(\pi(x)_{S_j})) = f(x)] \;\geq\; p'.$$

The circuit C'' is almost what we seek. The only problem is that C'' gets as input not only x, but also $f(\pi(x)_{S_1}), ..., f(\pi(x)_{S_j})$, whereas we seek an approximator of $f(x)$ that only gets x.

The key observation is that each of the "missing" values $f(\pi(x)_{S_1}), ..., f(\pi(x)_{S_j})$ depend only on a relatively small number of the bits of x. This fact is due to the hypothesis that $|S_t \cap S_{j+1}| \leq \gamma \cdot m(k)$ for $t = 1, ..., j$, which means that $\pi(x)_{S_t}$ is an $m(k)$-bit long string in which $m_t \stackrel{\text{def}}{=} |S_t \cap S_{j+1}|$ bits are projected from x and the rest are projected from the *fixed* string s'. Thus, given x, the value $f(\pi(x)_{S_t})$ can be computed by a (trivial) circuit of size $\widetilde{O}(2^{m_t})$; that is, by a circuit implementing a look-up table on m_t bits. Using all these circuits (together with C''), we will obtain the desired approximator of f. Details follow.

We obtain the desired circuit, denoted C, that T-approximates f as follows. The circuit C depends on the index j and the string s' that are fixed as in the foregoing analysis. On input $x \in \{0,1\}^m$, the circuit C computes the values $f(\pi(x)_{S_1}), ..., f(\pi(x)_{S_j})$, invokes C'' on input x and these values, and outputs the answer as a guess for $f(x)$. That is,

$$C(x) = C''(x, f(\pi(x)_{S_1}), ..., f(\pi(x)_{S_j})) = C'(\pi(x), f(\pi(x)_{S_1}), ..., f(\pi(x)_{S_j})).$$

By the foregoing analysis, $\Pr_x[C(x) = f(x)] \geq p' > \frac{1}{2} + \frac{1}{T(m)}$, where the second inequality is due to $T(m(k)) = 2^{\varepsilon m(k)} = 2^{\varepsilon \beta k} \gg 2^{2\alpha k} \gg 7\ell(k)$. The size of C is upper-bounded by $\ell(k)^2 + \ell(k) \cdot \widetilde{O}(2^{\gamma \cdot m(k)}) \ll \widetilde{O}(\ell(k)^2 \cdot 2^{\gamma \cdot m(k)}) \ll T(m(k))$, where the second inequality is due to $T(m(k)) = 2^{\varepsilon m(k)} \gg \widetilde{O}(2^{2\alpha \cdot k + \gamma \cdot m(k)})$ and $\ell(k) = 2^{\alpha k}$. Thus, we derived a contradiction to the hypothesis that f is T-inapproximable. ■

8.4.3 Variants and a conceptual discussion

We start this section by discussing a general framework that underlies Construction 8.17 and end it with a conceptual discussion regarding derandomization.

8.4.3.1 Construction 8.17 as a general framework

The Nisan–Wigderson Construction (i.e., Construction 8.17) is actually a general framework, which can be instantiated in various ways. Some of these instantiations are briefly reviewed next, and are based on an abstraction of the construction as well as of its analysis.

We first note that the generator described in Construction 8.17 consists of a generic algorithmic scheme that can be instantiated with any predicate f. Furthermore, this algorithmic scheme, denoted G, is actually an *oracle machine* that makes (non-adaptive) queries to the function f, and thus the combination may be written as G^f. Likewise, the proof of pseudorandomness of G^f yields a (non-uniform) circuit C that given oracle access to any distinguisher yields an approximation procedure for f. The circuit C does depends on f (but in a restricted way), and uses the distinguisher as a black-box. Specifically, C contains look-up tables for computing functions obtained from f by fixing some of the input bits (i.e., look-up tables for the functions $f(\pi(\cdot)_{S_t})$'s).

Derandomization of constant-depth circuits. In this case we instantiate Construction 8.17 using the `parity` function in the role of the inapproximable predicate f, noting that `parity` is indeed inapproximable by "small" constant-depth circuits. With an adequate setting of parameters we obtain pseudorandom generators with stretch $\ell(k) = \exp(k^{1/O(1)})$ that fool "small" constant-depth circuits (see [161]). The analysis of this construction proceeds very much like the proof of Theorem 8.18. One important observation is that incorporating the (straightforward) circuits that compute $f(\pi(x)_{S_t})$ into the distinguishing circuit only increases its depth by two levels. Specifically, the circuit C uses depth-two circuits that compute the values $f(\pi(x)_{S_t})$'s, and then obtains a prediction of $f(x)$ by using these values in its (single) invocation of the (given) distinguisher.

The resulting pseudorandom generator, which use a seed of polylogarithmic length (equiv., $\ell(k) = \exp(k^{1/O(1)})$), can be used for derandomizing $\mathcal{RAC}^0$ (i.e., random $\mathcal{AC}^0$), analogously to Theorem 8.16. In other words, we can *deterministically* approximate, in quasi-polynomial-time and up-to an additive error, the fraction of inputs that satisfy a given (constant-depth) circuit. Specifically, for any constant d, given a depth-d circuit C, we can deterministically approximate the fraction of the inputs that satisfy C (i.e., cause C to evaluate to 1) to within any *additive constant error*[23] in time $\exp(\text{poly}(\log |C|))$, where the polynomial depends on d. Providing a deterministic polynomial-time approximation, even in the case $d = 2$ (i.e., CNF/DNF formulae) is an open problem.

Derandomization of probabilistic proof systems. A different (and more surprising) instantiation of Construction 8.17 utilizes predicates that are inapproximable by small *circuits having oracle access to* $\mathcal{NP}$. The result is a pseudorandom generator robust against two-move public-coin interactive proofs (which are as powerful as constant-round interactive proofs (see §9.1.3.1)). The key observation is that the analysis of Construction 8.17 provides a black-box procedure for approximating the underlying predicate when given oracle access to a distinguisher (and this procedure is valid also in case the distinguisher is a non-deterministic machine). Thus, under suitably strong (and yet plausible) assumptions, constant-round interactive proofs collapse to $\mathcal{NP}$. We note that a stronger result, which deviates from the foregoing framework, has been subsequently obtained (cf. [156]).

Construction of randomness extractors. An even more radical instantiation of Construction 8.17 was used to obtain explicit constructions of randomness extractors (see Appendix D.4). In this case, the predicate f is viewed as (an error

[23] We mention that in the special case of approximating the number of satisfying assignment of a DNF formula, *relative error* approximations can be obtained by employing a deterministic reduction to the case of additive constant error (see §6.2.2.1). Thus, using a pseudorandom generator that fools DNF formulae, we can deterministically obtain a relative (rather than additive) error approximation to the number of satisfying assignment in a given DNF formula.

correcting encoding of) a somewhat random function, and the construction makes sense because it refers to f in a black-box manner. In the analysis we rely on the fact that f can be approximated by combining relatively little information (regarding f) with (black-box access to) a distinguisher for G^f. For further details see Appendix D.4.

8.4.3.2 A conceptual discussion regarding derandomization

Part 1 of Theorem 8.19 is often summarized by saying that (under some reasonable assumptions) *randomness is useless*. We believe that this interpretation is wrong even within the restricted context of traditional complexity classes, and is bluntly wrong if taken outside of the latter context. Let us elaborate.

Taking a closer look at the proof of Theorem 8.16 (which underlies Theorem 8.19), we note that a randomized algorithm A of time complexity t is emulated by a deterministic algorithm A' of time complexity $t' = \text{poly}(t)$. Further noting that $A' = A_G$ invokes A (as well as the canonical derandomizer G) for a number of times that must exceed t, we infer that $t' > t^2$ must hold. Thus, derandomization via (Part 1 of) Theorem 8.19 is not really for free.

More importantly, we note that derandomization is not possible in various distributed settings, when both parties may protect their conflicting interests by employing randomization. Notable examples include most cryptographic primitives (e.g., encryption) as well as most types of probabilistic proof systems (e.g., PCP). For further discussion see Chapter 9 and Appendix C. Additional settings where randomness makes a difference (either between impossibility and possibility or between formidable and affordable cost) include distributed computing (see [15]), communication complexity (see [139]), parallel architectures (see [142]), sampling (see Appendix D.3), and property testing (see Section 10.1.2).

8.5 Space-Bounded Distinguishers

In the previous two sections we have considered generators that output sequences that look random to any efficient procedures, where the latter were modeled by time-bounded computations. Specifically, in Section 8.3 we considered indistinguishability by polynomial-time procedures. A finer classification of time-bounded procedures is obtained by considering their space-complexity (i.e., restricting the space-complexity of time-bounded computations). This leads to the notion of pseudorandom generators that fool space-bounded distinguishers. Interestingly, in contrast to the notions of pseudorandom generators that were considered in Sections 8.3 and 8.4, the existence of pseudorandom generators that fool space-bounded distinguishers can be established without relying on computational assumptions.

8.5.1 Definitional issues

Unfortunately, natural notions of space-bounded computations are quite subtle, especially when non-determinism or randomization are concerned (see Sections 5.3 and 6.1.4, respectively). Two major issues are *time bounds* and *access to the random tape.*

1. Time bound: The question is whether or not one restricts the space-bounded machines to run in time-complexity that is at most exponential in the space-complexity.[24] Recall that such an upper-bound follows automatically in the deterministic case (Theorem 5.3), and can be assumed without loss of generality in the non-deterministic case (see Section 5.3.2), *but it does not necessarily hold in the randomized case* (see §6.1.4.1).

   As in Section 6.1.4, we do postulate the aforementioned time-bound.

2. Access to the random tape: The question is whether whether the space-bounded machine has one-way or two-way access to the randomness tape. (Allowing two-way access means that the randomness is recorded for free; that is, without being accounted for in the space-bound; see discussions in Sections 5.3 and 6.1.4.) Recall that one-way access to the randomness tape corresponds to the natural model of on-line randomized machine (which determines its moves based on its internal coin tosses).

   Again, as in Section 6.1.4, we consider one-way access.[25]

In accordance with the resulting definition of randomized space-bounded computation, we consider space-bounded distinguishers that have a one-way access to the input sequence that they examine. Since all known constructions remain valid also when these distinguishers are non-uniform (and since non-uniform distinguishers arise anyhow in derandomization), we use this stronger notion here.[26]

In the context of non-uniform algorithms that have one-way access to their input, we may assume, without loss of generality, that the running-time of such algorithms equals the length of their input, denoted $\ell = \ell(k)$. Thus, we define a non-uniform machine of space $s:\mathbb{N}\rightarrow\mathbb{N}$ as a family, $\{D_k\}_{k\in\mathbb{N}}$, of directed layered graphs such that D_k has at most $2^{s(k)}$ vertices at each layer, and labeled directed edges from each layer to the next layer.[27] Each vertex has two (possibly parallel)

[24]Alternatively, one can ask whether these machines must always halt or only halt with probability approaching 1. It can be shown that the only way to ensure "absolute halting" is to have time-complexity that is at most exponential in the space-complexity.

[25]We note that the fact that we restrict our attention to one-way access is instrumental in obtaining space-robust generators without making intractability assumptions. Analogous generators for two-way space-bounded computations would imply hardness results of a breakthrough nature in the area.

[26]We note that these non-uniform space-bounded distinguishers correspond to branching programs of width that is exponential in the space-bound. Furthermore, these branching programs read their input in a fixed predetermined order (which is determined by the designer of the generator).

[27]Note that the space bound of the machine is stated in terms of a parameter k, rather than in terms of the length of its input. In the sequel this parameter will be set to the length of a seed to a pseudorandom generator. We warn that our presentation here is indeed non-standard for this area. To compensate for this, we will also state the consequences in the standard format.

outgoing directed edges, one labeled 0 and the other labeled 1, and there is a single vertex in the first layer of D_k. The result of the computation of such a machine, on an input of adequate length (i.e., length ℓ where D_k has $\ell+1$ layers), is defined as the vertex (in last layer) reached when following the sequence of edges that are labeled by the corresponding bits of the input. That is, on input $x = x_1 \cdots x_\ell$, for $i = 1, ..., \ell$, we move from the vertex reached in the i^{th} layer by using the outgoing edge labeled x_i (thus reaching a vertex in the $i+1^{\text{st}}$ layer). Using a fixed partition of the vertices of the last layer, this defines a natural notion of decision (by D_k); that is, we write $D_k(x) = 1$ if on input x machine D_k reached a vertex that belongs to the first part of the aforementioned partition.

Definition 8.20 (Indistinguishability by space-bounded machines):

- *For a non-uniform machine, $\{D_k\}_{k\in\mathbb{N}}$, and two probability ensembles, $\{X_k\}_{k\in\mathbb{N}}$ and $\{Y_k\}_{k\in\mathbb{N}}$, the function $d:\mathbb{N}\to[0,1]$ defined as*

$$d(k) \stackrel{\text{def}}{=} |\mathsf{Pr}[D_k(X_k) = 1] - \mathsf{Pr}[D_k(Y_k) = 1]|$$

  is called the distinguishability-gap *of $\{D_k\}$ between the two ensembles.*

- *Let $s : \mathbb{N}\to\mathbb{N}$ and $\varepsilon : \mathbb{N} \to [0,1]$. A probability ensemble, $\{X_k\}_{k\in\mathbb{N}}$, is called* (s,ε)-pseudorandom *if for any (non-uniform) machine of space $s(\cdot)$, the distinguishability-gap of the machine between $\{X_k\}_{k\in\mathbb{N}}$ and the corresponding uniform ensemble (i.e., $\{U_{|X_k|}\}_{k\in\mathbb{N}}$) is at most $\varepsilon(\cdot)$.*

- *A deterministic algorithm G of stretch function ℓ is called a* (s,ε)-pseudorandom generator *if the ensemble $\{G(U_k)\}_{k\in\mathbb{N}}$ is (s,ε)-pseudorandom. That is, every non-uniform machine of space $s(\cdot)$ has a distinguishing-gap of at most $\varepsilon(\cdot)$ between $\{G(U_k)\}_{k\in\mathbb{N}}$ and $\{U_{\ell(k)}\}_{k\in\mathbb{N}}$.*

Thus, when using a random seed of length k, a (s,ε)-pseudorandom generator outputs a sequence of length $\ell(k)$ that looks random to observers having space $s(k)$. (Setting $m = s(k)$, we have $k = s^{-1}(m)$ and $\ell(k) = \ell(s^{-1}(m))$.)

8.5.2 Two Constructions

In contrast to the case of pseudorandom generators that fool time-bounded distinguishers, pseudorandom generators that fool space-bounded distinguishers can be established without relying on any computational assumption. The following two constructions exhibit two extreme cases of a general trade-off between the space bound of the potential distinguisher and the stretch function of the generator.[28] We start with an attempt to maximize the stretch.

Theorem 8.21 (exponential stretch with quadratic length seed): *For every space constructible function $s:\mathbb{N}\to\mathbb{N}$, there exists a $(s, 2^{-s})$-pseudorandom generator of*

[28] These two results have been "interpolated" in [11]: There exists a parameterized family of "space fooling" pseudorandom generators that includes both results as extreme special cases.

stretch function $\ell(k) = 2^{k/O(s(k))} \leq 2^{s(k)}$. *Furthermore, the generator works in space that is linear in the length of the seed, and in time that is linear in the stretch function.*

In other words, for every $t \leq m$, we have a generator that takes a random seed of length $k = O(t \cdot m)$ and produce a sequence of length 2^t that looks random to any (non-uniform) machine of space m (up to a distinguishing-gap of 2^{-m}). In particular, using a random seed of length $k = O(m^2)$, one can produce a sequence of length 2^m that looks random to any (non-uniform) machine of space m. Thus, *one may replace random sequences used by any space-bounded computation, by sequences that are efficiently generated from random seeds of length quadratic in the space bound*. The common instantiation is for log-space machines. In §8.5.2.2, we apply Theorem 8.21 (and its underlying ideas) for the derandomization of space complexity classes such as $\mathcal{BPL}$ (i.e., the log-space analogue of $\mathcal{BPP}$).

We now turn to the case where one wishes to maximize the space bound of potential distinguishers. We warn that Theorem 8.22 only guarantees a subexponential distinguishing gap (rather than the exponential distinguishing gap guaranteed in Theorem 8.21). This warning is voiced because failing to recall this limitation has led to errors in the past.

Theorem 8.22 (polynomial stretch with linear length seed): *For any polynomial p and for some $s(k) = k/O(1)$, there exists a $(s, 2^{-\sqrt{s}})$-pseudorandom generator of stretch function p. Furthermore, the generator works in linear-space and polynomial-time* (both stated in terms of the length of the seed).

In other words, we have a generator that takes a random seed of length $k = O(m)$ and produce a sequence of length $\text{poly}(m)$ that looks random to any (non-uniform) machine of space m. Thus, one may *convert any randomized computation utilizing polynomial-time and linear-space into a functionally equivalent randomized computation of similar time and space complexities that uses only a linear number of coin tosses.*

8.5.2.1 Overviews of the proofs of Theorems 8.21 and 8.22

In both cases, we start the proof by considering a generic space-bounded distinguisher and show that the input distribution that this distinguisher examines can be modified (from the uniform distribution into a pseudorandom one) without the distinguisher noticing the difference. This modification (or rather a sequence of modifications) yields a construction of a pseudorandom generator, which is only spelled-out at the end of argument.

Overview of the proof of Theorem 8.21.[29] The main technical tool used in this proof is the "mixing property" of pairwise independent hash functions (see Appendix D.2). A family of functions H_n, which map $\{0,1\}^n$ to itself, is called mixing

[29] A detailed proof appears in [162].

if for every pair of subsets $A, B \subseteq \{0,1\}^n$ for all but very few (i.e., $\exp(-\Omega(n))$ fraction) of the functions $h \in H_n$, it holds that

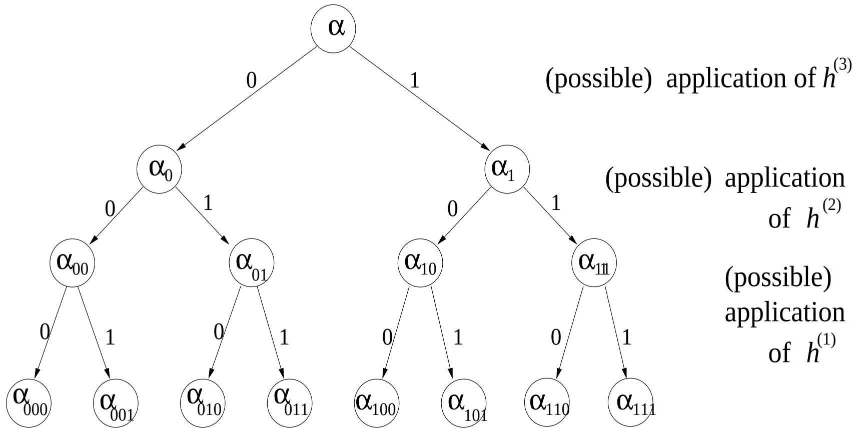
$$\mathsf{Pr}[U_n \in A \wedge h(U_n) \in B] \approx \frac{|A|}{2^n} \cdot \frac{|B|}{2^n} \tag{8.11}$$

where the approximation is up to an additive term of $\exp(-\Omega(n))$. (See the generalization of Lemma D.4, which implies that $\exp(-\Omega(n))$ can be set to $2^{-n/3}$.)

For any $s(k)$-space distinguisher D_k as in Definition 8.20, we consider an auxiliary "distinguisher" D'_k that is obtained by "contracting" every block of $n \stackrel{\text{def}}{=} \Theta(s(k))$ consecutive blocks layers in D_k, yielding a directed layered graph with $\ell' \stackrel{\text{def}}{=} \ell(k)/n < 2^{s(k)}$ layers (and $2^{s(k)}$ vertices in each layer). Specifically, in D'_k, each vertex has a directed edge going to each vertex of the next layer, and these edges are *labeled with* (possibly empty) *subsets of* $\{0,1\}^n$ that correspond to the set of corresponding n-paths in D_k (and in particular form a partition of $\{0,1\}^n$). The graph D'_k simulates D_k in the obvious manner; that is, the computation of D'_k on an input of length $\ell(k) = \ell' \cdot n$ is defined by breaking the input into consecutive blocks of length n and following the path of edges that are labeled by the subsets containing the corresponding block. Now, for each pair of neighboring vertices, u and v (in layers i and $i+1$, respectively), consider the label, $L_{u,v} \subseteq \{0,1\}^n$, of the edge going from u to v. Similarly, for a vertex w at layer $i+2$, we consider the label $L'_{v,w}$ of the edge from v to w. By Eq. (8.11), for all but very few of $h \in H_n$, it holds that

$$\mathsf{Pr}[U_n \in L_{u,v} \wedge h(U_n) \in L'_{v,w}] \approx \mathsf{Pr}[U_n \in L_{u,v}] \cdot \mathsf{Pr}[U_n \in L'_{v,w}]$$

where "very few" and $\approx$ are as in Eq. (8.11). Thus, for all but $\exp(-\Omega(n))$ fraction of the choices of $h \in H_n$, replacing the coins in the second block (i.e., used in transitions from layer $i+1$ to layer $i+2$) with the value of h applied to the outcomes of the coins used in the first block (i.e., in transitions from layer i to $i+1$), approximately maintains the probability that D'_k moves from u to w via v. Using a union bound (on all triplets (u,v,w) as in the foregoing), for all but $2^{3s(k)} \cdot \ell' \cdot \exp(-\Omega(n))$ fraction of the choices of $h \in H_n$, the foregoing replacement approximately maintains the probability that D'_k moves through any specific 2-edge path of D'_k. Using $\ell' < 2^{s(k)}$ and a suitable choice of $n = \Theta(s(k))$, we have $2^{3s(k)} \cdot \ell' \cdot \exp(-\Omega(n)) < \exp(-\Omega(n))$, and thus all but "few" functions $h \in H_n$ are good for approximating all these transition probabilities. (We stress that the same h can be used in all these approximations.) Thus, at the cost of extra $|h|$ random bits, we can reduce the number of true random coins used in transitions on D'_k by a factor of 2, without significantly affecting the final decision of D'_k (where again we use the fact that $\ell' \cdot \exp(-\Omega(n)) < \exp(-\Omega(n))$, which implies that the approximation errors do not accumulate to too much). In other words, at the cost of extra $|h|$ random bits, we can effectively contract the distinguisher to half its length. That is, fixing a good h (i.e., one that provides a good approximation to the transition probability over all $2^{3s(k)} \cdot \ell'$ 2-edge paths), we can replace the 2-edge paths in D'_k by edges in a new distinguisher D''_k such that r is in the set that labels



The output of the generator (on seed $\alpha, h^{(1)}, ..., h^{(t)}$) consists of the concatenation of the strings denoted $\alpha_{0^t}, ..., \alpha_{1^t}$, appearing in the leaves of the tree. For every $x \in \{0,1\}^*$ it holds that $\alpha_{x0} = \alpha_x$ and $\alpha_{x1} = h^{(t-|x|)}(\alpha_x)$. In particular, for $t = 3$, we have $\alpha_{011} = h^{(1)}(\alpha_{01})$, which equals $h^{(1)}(h^{(2)}(\alpha_0)) = h^{(1)}(h^{(2)}(\alpha))$, where $\alpha = \alpha_\lambda$.

Figure 8.3: The first generator that "fools" space-bounded machines.

the edge u–w in D''_k if and only if, for some v, the string r is in the label of the edge u–v in D'_k and $h(r)$ is in the label of the edge v–w (also in D'_k).

Repeating the process for a logarithmic (in the depth of D'_k) number of times we obtain a distinguisher that only examines n bits, at which point we stop. In total, we have used $t \stackrel{\text{def}}{=} \log_2(\ell'/n) < \log_2 \ell(k)$ random hash functions, denoted $h^{(1)}, ..., h^{(t)}$, which means that we can generate a sequence that fools the original D_k using a seed of length $n + t \cdot \log_2 |H_n|$ (see Figure 8.3 and Exercise 8.22). Using $n = \Theta(s(k))$ and an adequate family H_n (e.g., Construction D.3) yields the claimed seed length of $O(s(k) \cdot \log_2 \ell(k)) = k$. □

Overview of the proof of Theorem 8.22.[30] The main technical tool used in this proof is a suitable randomness extractor (as defined in §D.4.1.1), which is indeed a much more powerful tool than hashing functions. The basic idea is that when D_k is at some "distant" layer, say at layer t, it typically "knows" little about the random choices that led it there. That is, D_k has only $s(k)$ bits of memory, which leaves out $t - s(k)$ bits of "uncertainty" (or randomness) regarding the previous moves. Thus, much of the randomness that led D_k to its current state may be "re-used" (or "recycled"). To re-use these bits we need to extract *almost* uniform distribution on strings of sufficient length out of the aforementioned distribution over $\{0,1\}^t$ that has entropy[31] at least $t - s(k)$. Furthermore, such an extraction requires

[30] A detailed proof appears in [165].
[31] Actually, a stronger technical condition needs and can be imposed on the latter distribution.

some additional truly random bits, yet relatively few such bits. In particular, using $k' = \Omega(\log t)$ bits towards this end, the extracted bits are $\exp(-\Omega(k'))$ away from uniform.

The gain from the aforementioned recycling is significant if recycling is repeated sufficiently many times. Towards this end, we break the k-bit long seed into two parts, denoted $r' \in \{0,1\}^{k/2}$ and $(r_1, ..., r_{3\sqrt{k}})$, where $|r_i| = \sqrt{k}/6$, and set $n = k/3$. Intuitively, r' will be used for determining the first n steps, and it will be re-used (or recycled) together with r_i for determining the steps $i \cdot n + 1$ through $(i+1) \cdot n$. Looking at layer $i \cdot n$, we consider the information regarding r' that is known to D_k (at layer $i \cdot n$). Typically, the conditional distribution of r', given that we reached a specific vertex at layer $i \cdot n$, has (min-)entropy greater than $0.99 \cdot ((k/2) - s(k))$. Using r_i (as a seed of an extractor applied to r'), we can extract $0.9 \cdot ((k/2) - s(k) - o(k)) > k/3 = n$ bits that are almost-random (i.e., $2^{-\Omega(\sqrt{k})}$-close to U_n) with respect to D_k, and use these bits for determining the next n steps. Hence, using k random bits we are produce a sequence of length $(1 + 3\sqrt{k}) \cdot n > k^{3/2}$ that fools machines of space bound, say, $s(k) = k/10$. Specifically, using an extractor of the form $\mathrm{Ext} : \{0,1\}^{\sqrt{k}/6} \times \{0,1\}^{k/2} \to \{0,1\}^{k/3}$, we map the seed $(r', r_1, ..., r_{3\sqrt{k}})$ to the output sequence $(r', \mathrm{Ext}(r_1, r'), ..., \mathrm{Ext}(r_{3\sqrt{k}}, r'))$. Thus, we obtained a $(s, 2^{-\Omega(\sqrt{s})})$-pseudorandom generator of stretch function $\ell(k) = k^{3/2}$.

To obtain an arbitrary polynomial stretch rather than a specific polynomial stretch (i.e., $\ell(k) = k^{3/2}$) we repeatedly apply an adequate composition, to be outlined next. Suppose that G_1 is a (s_1, ε_1)-pseudorandom generator of stretch function ℓ_1 that works in linear space, and similarly for G_2 with respect to (s_1, ε_1) and ℓ_2. Then, we consider the following construction of a generator G:

1. On input $s \in \{0,1\}^k$, obtain $G_1(s)$, and parse it into consecutive blocks, each of length $k' = s_1(k)/O(1)$, denoted $r_1, ..., r_t$, where $t = \ell_1(k)/k'$.
2. Output the $t \cdot \ell_2(k')$-bit long sequence $G_2(r_1) \cdots G_2(r_t)$.

Note that $|G(s)| = \ell_1(k) \cdot \ell_2(k')/k'$, which for $s_1(k) = \Theta(k)$ yields $|G(s)| = \ell_1(k) \cdot \ell_2(\Omega(k))/O(k)$, which for polynomials ℓ_1 and ℓ_2 yields $|G(s)| = \ell_1(|s|) \cdot \ell_2(|s|)/O(|s|)$. We claim that G is a (s, ε)-pseudorandom generator, for $s(k) = \min(s_1(k)/2, s_2(\Omega(s_1(k)))$ and $\varepsilon(k) = \varepsilon_1(k) + \ell_1(k) \cdot \varepsilon_2(\Omega(s_1(k)))$. The proof uses a hybrid argument, which refers to the distributions $G(U_k)$, $I_k \stackrel{\text{def}}{=} G_2(U_{k'}^{(1)}) \cdots G_2(U_{k'}^{(t)})$, and $U_{t \cdot \ell_2(k')} \equiv U_{\ell_2(k')}^{(1)}) \cdots U_{\ell_2(k')}^{(t)}$. The reader can verify that I_k is $(s_2(k'), t \cdot \varepsilon_2(k'))$-pseudorandom (see Exercise 8.21), and so we focus on proving that I_k is indistinguishable from $G(U_k)$ by machines of space $s_1(k)/2$ (with respect to distinguishing-gap $\varepsilon_1(k)$). This is proved by converting a potential distinguisher into a distinguisher of $U_{\ell_1(k)} \equiv U_{t \cdot k'}$ and $G_1(U_k)$, where the new distinguisher parses the $\ell_1(k)$-bit long input into t blocks (each of length k'), invokes G_2 on the corresponding k'-bit long blocks, and feeds the resulting sequence of $\ell_1(k')$-bit long blocks to the

Specifically, with overwhelmingly high probability, at layer t machine D_k is at a vertex that can be reached in more than $2^{0.99 \cdot (t - s(k))}$ different ways. In this case, the distribution representing a random walk that reaches this vertex has min-entropy greater than $0.99 \cdot (t - s(k))$. The reader is referred to §D.4.1.1 for definitions of min-entropy and extractors.

original distinguisher. For this end, it is crucial that G_2 can be evaluate on k'-bit long strings using space at most $s_1(k)/2$, which is guaranteed by our setting of $k' = s_1(k)/O(1)$ and the hypothesis that G_2 works in linear space. □

8.5.2.2 Derandomization of space-complexity classes

As a direct application of Theorem 8.21, we obtain that $\mathcal{BPL} \subseteq \mathrm{DSPACE}(\log^2)$, where $\mathcal{BPL}$ denotes the log-space analogue of $\mathcal{BPP}$ (see Definition 6.9). (Recall that $\mathcal{NL} \subseteq \mathrm{DSPACE}(\log^2)$, but it is not known whether or not $\mathcal{BPL} \subseteq \mathcal{NL}$.)[32] A stronger derandomization result can be obtained by a finer analysis of the proof of Theorem 8.21.

Theorem 8.23 *$\mathcal{BPL} \subseteq \mathcal{SC}$, where $\mathcal{SC}$ denotes the class of decision problems that can be solved by a deterministic machine that runs in polynomial-time and polylogarithmic-space.*

Thus, $\mathcal{BPL}$ (and in particular $\mathcal{RL} \subseteq \mathcal{BPL}$) is placed in a class not known to contain $\mathcal{NL}$. Another such result was subsequently obtained in [184]: Randomized log-space can be simulated in deterministic space $o(\log^2)$; specifically, in space $\log^{3/2}$. We mention that the archetypical problem of $\mathcal{RL}$ has been recently proved to be in $\mathcal{L}$ (see Section 5.2).

Overview of the proof of Theorem 8.23.[33] We are going to use the generator construction provided in the proof of Theorem 8.21, but show that the main part of the seed (i.e., the sequence of hash functions) can be fixed (depending on the distinguisher at hand). Furthermore, this fixing can be performed in polylogarithmic space and polynomial-time. Specifically, wishing to derandomize a specific log-space computation (which refers to a specific input), we first obtain the corresponding distinguisher, denoted D'_k, that represents this computation (as a function of the outcomes of the internal coin tosses of the log-space algorithm). The key observation is that the question of whether or not a specific hash function $h \in H_n$ is good for a specific D'_k can be determined in space that is linear in $n = |h|/2$ and logarithmic in the size of D'_k. Indeed, the time complexity of this decision procedure is exponential in its space complexity. It follows that we can find a good $h \in H_n$, for a given D'_k, within these complexities (by scanning through all possible $h \in H_n$). Once a good h is found, we can also construct the corresponding graph D''_k (in which edges represent 2-edge paths in D'_k), again within the same complexity. Actually, it will be more instructive to note that we can determine a step (i.e., an edge-traversal) in D''_k by making two steps (edge-traversals) in D'_k. This will allow to fix a hash function for D''_k, and so on. Details follow.

The main claim is that the entire process of finding a sequence of $t \stackrel{\text{def}}{=} \log_2 \ell'(k)$ good hash functions can be performed in space $t \cdot O(n + \log |D_k|) = O(n + \log |D_k|)^2$ and time $\mathrm{poly}(2^n \cdot |D_k|)$; that is, the time complexity is sub-exponential in the space

[32] Indeed, the log-space analogue of $\mathcal{RP}$, denoted $\mathcal{RL}$, is contained in $\mathcal{NL} \subseteq \mathrm{DSPACE}(\log^2)$, and thus the fact that Theorem 8.21 implies $\mathcal{RL} \subseteq \mathrm{DSPACE}(\log^2)$ is of no interest.

[33] A detailed proof appears in [163].

complexity (i.e., the time complexity is significantly smaller than than the generic bound of $\exp(O(n+\log|D_k|)^2))$. Starting with $D_k^{(1)}=D'_k$, we find a good (for $D_k^{(1)}$) hashing function $h^{(1)}\in H_n$, which defines $D_k^{(2)}=D''_k$. Having found (and stored) $h^{(1)},...,h^{(i)}\in H_n$, which determine $D_k^{(i+1)}$, we find a good hashing function $h^{(i+1)}\in H_n$ for $D_k^{(i+1)}$ by emulating pairs of edge-traversals on $D_k^{(i+1)}$. Indeed, a key point is that we do *not* construct the sequence of graphs $D_k^{(2)},...,D_k^{(i+1)}$, but rather emulate an edge-traversal in $D_k^{(i+1)}$ by making 2^i edge-traversals in D'_k, using $h^{(1)},...,h^{(i)}$: The (edge-traversal) move $\alpha\in\{0,1\}^n$ starting at vertex v of $D_k^{(i+1)}$ translates to a sequence of 2^i moves starting at vertex v of D'_k, where the moves are determined by the sequence of n-bit strings

$$h^{(0^i)}(\alpha), h^{(0^{i-2}01)}(\alpha), h^{(0^{i-2}10)}(\alpha), h^{(0^{i-2}11)}(\alpha), ..., h^{(1^i)}(\alpha),$$

where $h^{(\sigma_i\cdots\sigma_1)}$ is the function obtained by the composition of some of the functions $h^{(1)},...,h^{(i)}$. (Specifically, $h^{(\sigma_i\cdots\sigma_1)}$ equals $h^{(i_1)}\circ h^{(i_2)}\circ\cdots\circ h^{(i_{t'})}$, where $\{i_j:j=1,...,t'\}=\{j:\sigma_j=1\}$ and $i_1<i_2<\cdots<i_{t'}$.) Thus, for $n=\Theta(\log|D'_k|)$, given D'_k and a pair (u,v) of source and sink in D'_k (which reside in the first and last layer, respectively), we can (deterministically) approximate the probability that a random walk starting at u reaches v in $O(\log|D'_k|)^2$-space and $\text{poly}(|D'_k|)$-time. The approximation can be made accurate up to a factor of $1\pm(1/\text{poly}(|D'_k|))$.

We conclude the proof by recalling the connection between such an approximation and the derandomization of $\mathcal{BPL}$ (indeed, note the analogy to the proof of Theorem 8.13). The computation of a log-space probabilistic machine M on input x, can be represented by a directed layer graph $G_{M,x}$ of size $\text{poly}(|x|)$. Specifically, the probability that M accepts x equals the probability that a random walk starting at the single vertex of the first layer of $G_{M,x}$ reaches some vertex in the last layer that represents an accepting configuration. Setting $k=\Theta(\log|x|)$ and $n=\Theta(k)$, the graph $G_{M,x}$ coincides with the graph D_k referred to at the beginning of the proof of Theorem 8.21, and D'_k is obtained from D_k by an "n-layer contraction" (see ibid.). Combining this with the foregoing analysis, we conclude that the probability that M accepts x can be deterministically approximated in $O(\log|x|)^2$-space and $\text{poly}(|x|)$-time. The theorem follows. □

8.6 Special Purpose Generators

In this section we consider even weaker types of pseudorandom generators, producing sequences that can fool only very restricted types of distinguishers. Still, such generators have many applications in complexity theory and in the design of algorithms. (These applications will only be mentioned briefly.)

Our choice is to start with the simplest of these generators: the pairwise-independent generator, and its generalization to t-wise independence for any $t\geq 2$. Such generators perfectly fool any distinguisher that only observe t locations in the output sequence. This leads naturally to almost pairwise (or t-wise) independence generators, which also fool (albeit non-perfectly) such distinguishers. The latter generators are implied by a stronger class of generators, which is of independent interest: the small-bias generators. Small-bias generators fool any linear test (i.e., any distinguisher that merely considers the XOR of some fixed locations in the input sequence). We then turn to the Expander Random Walk Generator: this generator produces a sequence of strings that hit any dense subset of strings with probability that is close to the hitting probability of a truly random sequence. Related notions such as samplers, dispersers, and extractors are treated in Appendix D.

Comment regarding our parameterization: To maintain consistency with prior sections, we continue to present the generators in terms of the seed length, denoted k. Since this is not the common presentation for most results presented in the sequel, we provide (in footnotes) the common presentation in which the seed length is determined as a function of other parameters.

8.6.1 Pairwise-Independence Generators

Pairwise (resp., t-wise) independence generators fool tests that inspect only two (resp., t) elements in the output sequence of the generator. Such local tests are indeed very restricted, yet they arise naturally in many settings. For example, such a test corresponds to a probabilistic analysis (of a procedure) that only relies on the pairwise independence of certain choices made by the procedure. We also mention that, in some natural range of parameters, pairwise independent sampling is as good as sampling by totally independent sample points; see Sections D.1.2 and D.3.

A t-wise independence generator of block-size $b:\mathbb{N}\to\mathbb{N}$ (and stretch function ℓ) is an efficient deterministic algorithm (e.g., one that works in time polynomial in the output length) that expands a k-bit long random seed into a sequence of $\ell(k)/b(k)$ blocks, each of length $b(k)$, such that any t blocks are uniformly and independently distributed in $\{0,1\}^{t\cdot b(k)}$. That is, denoting the $i^{\rm th}$ block of the generator's output (on seed s) by $G(s)_i$, we requite that for every $i_1 < i_2 < \cdots < i_t$ (in $[\ell(k)/b(k)]$) it holds that

$$G(U_k)_{i_1}, G(U_k)_{i_2}, ..., G(U_k)_{i_t} \equiv U_{t\cdot b(k)}.$$

In case $t = 2$, we call the generator pairwise independent. We note that this condition holds even if the inspected t blocks are selected adaptively (see Exercise 8.23)

8.6.1.1 Constructions

In the first construction, we refer to $\mathrm{GF}(2^{b(k)})$, the finite field of $2^{b(k)}$ elements, and associate its elements with $\{0,1\}^{b(k)}$.

Proposition 8.24 (t-wise independence generator):[34] *Let t be a fixed integer and $b,\ell,\ell':\mathbb{N}\to\mathbb{N}$ such that $b(k) = k/t$, $\ell'(k) = \ell(k)/b(k) > t$ and $\ell'(k) \leq 2^{b(k)}$. Let*

[34] In the common presentation of this t-wise independence generator, the length of the seed is determined as a function of the desired block-length and stretch. That is, given the parameters b and $\ell' \leq 2^b$, the seed length is set to $t \cdot b$.

$\alpha_1, ..., \alpha_{\ell'(k)}$ be fixed distinct elements of the field $\mathrm{GF}(2^{b(k)})$. *For $s_0, s_1, ..., s_{t-1} \in \{0,1\}^{b(k)}$, let*

$$G(s_0, s_1, ..., s_{t-1}) \stackrel{\text{def}}{=} \left(\sum_{j=0}^{t-1} s_j \alpha_1^j \, , \, \sum_{j=0}^{t-1} s_j \alpha_2^j \, , ..., \, \sum_{j=0}^{t-1} s_j \alpha_{\ell'(k)}^j \right) \tag{8.12}$$

where the arithmetic is that of $\mathrm{GF}(2^{b(k)})$. *Then, G is a t-wise independence generator of block-size b and stretch ℓ.*

That is, given a seed that consists of t elements of $\mathrm{GF}(2^{b(k)})$, the generator outputs a sequence of $\ell'(k)$ such elements. To make the above generator totally explicit, we need an explicit representation of $\mathrm{GF}(2^{b(k)})$, which requires an irreducible polynomial of degree $b(k)$ over $\mathrm{GF}(2)$. For specific values of $b(k)$, a good representation does exist: For example, for $d \stackrel{\text{def}}{=} b(k) = 2 \cdot 3^e$ (with e being an integer), the polynomial $x^d + x^{d/2} + 1$ is irreducible over $\mathrm{GF}(2)$. The proof of Proposition 8.24 is left as an exercise (see Exercise 8.24). We note that an analogous constructions work for every finite field (e.g., a finite field of any prime cardinality).

An alternative construction for the case of $t = 2$ is obtained by using (random) affine transformations (as possible seeds). In fact, better performance (i.e., shorter seed length) is obtained by using affine transformations defined by Toeplitz matrices. A **Toeplitz matrix** is a matrix with all diagonals being homogeneous (see Figure 8.4); that is, $T = (t_{i,j})$ is a Toeplitz matrix if $t_{i,j} = t_{i+1,j+1}$ for all i, j. Note that a Toeplitz matrix is determined by its first row and first column (i.e., the values of $t_{1,j}$'s and $t_{i,1}$'s).

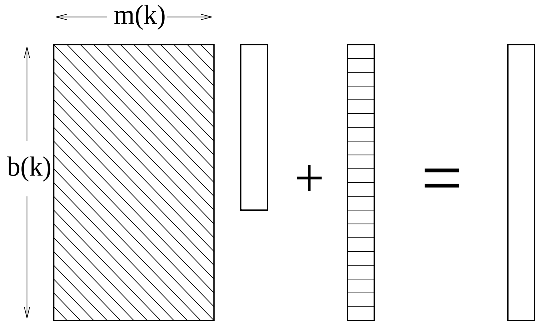


Figure 8.4: An affine transformation defined by a Toeplitz matrix.

Proposition 8.25 (Alternative pairwise independence generator, see Figure 8.4):[35] *Let $b, \ell, \ell', m : \mathbb{N} \to \mathbb{N}$ such that $\ell'(k) = \ell(k)/b(k)$ and $m(k) = \lceil \log_2 \ell'(k) \rceil = k - 2b(k) + 1$. Associate $\{0,1\}^n$ with the n-dimensional vector space over* $\mathrm{GF}(2)$,

[35] In the common presentation of this pairwise independence generator, the length of the seed is determined as a function of the desired block-length and stretch. That is, given the parameters b and ℓ', the seed length is set to $2b + \lceil \log_2 \ell' \rceil - 1$.

and let $v_1, ..., v_{\ell'(k)}$ *be fixed distinct vectors in the* $m(k)$*-dimensional vector space over* GF(2)*. For* $s \in \{0,1\}^{b(k)+m(k)-1}$ *and* $r \in \{0,1\}^{b(k)}$*, let*

$$G(s,r) \stackrel{\text{def}}{=} (T_s v_1 + r \, , \, T_s v_2 + r \, , ..., \, T_s v_{\ell'(k)} + r) \tag{8.13}$$

where T_s *is an* $b(k)$*-by-*$m(k)$ *Toeplitz matrix specified by the string* s*. Then* G *is a pairwise independence generator of block-size* b *and stretch* ℓ*.*

That is, given a seed that represents an affine transformation defined by an $b(k)$-by-$m(k)$ Toeplitz matrix, the generator outputs a sequence of $\ell'(k) \leq 2^{m(k)}$ strings, each of length $b(k)$. Note that $k = 2b(k)+m(k)-1$, and that the stretching property requires $\ell'(k) > k/b(k)$. The proof of Proposition 8.25 is left as an exercise (see Exercise 8.25).

A stronger notion of efficient generator. We note that the aforementioned constructions satisfy a stronger notion of efficient generation, which is useful in several applications. Specifically, there exists a polynomial-time algorithm that given a seed, $s \in \{0,1\}^k$, and a block location $i \in [\ell'(k)]$ (in binary), outputs the i^{th} block of the corresponding output (i.e., the i^{th} block of $G(s)$).

8.6.1.2 Applications

Pairwise independence generators do suffice for a variety of applications (cf., [222, 150]). In particular, we mention the application to sampling discussed in Appendix D.3, and the (celebrated) derandomization of the fast parallel algorithm for the Maximal Independent Set problem. This derandomization relies on the fact that the analysis of the randomized algorithm only relies on the hypothesis that some objects are distributed in pairwise independent manner. Thus, this analysis holds also when these objects are selected using a pairwise independence generator. In general, pairwise independence generators do suffice to fool distinguishers that are derived from some natural and interesting randomized algorithms.

Referring to Eq. (8.12), we remark that for any constant $t \geq 2$, the cost of derandomization (i.e., going over all 2^k possible seeds) is exponential in the block-size (because $b(k) = \Omega(k)$), which in turn also bounds the number of blocks (because $\ell'(k) \leq 2^{b(k)}$). Note that if a larger number of blocks is needed, we can artificially increase the block-length in order to accommodate it (i.e., allow $\ell'(k) = 2^{b(k)} = \exp(k/t)$), and in this case the cost of derandomization will be polynomial in the number of blocks. Thus, *whenever the analysis of a randomized algorithm can be based on a constant amount of independence* between (feasibly-many) random choices, each made within a feasible domain, *a feasible derandomization is possible.*[36] On the other hand, the relationship $\ell(k) = \exp(k/t)$ is the best possible; specifically, one cannot produce from a seed of length k an $\exp(k/O(1))$-long sequence of non-constantly independent random bits. In other words, t-wise

[36] We stress that it is important to have the cost of derandomization be polynomial in the length of the produced pseudorandom sequence, because the latter is typically polynomially-related to the length of the input to the algorithm that we wish to derandomize.

independent generators of (any block-length and) stretch ℓ require a seed of length $\Omega(t \cdot \log \ell)$. In the next subsection (cf. §8.6.2.2) we will see that meaningful approximations may be obtained with much shorter seeds.

8.6.2 Small-Bias Generators

Trying to go beyond constant-independence in derandomizations (while using seeds of length that is logarithmic in the length of the pseudorandom sequence) was the original motivation (and remain an important application) of the notion of small-bias generators. Still, small-bias generators are interesting for their own sake, and in particular they fool "global tests" that look at the entire output sequence and not merely at a fixed number of positions in it (as the limited independence generators). Specifically, small-bias generators generate a sequence of bits that fools any linear test (i.e., a test that computes a fixed linear combination of the bits).

For $\varepsilon : \mathbb{N} \to [0,1]$, an ε-bias generator with stretch function ℓ is an efficient deterministic algorithm (e.g., working in poly$(\ell(k))$ time) that expands a k-bit long random seed into a sequence of $\ell(k)$ bits such that for any fixed non-empty set $S \subseteq \{1, ..., \ell(k)\}$ the bias of the output sequence over S is at most $\varepsilon(k)$. The bias of a sequence of n (possibly dependent) Boolean random variables $\zeta_1, ..., \zeta_n \in \{0,1\}$ over a set $S \subseteq \{1, .., n\}$ is defined as

$$2 \cdot \left| \mathsf{Pr}[\oplus_{i \in S} \zeta_i = 1] - \frac{1}{2} \right| = |\mathsf{Pr}[\oplus_{i \in S} \zeta_i = 1] - \mathsf{Pr}[\oplus_{i \in S} \zeta_i = 0]|. \tag{8.14}$$

The factor of 2 was introduced so to make these biases correspond to the Fourier coefficients of the distribution (viewed as a function from $\{0,1\}^n$ to the reals). To see the correspondence replace $\{0,1\}$ by $\{\pm 1\}$, and substitute XOR by multiplication. The bias with respect to set S is thus written as

$$\left| \mathsf{Pr}\left[\prod_{i \in S} \zeta_i = +1 \right] - \mathsf{Pr}\left[\prod_{i \in S} \zeta_i = -1 \right] \right| = \left| \mathsf{E}\left[\prod_{i \in S} \zeta_i \right] \right|, \tag{8.15}$$

which is merely the (absolute value of the) Fourier coefficient corresponding to S.

8.6.2.1 Constructions

Efficient small-bias generators with exponential stretch and exponentially vanishing bias are know.

Theorem 8.26 (small-bias generators):[37] *For some universal constant $c > 0$, let $\ell : \mathbb{N} \to \mathbb{N}$ and $\varepsilon : \mathbb{N} \to [0,1]$ such that $\ell(k) \le \varepsilon(k) \cdot \exp(k/c)$. Then, there exists an ε-bias generator with stretch function ℓ operating in time polynomial in the length of its output.*

[37] In the common presentation of this generator, the length of the seed is determined as a function of the desired bias and stretch. That is, given the parameters ε and ℓ, the seed length is set to $c \cdot \log(\ell/\varepsilon)$. We comment that using [9] the constant c is merely 2 (i.e., $k \approx 2\log_2(\ell/\varepsilon)$), whereas using [159] $k \approx \log_2 \ell + 4\log_2(1/\varepsilon)$.

Three simple constructions of small-bias generators that satisfy Theorem 8.26 are known (see [9]). One of these constructions is based on Linear Feedback Shift Registers. Loosely speaking, the first half of the seed, denoted $f_0 f_1 \cdots f_{(k/2)-1}$, is interpreted as a (non-degenerate) feedback rule[38], the other half, denoted $s_0 s_1 \cdots s_{(k/2)-1}$, is interpreted as "the start sequence", and the output sequence, denoted $r_0 r_1 \cdots r_{\ell(k)-1}$, is obtained by setting $r_i = s_i$ for $i < k/2$ and $r_i = \sum_{j=0}^{(k/2)-1} f_j \cdot r_{i-(k/2)+j}$ for $i \geq k/2$. (See Figure 8.5 and Exercise 8.29.)

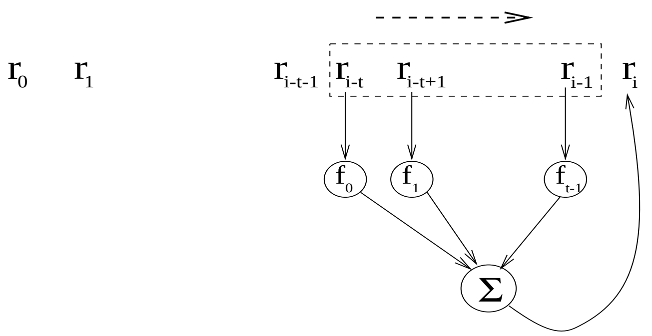


Figure 8.5: The LFSR small-bias generator (for $t = k/2$).

As in Section 8.6.1.1, we note that the aforementioned constructions satisfy a stronger notion of efficient generation, which is useful in several applications. Specifically, there exists a polynomial-time algorithm that given a seed and a bit location $i \in [\ell(k)]$ (in binary), outputs the i^{th} bit of the corresponding output.

8.6.2.2 Applications

An archetypical application of small-bias generators is for producing short and random "fingerprints" (or "digests") such that equality/inequality among strings is (probabilistically) reflected in equality/inequality between their corresponding fingerprints. The key observation is that checking whether or not $x = y$ is probabilistically reducible to checking whether the inner product modulo 2 of x and r equals the inner product modulo 2 of y and r, where r is generated by a small-bias generator G. Thus, the pair (s, v), where s is a random seed to G and v equals the inner product modulo 2 of z and $G(s)$, serves as the randomized fingerprint of the string z. One advantage of this reduction is that only few bits (i.e., the seed of the generator and the result of the inner product) needs to be "communicated between x and y" in order to enable the checking (see Exercise 8.27). A related advantage (i.e., low randomness complexity) underlies the application of small-bias generators in §9.3.2.2.

Small-bias generators have been used in a variety of areas (e.g., inapproximation, structural complexity, and applied cryptography; see references in [86, Sec

[38]That is, $f_0 = 1$ and $f(z) \stackrel{\text{def}}{=} z^{k/2} + \sum_{j=0}^{(k/2)-1} f_j \cdot z^j$ is required to be an irreducible polynomial over GF(2). The enforcing of the latter condition is discussed in Exercise 8.29.

3.6.2]). In addition, they seem an important tool in the design of various types of "pseudorandom" objects; see next.

Approximate independence generators. As hinted at the beginning of this section, small-bias is related to approximate limited independence.[39] Actually, even a restricted type of ε-bias (in which only subsets of size $t(k)$ are required to have bias upper-bounded by ε) implies that any $t(k)$ bits in the said sequence are $2^{t(k)/2} \cdot \varepsilon(k)$-close to $U_{t(k)}$, where here we refer to the variation distance (i.e., Norm-1 distance) between the two distributions. (The max-norm of the difference is bounded by $\varepsilon(k)$.)[40] Combining Theorem 8.26 and the foregoing upper-bound, and relying on the linearity of the construction presented in Proposition 8.24, we obtain generators with $\exp(k)$ stretch that are approximately $t(k)$-independent, for some non-constant $t(k)$; see Exercise 8.32. Specifically, for $k = O(t(k) + \log(1/\varepsilon(k)) + \log\log \ell(k))$ (equiv., for $\ell(k) = 2^{2^{k/O(1)}}$, $t(k) = k/O(1)$, and $\varepsilon(k) = 2^{-k/O(1)}$), one may obtain generators with stretch function ℓ, producing bit sequences in which any $t(k)$ positions are at most $\varepsilon(k)$-away from uniform (in variation distance). In the corresponding result for the max-norm distance, it suffices to have $k = O(\log(t(k)/\varepsilon(k) + \log\log \ell(k))$. Thus, whenever the analysis of a randomized algorithm can be based on a logarithmic amount of (almost) independence between feasibly-many binary random choices, a feasible derandomization is possible (by using an adequate generator of logarithmic seed length).

Extensions to non-binary choices were considered in various works (see references in [86, Sec 3.6.2]). Some of these works also consider the related problem of constructing small "discrepancy sets" for geometric and combinatorial rectangles.

t-universal set generators. Using the aforementioned upper-bound on the max-norm (of the deviation from uniform of any t locations), any ε-bias generator yields a *t-universal set generator*, provided that $\varepsilon < 2^{-t}$. The latter generator outputs sequences such that in every subsequence of length t all possible 2^t patterns occur (i.e., each for at least one possible seed). Such generators have many applications.

8.6.2.3 Generalization

In this subsection, we outline a generalization of the treatment of small-bias generators to the generation of sequences over an arbitrary finite field. Focusing on the case of a field of prime characteristic, denoted $\mathrm{GF}(p)$, we first define an adequate notion of bias. Generalizing Eq. (8.15), we define the bias of a sequence of n (possibly dependent) random variables $\zeta_1, ..., \zeta_n \in \mathrm{GF}(p)$ with respect to the linear combination $(c_1, ..., c_n) \in \mathrm{GF}(p)^n$ as $\left\| \mathsf{E}\left[\omega^{\sum_{i=1}^n c_i\zeta_i}\right]\right\|$, where ω denotes the p^{th} (complex) root of unity (i.e., $\omega = -1$ if $p = 2$). Using Exercise 8.34, we note that upper-bounds on the biases of $\zeta_1, ..., \zeta_n$ (with respect to any non-zero linear

[39] We warn that, unlike in the case of perfect independence, here we refer only to the distribution on fixed bit locations. See Exercise 8.26 for further discussion.

[40] Both bounds are derived from the Norm2 bound on the difference vector (i.e., the difference between the two probability vectors). For details, see Exercise 8.28.

combinations) yield upper-bounds on the distance of $\sum_{i=1}^{n} c_i \zeta_i$ from the uniform distribution.

We say that $S \subseteq \mathrm{GF}(p)^n$ is an ε-bias probability space if a uniformly selected sequence in S has bias at most ε with respect to any non-zero linear combination over $\mathrm{GF}(p)$. (Whenever such a space is efficiently constructible, it yields a corresponding ε-biased generator.) We mention that the LFSR construction, outlined in §8.6.2.1 and analyzed in Exercise 8.29, generalizes to $\mathrm{GF}(p)$ and yields an ε-bias probability space of size (at most) p^{2e}, where $e = \lceil \log_p(n/\varepsilon) \rceil$. Such constructions can be used in applications that generalize those in §8.6.2.2.

8.6.3 Random Walks on Expanders

In this section we review generators that produce a sequence of values by taking a random walk on a large graph that has a small degree but an adequate "mixing" property. Such a graph is called an expander, and by taking a random walk on it we may generate a sequence of ℓ' values over its vertex set, while using a random seed of length $b + (\ell' - 1) \cdot \log_2 d$, where 2^b denotes the number of vertices in the graph and d denotes its degree. This seed length should be compared against the $\ell' \cdot b$ random bits required for generating a sequence of ℓ' independent samples from $\{0,1\}^b$ (or taking a random walk on a clique of size 2^b). Interestingly, as we shall see, the pseudorandom sequence (generated by the said random walk on an expander) *behaves similarly to a truly random sequence with respect to hitting any fixed subset of* $\{0,1\}^b$. Let us start by defining this property (or rather by defining the corresponding hitting problem).

Definition 8.27 (the hitting problem): *A sequence of* (possibly dependent) *random variables, denoted* $(X_1, ..., X_{\ell'}$, *over* $\{0,1\}^b$ *is* (ε, δ)-hitting *if for any* (target) *set* $T \subseteq \{0,1\}^b$ *of cardinality at least* $\varepsilon \cdot 2^b$, *with probability at least* $1 - \delta$, *at least one of these variables hits* T; *that is,* $\Pr[\exists i \text{ s.t. } X_i \in T] \geq 1 - \delta$.

Clearly, a truly random sequence of length ℓ' over $\{0,1\}^b$ is (ε, δ)-hitting for $\delta = (1-\varepsilon)^{\ell'}$. The aforementioned "expander random walk generator" (to be described next) achieves similar behavior. Specifically, for arbitrary small $c > 0$ (which depends on the degree and the mixing property of the expander), the generator's output is (ε, δ)-hitting for $\delta = (1 - (1-c) \cdot \varepsilon)^{\ell'}$. To describe this generator, we need to discuss expanders.

Expanders. By expander graphs (or expanders) of degree d and eigenvalue bound $\lambda < d$, we actually mean an infinite family of d-regular graphs, $\{G_N\}_{N \in S}$ ($S \subseteq \mathbb{N}$), such that G_N is a d-regular graph over N vertices and the absolute value of all eigenvalues, save the biggest one, of the adjacency matrix of G_N is upper-bounded by λ. We will refer to such a family as to a (d, λ)-expander (for S). This technical definition is related to the aforementioned notion of "mixing" (which refers to the rate at which a random walk starting at a fixed vertex reaches uniform distribution over the graph's vertices). For further detail, see Appendix E.2.

We are interested in explicit constructions of such graphs, by which we mean that there exists a polynomial-time algorithm that on input N (in binary), a vertex $v \in G_N$ and an index $i \in \{1,...,d\}$, returns the i^{th} neighbor of v. (We also require that the set S for which G_N's exist is sufficiently "tractable" – say that given any $n \in \mathbb{N}$ one may efficiently find an $s \in S$ such that $n \le s < 2n$.) Several explicit constructions of expanders are known (see Appendix E.2.2). Below, we rely on the fact that for every $\overline{\lambda} > 0$, there exist d and an explicit construction of a $(d, \overline{\lambda} \cdot d)$-expander over $\{2^b : b \in \mathbb{N}\}$.[41] The relevant (to us) fact about expanders is stated next.

Theorem 8.28 (Expander Random Walk Theorem): *Let $G = (V,E)$ be an expander graph of degree d and eigenvalue bound λ. Let W be a subset of V and $\rho \stackrel{\text{def}}{=} |W|/|V|$, and consider walks on G that start from a uniformly chosen vertex and take $\ell' - 1$ additional random steps, where in each such step one uniformly selects one out of the d edges incident at the current vertex and traverses it. Then the probability that such a random walk stays in W is at most*

$$\rho \cdot \left(\rho + (1-\rho) \cdot \frac{\lambda}{d}\right)^{\ell'-1} \tag{8.16}$$

Thus, a random walk on an expander is "pseudorandom" with respect to the hitting property (i.e., when we consider hitting the set $V \setminus W$ and use $\varepsilon = 1 - \rho$); that is, a set of density ε is hit with probability $1-\delta$, where $\delta = (1-\varepsilon)\cdot(1-\varepsilon+(\lambda/d)\cdot\varepsilon)^{\ell'-1} < (1-(1-(\lambda/d))\cdot\varepsilon)^{\ell'}$. A proof of an upper-bound that is weaker than Eq. (8.16) is outlined in Exercise 8.35. Using Theorem 8.28 and an explicit $(2^t, \overline{\lambda} \cdot 2^t)$-expander, we get

Proposition 8.29 (The Expander Random Walk Generator):[42]

- *For every constant $\overline{\lambda} > 0$, consider an explicit construction of $(2^t, \overline{\lambda} \cdot 2^t)$-expanders for $\{2^n : n \in \mathbb{N}\}$, where $t \in \mathbb{N}$ is a sufficiently large constant. For $v \in [2^n] \equiv \{0,1\}^n$ and $i \in [2^t] \equiv \{0,1\}^t$, denote by $\Gamma_i(v)$ the vertex of the corresponding 2^n-vertex graph that is reached from vertex v when following its i^{th} edge.*

- *For $b, \ell' : \mathbb{N} \to \mathbb{N}$ such that $k = b(k) + (\ell'(k) - 1) \cdot t < \ell'(k) \cdot b(k)$, and for $v_0 \in \{0,1\}^{b(k)}$ and $i_1, ..., i_{\ell'(k)-1} \in [2^t]$, let*

  $$G(v_0, i_1, ..., i_{\ell'(k)-1}) \stackrel{\text{def}}{=} (v_0, v_1, ..., v_{\ell'(k)-1}), \tag{8.17}$$

  where $v_j = \Gamma_{i_j}(v_{j-1})$.

[41] This can be obtained with $d = \text{poly}(1/\overline{\lambda})$. In fact $d = O(1/\overline{\lambda}^2)$, which is optimal, can be obtained too, albeit with graphs of sizes that are only approximately close to powers of two.

[42] In the common presentation of this generator, the length of the seed is determined as a function of the desired block-length and stretch. That is, given the parameters b and ℓ', the seed length is set to $b + O(\ell' - 1)$.

TYPE	distinguisher's resources	generator's resources	stretch (i.e., $\ell(k)$)	comments
gen.-purpose	$p(k)$-time, $\forall$ poly. p	$\mathrm{poly}(k)$-time	$\mathrm{poly}(k)$	Assumes OW[43]
derand. BPP	$2^{k/O(1)}$-time	$2^{O(k)}$-time	$2^{k/O(1)}$	Assumes EvEC[43]
space-bounded	$s(k)$-space	$O(k)$-space	$2^{k/O(s(k))}$	runs in time
robustness	$k/O(1)$-space	$O(k)$-space	$\mathrm{poly}(k)$	$\mathrm{poly}(k) \cdot \ell(k)$
t-wise indepen.	"t-wise"	$\mathrm{poly}(k) \cdot \ell(k)$-time	$2^{k/O(t)}$	(e.g., pairwise)
small bias	"ε-bias"	$\mathrm{poly}(k) \cdot \ell(k)$-time	$2^{k/O(1)} \cdot \varepsilon(k)$	
expander	"hitting"	$\mathrm{poly}(k) \cdot \ell(k)$-time	$\ell'(k) \cdot b(k)$	
rand. walk	$(0.5, 2^{-\ell'(k)/O(1)})$-hitting for $\{0,1\}^{b(k)}$, with $\ell'(k) = ((k - b(k))/O(1)) + 1$.			

Figure 8.6: Pseudorandom generators at a glance

Then G has stretch $\ell(k) = \ell'(k) \cdot b(k)$, and $G(U_k)$ is (ε, δ)-hitting for any $\varepsilon > 0$ and $\delta = (1 - (1 - \overline{\lambda}) \cdot \varepsilon)^{\ell'(k)}$.

The stretch of G is optimized at $b(k) \approx k/2$ (and $\ell'(k) = k/2t$), but optimizing the stretch is not necessarily the goal in all applications. Expander random-walk generators have been used in a variety of areas (e.g., PCP and inapproximability (see [27, Sec. 11.1]), cryptography (see [87, Sec. 2.6]), and the design of various types of "pseudorandom" objects (see, in particular, Appendix D.3)).

Chapter Notes

Figure 8.6 depicts some of the notions of pseudorandom generators discussed in this chapter. We highlight a key distinction between the case of general-purpose pseudorandom generators (treated in Section 8.3) and the other cases (cf. Sections 8.4 and 8.5): in the former case the distinguisher is more complex than the generator, whereas in the latter cases the generator is more complex than the distinguisher. Specifically, in the general-purpose case the generator runs in (some *fixed*) polynomial-time and needs to withstand *any* probabilistic polynomial-time distinguisher. In fact, some of the proofs presented in Section 8.3 utilize the fact that the distinguisher can invoke the generator on seeds of its choice. In contrast, the Nisan-Wigderson Generator, analyzed in Theorem 8.18 (of Section 8.4), runs more time than the distinguishers that it tries to fool, and the proof relies on this fact in an essential manner. Similarly, the space complexity of the space-resilient generators presented in Section 8.5 is higher than the space-bound on the distinguishers that they fool.

The general paradigm of pseudorandom generators. Our presentation, which views vastly different notions of pseudorandom generators as incarnations of a general paradigm, has emerged mostly in retrospect. We note that, while the

[43]By the OW we denote the assumption that one-way functions exists. By EvEC we denote the assumption that the class $\mathcal{E}$ has (almost-everywhere) exponential circuit complexity.

historical study of the various notions was mostly unrelated at a technical level, the case of general-purpose pseudorandom generators served as a source of inspiration to most of the other cases. In particular, the concept of computational indistinguishability, the connection between hardness and pseudorandomness, and the equivalence between pseudorandomness and unpredictability, appeared first in the context of general-purpose pseudorandom generators (and inspired the development of "generators for derandomization" and "generators for space bounded machines"). Indeed, the study of the special-purpose generators (see Section 8.6) was unrelated to all of these.

General-purpose pseudorandom generators. The concept of *computational indistinguishability*, which underlies the entire computational approach to randomness, was suggested by Goldwasser and Micali [104] in the context of defining secure encryption schemes. Indeed, computational indistinguishability plays a key role in cryptography (see Appendix C). The general formulation of computational indistinguishability is due to Yao [223]. Using the hybrid technique of [104], Yao also observed that defining pseudorandom generators as producing sequences that are computationally indistinguishable from the corresponding uniform distribution is equivalent to defining such generators as producing unpredictable sequences. The latter definition originates in the earlier work of Blum and Micali [37].

Blum and Micali [37] pioneered the rigorous study of pseudorandom generators and, in particular, the construction of pseudorandom generators based on some simple intractability assumption. In particular, they constructed pseudorandom generators assuming the intractability of Discrete Logarithm problem over prime fields. Their work also introduces basic paradigms that were used in all subsequent improvements (cf., e.g., [223, 113]). We refer to the transformation of computational difficulty into pseudorandomness, the use of hard-core predicates (also defined in [37]), and the iteration paradigm (cf. Eq. (8.8)).

Theorem 8.11 (by which pseudorandom generators exist if and only if one-way functions exist) is due to Håstad, Impagliazzo, Levin and Luby [113], building upon the hard-core predicate of [95] (see Theorem 7.7). Unfortunately, the current proof of Theorem 8.11 is very complicated and unfit for presentation in a book of the current nature. Presenting a simpler and tighter (cf. §8.3.7.1) proof is indeed an important research project.

Pseudorandom functions (further discussed in Appendix C.3.3) were defined and first constructed by Goldreich, Goldwasser and Micali [91]. We also mention (and advocate) the study of a general theory of pseudorandom objects initiated in [92].

Derandomization of time-complexity classes. As observed by Yao [223], a non-uniformly strong notion of pseudorandom generators yields improved derandomization of time-complexity classes. A key observation of Nisan [161, 164] is that whenever a pseudorandom generator is used in this way, it suffices to require that the generator runs in time exponential in its seed length, and so the generator may have running-time greater than the distinguisher (representing the algorithm to be derandomized). This observation underlines the construction of Nisan and Wigderson [161, 164], and is the basis for further improvements culminating in [121]. Part 1 of Theorem 8.19 (i.e., the so-called "high end" derandomization of $\mathcal{BPP}$) is due to Impagliazzo and Wigderson [121], whereas Part 2 (the "low end") is from [164].

The Nisan–Wigderson Generator [164] was subsequently used in several ways transcending its original presentation. We mention its application towards fooling non-deterministic machines (and thus derandomizing constant-round interactive proof systems) and to the construction of randomness extractors [209].

In contrast to the aforementioned derandomization results, which place $\mathcal{BPP}$ in some worst-case deterministic complexity class, we now mention a result that places $\mathcal{BPP}$ in an average-case deterministic complexity class (cf. Section 10.2). We refer specifically to the theorem, which is due to Impagliazzo and Wigderson [122] but is not presented in the main text, that asserts the following: *if $\mathcal{BPP}$ is not contained in $\mathcal{EXP}$* (almost always) *then $\mathcal{BPP}$ has deterministic sub-exponential time algorithms that are correct on all typical cases* (i.e., with respect to any polynomial-time sampleable distribution).

Space Pseudorandom Generators. As stated in the first paper on the subject of space-resilient pseudorandom generators [4][44], this research direction was inspired by the derandomization result obtained via the use of general-purpose pseudorandom generators. The latter result (necessarily) depends on intractability assumptions, and so the objective was finding classes of algorithms for which derandomization is possible without relying on intractability assumptions. (This objective was achieved before for the case of constant-depth circuits.) Fundamentally different constructions of space pseudorandom generators were given in several works, but are superseded by the two incomparable results mentioned in Section 8.5.2: Theorem 8.21 (a.k.a Nisan's Generator [162]) and Theorem 8.22 (a.k.a the Nisan–Zuckerman Generator [165]). These two results have been "interpolated" in [11]. Theorem 8.23 ($\mathcal{BPL} \subseteq \mathcal{SC}$) was proved by Nisan [163].

Special Purpose Generators. The various generators presented in Section 8.6 were not inspired by any of the other types of pseudorandom generator (nor even by the generic notion of pseudorandomness). Pairwise-independence generator were explicitly suggested in [51] (and are implicit in [47]). The generalization to t-wise independence (for $t \geq 2$) is due to [6]. Small-bias generators were first defined and constructed by Naor and Naor [159], and three simple constructions were subsequently given in [9]. The Expander Random Walk Generator was suggested by Ajtai, Komlos, and Szemerédi [4], who discovered that random walks on expander graphs provide a good approximation to repeated independent attempts with respect to hitting any fixed subset of sufficient density (within the vertex set). The analysis of the hitting property of such walks was subsequently improved, culminating in the bound cited in Theorem 8.28, which is taken from [126, Cor. 6.1].

[44]This paper is more frequently cited for the Expander Random Walk technique which it has introduced.

(The foregoing historical notes do not mention several technical contributions that played an important role in the development of the area. For further details, the reader is referred to [86, Chap. 3]. In fact, the current chapter is a revision of [86, Chap. 3], providing more details for the main topics, and omitting relatively secondary material (a revision of which appears in Appendix D).)

Exercises

Exercise 8.1 Show that placing no computational requirements on the generator enables unconditional results regarding "generators" that fool any family of subexponential-size circuits. That is, making no computational assumptions, prove that there exist functions $G : \{0,1\}^* \to \{0,1\}^*$ such that $\{G(U_k)\}_{k\in\mathbb{N}}$ is (strongly) pseudorandom, while $|G(s)| = 2|s|$ for every $s \in \{0,1\}^*$. Furthermore, show that G can be computed in double-exponential time.

Guideline: Use the Probabilistic Method (cf. [10]). First, for any fixed circuit $C : \{0,1\}^n \to \{0,1\}$, upper-bound the probability that for a random set $S \subset \{0,1\}^n$ of size $2^{n/2}$ the absolute value of $\mathsf{Pr}[C(U_n) = 1] - (|\{x \in S : C(x) = 1\}|/|S|)$ is larger than $2^{-n/50}$. Next, using a union bound, prove the existence of a set $S \subset \{0,1\}^n$ of size $2^{n/2}$ such that no circuit of size $2^{n/100}$ can distinguish a uniformly distributed element of S from a uniformly distributed element of $\{0,1\}^n$, where distinguishing means with a probability gap of at least $2^{-n/100}$.

Exercise 8.2 Let A be a probabilistic polynomial-time algorithm solving the search associated with the NP-relation R, and let A_G be as in Construction 8.2. Prove that it is infeasible to find an x on which A_G outputs a wrong solution; that is, assuming for simplicity that A has error probability 1/3, prove that on input 1^n it is infeasible to find an $x \in \{0,1\}^n \cap S_R$ such that $\mathsf{Pr}[(x, A_G(x)) \notin R] > 0.4$, where $S_R \stackrel{\text{def}}{=} \{x : \exists y\, (x,y) \in R\}$.
(Hint: For x that violates the claim, it holds that $|\mathsf{Pr}[(x, A(x)) \notin R] - \mathsf{Pr}[(x, A_G(x)) \notin R]| > 0.06$.)

Exercise 8.3 Prove that omitting the absolute value in Eq. (8.4) keeps Definition 8.4 intact.
(Hint: consider $D'(z) \stackrel{\text{def}}{=} 1 - D(z)$.)

Exercise 8.4 Show that the existence of pseudorandom generators implies the existence of polynomial-time constructible probability ensembles that are statistically far apart and yet are computationally indistinguishable.

Guideline: Lower-bound the statistical distance between $G(U_k)$ and $U_{\ell(k)}$, where G is a pseudorandom generator with stretch ℓ.

Exercise 8.5 Prove that the sufficient condition in Exercise 8.4 is in fact necessary.[45] Recall that $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$ are said to be **statistically far apart** if, for some positive polynomial p and all sufficiently large n, the variation distance between

[45] This exercise follows [84], which in turn builds on [113].

X_n and Y_n is greater than $1/p(n)$. Using the following three steps, prove that the existence of *polynomial-time constructible* probability ensembles that are statistically far apart and yet are computationally indistinguishable implies the existence of pseudorandom generators.

1. Show that, without loss of generality, we may assume that the variation distance between X_n and Y_n is greater than $1 - \exp(-n)$.

   Guideline: For X_n and Y_n as in the forgoing, consider $\overline{X}_n = (X_n^{(1)}, ..., X_n^{(t(n))})$ and $\overline{Y}_n = (Y_n^{(1)}, ..., Y_n^{(t(n))})$, where the $X_n^{(i)}$'s (resp., $Y_n^{(i)}$'s) are independent copies of X_n (resp., Y_n), and $t(n) = O(n \cdot p(n)^2)$. To lower-bound the statistical difference between $\overline{X}_n$ and $\overline{Y}_n$, consider the set $S_n \stackrel{\text{def}}{=} \{z : \Pr[X_n = z] > \Pr[Y_n = z]\}$ and the random variable representing the number of copies in $\overline{X}_n$ (resp., $\overline{Y}_n$) that reside in S_n.

2. Using $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$ as in Step 1, prove the existence of a *false entropy generator*, where a false entropy generator is a deterministic polynomial-time algorithm G such that $G(U_k)$ has entropy $e(k)$ but $\{G(U_k)\}_{k\in\mathbb{N}}$ is computationally indistinguishable from a polynomial-time constructible ensemble that has entropy greater than $e(\cdot) + (1/2)$.

   Guideline: Let S_0 and S_1 be sampling algorithms such that $X_n \equiv S_0(U_{\text{poly}(n)})$ and $Y_n \equiv S_1(U_{\text{poly}(n)})$. Consider the generator $G(\sigma, r) = (\sigma, S_\sigma(r))$, and the distribution Z_n that equals (U_1, X_n) with probability $1/2$ and (U_1, Y_n) otherwise. Note that in $G(U_1, U_{\text{poly}(n)})$ the first bit is almost determined by the rest, whereas in Z_n the first bit is statistically independent of the rest.

3. Using a false entropy generator, obtain one in which the excess entropy is $\sqrt{k}$, and using the latter construct a pseudorandom generator.

   Guideline: Use the ideas presented at the end of Section 8.3.5 (i.e., the discussion of the interesting direction of the proof of Theorem 8.11).

Exercise 8.6 Prove that *if* $\{X_n\}_{n\in\mathbb{N}}$ *and* $\{Y_n\}_{n\in\mathbb{N}}$ *are computationally indistinguishable and* A *is a probabilistic polynomial-time algorithm then* $\{A(X_n)\}_{n\in\mathbb{N}}$ *and* $\{A(Y_n)\}_{n\in\mathbb{N}}$ *are computationally indistinguishable.*

(Hint: If D distinguishes the latter ensembles then D' such that $D'(z) \stackrel{\text{def}}{=} D(A(z))$ distinguishes the former.)

Exercise 8.7 In continuation to Exercise 8.6, show that the conclusion may not hold in case A is not computationally bounded. That is, show that there exists computationally indistinguishable ensembles, $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$, and an exponential-time algorithm A such that $\{A(X_n)\}_{n\in\mathbb{N}}$ and $\{A(Y_n)\}_{n\in\mathbb{N}}$ are *not* computationally indistinguishable.

Guideline: For any pair of ensembles $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$, consider the Boolean function f such that $f(z) = 1$ if and only if $\Pr[X_n = z] > \Pr[Y_n = z]$. Show that $|\Pr[f(X_n) = 1] - \Pr[f(Y_n) = 1]|$ equals the statistical difference between X_n and Y_n. Consider an adequate (approximate) implementation of f (e.g., approximate $\Pr[X_n = z]$ and $\Pr[Y_n = z]$ up to $\pm 2^{-2|z|}$), and use Exercise 8.1.

Exercise 8.8 (multiple samples vs single sample, a separation) Prove that there exist two probability ensembles that are computational indistinguishable by a single sample, but are efficiently distinguishable by two samples. Furthermore, one of these ensembles is the uniform ensembles and the other has a sparse support (i.e., only poly(n) many strings are assigned non-zero probability weight by the second distribution).

Guideline: Prove that, for every function $d : \{0,1\}^n \to [0,1]$, there exists two strings, x_n and y_n (in $\{0,1\}^n$), and a number $p \in [0,1]$ such that $\Pr[d(U_n)=1] = p \cdot \Pr[d(x_n)=1] + (1-p) \cdot \Pr[d(y_n)=1]$. Generalize this claim to m functions, using $m+1$ strings and a convex combination of the corresponding probabilities.[46] Conclude that there exists a distribution Z_n with a support of size at most $m+1$ such that for each of the first (in lexicographic order) m (randomized) algorithms A it holds that $\Pr[A(U_n)=1] = \Pr[A(Z_n)=1]$. Note that with probability at least $1/(m+1)$, two independent samples of Z_n are assigned the same value, yielding a simple two-sample distinguisher of U_n from Z_n.

Exercise 8.9 (amplifying the stretch function, an alternative construction) For G_1 and ℓ as in Construction 8.7, consider $G(s) \stackrel{\text{def}}{=} G_1^{\ell(|s|)}(s)$, where $G_1^i(x)$ denotes G_1 iterated i times on x (i.e., $G_1^i(x) = G_1^{i-1}(G_1(x))$ and $G_1^0(x) = x$). Prove that G is a pseudorandom generator of stretch ℓ. Reflect on the advantages of Construction 8.7 over the current construction.

Guideline: Use a hybrid argument, with the i^{th} hybrid being $G_1^i(U_{\ell(k)-i})$, for $i = 0, ..., \ell(k) - k$. Note that $G_1^{i+1}(U_{\ell(k)-(i+1)}) = G_1^i(G_1(U_{\ell(k)-i-1}))$ and $G_1^i(U_{\ell(k)-i}) = G_1^i(U_{|G_1(U_{\ell(k)-i-1})|})$, and use Exercise 8.6.

Exercise 8.10 (pseudorandom versus unpredictability) Prove that a probability ensemble $\{Z_k\}_{k\in\mathbb{N}}$ is pseudorandom if and only if it is unpredictable. For simplicity, we say that $\{Z_k\}_{k\in\mathbb{N}}$ is (next-bit) unpredictable if for every probabilistic polynomial-time algorithm A it holds that $\Pr_i[A(F_i(Z_k)) = B_{i+1}(Z_k)] - (1/2)$ is negligible, where $i \in \{0, ..., |Z_k| - 1\}$ is uniformly distributed, and $F_i(z)$ (resp., $B_{i+1}(z)$) denotes the i-bit prefix (resp., $i + 1^{\text{st}}$ bit) of z.

Guideline: Show that pseudorandomness implies polynomial-time unpredictability; that is, polynomial-time predictability violates pseudorandomness (because the uniform ensemble is unpredictable regardless of computing power). Use a hybrid argument to prove that unpredictability implies pseudorandomness. Specifically, the i^{th} hybrid consists of the i-bit long prefix of Z_k followed by $|Z_k| - i$ uniformly distributed bits. Thus, distinguishing the extreme hybrids (which correspond to Z_k and $U_{|Z_k|}$) implies distinguishing some neighboring hybrids, which in turn implies next-bit predictability. For the last step, use an argument as in the proof of Proposition 8.9.

Exercise 8.11 Prove that a probability ensemble is unpredictable (from left to right) if and only if it is unpredictable from right to left (or in any other canonical order).

---

[46] That is, prove that for every m functions $d_1, ..., d_m : \{0,1\}^n \to [0,1]$ there exist $m+1$ strings $z_n^{(1)}, ..., z_n^{(m+1)}$ and $m+1$ non-negative numbers $p_1, ..., p_{m+1}$ that sum-up to 1 such that for every $i \in [m]$ it holds that $\Pr[d_i(U_n)=1] = \sum_j p_j \cdot \Pr[d_i(z_n^{(j)})=1]$.

(Hint: use Exercise 8.10, and note that an ensemble is pseudorandom if and only if its reverse is pseudorandom.)

Exercise 8.12 Let f be 1-1 and length preserving, and b be a hard-core predicate of f. For any polynomial ℓ, prove that $\{G'(U_k)\}$ is unpredictable (in the sense of Exercise 8.10), where $G'(s) \stackrel{\text{def}}{=} b(f^{\ell(|s|)-1}(s)) \cdots b(f(s)) \cdot b(s)$.

Guideline: Suppose towards the contradiction that, for a uniformly distributed $j \in \{0, ..., \ell(k)-1\}$, given the j-bit long prefix of $G'(U_k)$ an algorithm A' can predict the $j+1^{\text{st}}$ bit of $G'(U_k)$. That is, given $b(f^{\ell(k)-1}(s)) \cdots b(f^{\ell(k)-j}(s))$, algorithm A' predicts $b(f^{\ell(k)-(j+1)}(s))$, where s is uniformly distributed in $\{0,1\}^k$. Consider an algorithm A that given $y = f(x)$ approximates $b(x)$ by invoking A' on input $b(f^{j-1}(y)) \cdots b(y)$, where j is uniformly selected in $\{0, ..., \ell(k)-1\}$. Analyze the success probability of A using the fact that f induces a permutation over $\{0,1\}^n$, and thus $b(f^j(U_k)) \cdots b(f(U_k)) \cdot b(U_k)$ is distributed identically to $b(f^{\ell(k)-1}(U_k)) \cdots b(f^{\ell(k)-j}(U_k)) \cdot b(f^{\ell(k)-(j+1)}(U_k))$.

Exercise 8.13 Prove that if G is a strong pseudorandom generator in the sense of Definition 8.12 then it a pseudorandom generator in the sense of Definition 8.1.
(Hint: consider a sequence of internal coin tosses that maximizes the probability in Eq. (8.2).)

Exercise 8.14 (strong computational indistinguishability) Provide a definition of the notion of computational indistinguishability that underlies Definition 8.12 (i.e., indistinguishability with respect to (non-uniform) polynomial-size circuits). Prove the following two claims:

1. Computational indistinguishability with respect to (non-uniform) polynomial-size circuits is strictly stronger than Definition 8.4.
2. Computational indistinguishability with respect to (non-uniform) polynomial-size circuits remains invariant under multiple samples (even if the underlying ensembles are not polynomial-time constructible).

Guideline: For Part 1, see the solution to Exercise 8.8. For Part 2 note that samples as generated in the proof of Proposition 8.6 can be hard-wired into the distinguishing circuit.

Exercise 8.15 Show that there exists a circuit of size $O(2^k \cdot \ell(k))$ that violates Eq. (8.9), provided that $\ell(k) > k$.
(Hint: The circuit may incorporate all values in the range of G and decide by comparing its input to these values.)

Exercise 8.16 (constructing a set system for Theorem 8.18) For every $\gamma > 0$, show a construction of a set system S as in Condition 2 of Theorem 8.18, with $m(k) = \Omega(k)$ and $\ell(k) = 2^{\Omega(k)}$.

Guideline: We assume, without loss of generality, that $\gamma < 1$, and set $m(k) = (\gamma/2) \cdot k$ and $\ell(k) = 2^{\gamma m(k)/6}$. We construct the set system $S_1, ..., S_{\ell(k)}$ in iterations, selecting S_i as the first $m(k)$-subset of $[k]$ that has sufficiently small intersections with each of the previous sets $S_1, ..., S_{i-1}$. The existence of such a set S_i can be proved using the

Probabilistic Method (cf. [10]). Specifically, for a fixed $m(k)$-subset S', the probability that a random $m(k)$-subset has intersection greater than $\gamma m(k)$ with S' is smaller than $2^{-\gamma m(k)/6}$, because the expected intersection size is $(\gamma/2) \cdot m(k)$. Thus, with positive probability a random $m(k)$-subset has intersection at most $\gamma m(k)$ with each of the previous $i-1 < \ell(k) = 2^{\gamma m(k)/6}$ subsets. Note that we construct S_i in time $\binom{k}{m(k)} \cdot (i-1) \cdot m(k) < 2^k \cdot \ell(k) \cdot k$, and thus S is computable in time $k2^k \cdot \ell(k)^2 < 2^{2k}$.

Exercise 8.17 (pseudorandom versus unpredictability, by circuits) In continuation to Exercise 8.10, show that if there exists a circuit of size s that distinguishes Z_n from U_ℓ with gap δ, then there exists an $i < \ell = |Z_n|$ and a circuit of size $s + O(1)$ that given an i-bit long prefix of Z_n guesses the $i+1^{\text{st}}$ bit with success probability at least $\frac{1}{2} + \frac{\delta}{\ell}$.
(Hint: defining hybrids as in Exercise 8.10, note that, for some i, the given circuit distinguishes the i^{th} hybrid from the $i+1^{\text{st}}$ hybrid with gap at least δ/ℓ.)

Exercise 8.18 Suppose that the sets S_i's in Construction 8.17 are disjoint and that $f : \{0,1\}^m \to \{0,1\}$ is T-inapproximable. Prove that for every circuit C of size $T - O(1)$ it holds that $|\mathsf{Pr}[C(G(U_k)) = 1] - \mathsf{Pr}[C(U_\ell) = 1]| < \ell/T$.

Guideline: Prove the contrapositive using Exercise 8.17. Note that the values of the $i+1^{\text{st}}$ bit of $G(U_k)$ is statistically independent of the values of the first i bits of $G(U_k)$, and thus predicting it yields an approximator for f. Indeed, such an approximator can be obtained by fixing the the first i bits of $G(U_k)$ via an averaging argument.

Exercise 8.19 (Theorem 8.18, generalized) Let $\ell, m, m', T : \mathbb{N} \to \mathbb{N}$ satisfy $\ell(k)^2 + \widetilde{O}(\ell(k)2^{m'(k)}) < T(m(k))$. Suppose that the following two conditions hold:

1. There exists an exponential-time computable function $f : \{0,1\}^* \to \{0,1\}$ that is T-inapproximable.
2. There exists an exponential-time computable function $S : \mathbb{N}\times\mathbb{N} \to 2^{\mathbb{N}}$ such that for every k and $i = 1, ..., \ell(k)$ it holds that $S(k,i) \subseteq [k]$ and $|S(k,i)| = m(k)$, and $|S(k,i) \cap S(k,j)| \leq m'(k)$ for every k and $i \neq j$.

Prove that using G as defined in Construction 8.17, with $S_i = S(k,i)$, yields a canonical derandomizer with stretch ℓ.
(Hint: following the proof of Theorem 8.18, just note that the circuit constructed for approximating $f(U_{m(k)})$ has size $\ell(k)^2 + \ell(k) \cdot \widetilde{O}(2^{m'(k)})$ and success probability at least $(1/2) + (1/7\ell(k))$.)

Exercise 8.20 (Part 2 of Theorem 8.19) Prove that if for every polynomial T there exists a T-inapproximable predicate in $\mathcal{E}$ then $\mathcal{BPP} \subseteq \cap_{\varepsilon>0}\mathrm{D}\textsc{time}(t_\varepsilon)$, where $t_\varepsilon(n) \stackrel{\text{def}}{=} 2^{n^\varepsilon}$.

Guideline: For any p-time algorithm, apply Exercise 8.19 using $\ell(k) = p(k^{1/\varepsilon})$, $m(k) = \sqrt{k}$ and $m'(k) = O(\log k)$. Revisit Exercise 8.16 in order to obtain a set system as required in Exercise 8.19 (for these parameters), and use Theorem 7.10.

Exercise 8.21 (multiple samples and space-bounded machines) Suppose that two probability ensembles, $\{X_k\}_{k\in\mathbb{N}}$ and $\{Y_k\}_{k\in\mathbb{N}}$, are (s,ε)-indistinguishable by non-uniform machines (i.e., the distinguishability-gap of any non-uniform machine of space s is bounded by the function ε). For any function $t:\mathbb{N}\rightarrow\mathbb{N}$, prove that the ensembles $\{(X_k^{(1)},...,X_k^{(t(k))})\}_{k\in\mathbb{N}}$ and $\{(Y_k^{(1)},...,X_k^{(t(k))})\}_{k\in\mathbb{N}}$ are $(s,t\varepsilon)$-indistinguishable, where $X_k^{(1)}$ through $X_k^{(t(k))}$ and $Y_k^{(1)}$ through $Y_k^{(t(k))}$ are independent random variables, with each $X_k^{(i)}$ identical to X_k and each $Y_k^{(i)}$ identical to Y_k.

Guideline: Use the hybrid technique. When distinguishing the i^{th} and $(i+1)^{\text{st}}$ hybrids, note that the first i blocks (i.e., copies of X_k) as well as the last $t(k)-(i+1)$ blocks (i.e., copies of Y_k) can be fixed and hard-wired into the non-uniform distinguisher.

Exercise 8.22 Provide an explicit description of the generator outlined in the proof of Theorem 8.21.

Guideline: for $r\in\{0,1\}^n$ and $h^{(1)},...,h^{(t)}\in H_n$, the genera or outputs a 2^t-long sequence of n-bit strings such that the i^{th} block equals $h'(r)$, where h' is a composition of some of the $h^{(j)}$'s.

Exercise 8.23 (adaptive t-wise independence tests) Recall that a generator $G:\{0,1\}^k\rightarrow\{0,1\}^{\ell'(k)\cdot b(k)}$ is called t-wise independent if *for any t fixed block positions*, the distribution $G(U_k)$ restricted to these t blocks is uniform over $\{0,1\}^{t\cdot b(k)}$. Prove that the output of a t-wise independence generator is (perfectly) indistinguishable from the uniform distribution *by any test that examines t of the blocks*, even if the examined blocks are selected adaptively (i.e., the location of the i^{th} block to be examined is determined based on the contents of the previously inspected blocks).

Guideline: First show that, without loss of generality, it suffices to consider deterministic (adaptive) tester. Next, show that the probability that such a tester sees any fixed sequence of t values at the locations selected *adaptively* in the generator's output equals $2^{-t\cdot b(k)}$, where $b(k)$ is the block length.

Exercise 8.24 (t-wise independence generator) Prove that G as defined in Proposition 8.24 produces a t-wise independent sequence over $\mathrm{GF}(2^{b(k)})$.

Guideline: For every t fixed indices $i_1,...,i_t\in[\ell'(k)]$, consider the distribution of $G(U_k)_{i_1,...,i_t}$ (i.e., the projection of $G(U_k)$ on locations $i_1,...,i_t$). Show that for every sequence of t possible values $v_1,...,v_t\in\mathrm{GF}(2^{b(k)})$, there exists a unique seed $s\in\{0,1\}^k$ such that $G(s)_{i_1,...,i_t}=(v_1,...,v_t)$.

Exercise 8.25 (pairwise independence generators) As a warm-up, consider a construction analogous to the one in Proposition 8.25, where the seed specifies an affine $b(k)$-by-$m(k)$ transformation. That is, for $s\in\{0,1\}^{b(k)\cdot m(k)}$ and $r\in\{0,1\}^{b(k)}$, where $k=b(k)\cdot m(k)+b(k)$, let

$$G(s,r)\stackrel{\text{def}}{=}(A_sv_1+r\,,\,A_sv_2+r\,,...,\,A_sv_{\ell'(k)}+r) \tag{8.18}$$

where A_s is an $b(k)$-by-$m(k)$ matrix specified by the string s. Show that G as in Eq. (8.18) is a pairwise independence generator of block-size b and stretch ℓ. (Note that a related construction appears in the proof of Theorem 7.7; see also Exercise 7.5.) Next, show that G as in Eq. (8.13) is a pairwise independence generator of block-size b and stretch ℓ.

Guideline: The following description applies to both constructions. First note that for every fixed $i \in [\ell'(k)]$, the i^{th} element in the sequence $G(U_k)$, denoted $G(U_k)_i$, is uniformly distributed in $\{0,1\}^{b(k)}$. Actually, show that for every fixed $s \in \{0,1\}^{k-b(k)}$, it holds that $G(s, U_{b(k)})_i$ is uniformly distributed in $\{0,1\}^{b(k)}$. Next note that it suffices to show that, for every $j \neq i$, conditioned on the value of $G(U_k)_i$, the value of $G(U_k)_j$ is uniformly distributed in $\{0,1\}^{b(k)}$. The key technical detail is showing that for any non-zero vector $v \in \{0,1\}^{m(k)}$ it holds that $A_{U_{k-b(k)}} v$ (resp., $T_{U_{k-b(k)}} v$) is uniformly distributed in $\{0,1\}^{b(k)}$. This is easy in case of a random $b(k)$-by-$m(k)$ matrix, and can be proven also for a random Toeplitz matrix.

Exercise 8.26 (adaptive t-wise independence tests, revisited) In contrast to Exercise 8.23, we note that almost uniform distribution on any *fixed* t bit locations does not imply that an *adaptive* test that inspects t locations cannot detect "non-uniformity" (i.e., a "non random behavior" of the inspected sequence). Specifically, present a distribution over 2^{t-1}-bit long strings in which each $t-1$ fixed bit positions are $t \cdot 2^{-(t-1)}$-close to uniform, but some test that adaptively inspects t positions can distinguish this distribution from the uniform one with constant gap.
(Hint: Modify the uniform distribution over $((t-1)+2^{t-1})$-bit long strings such that the first $t-1$ locations indicate a bit position (among the rest) that is set to zero.)

Exercise 8.27 Suppose that G is an ε-bias generator with stretch ℓ. Show that equality between the $\ell(k)$-bit strings x and y can be probabilistically checked by comparing the inner product modulo 2 of x and $G(s)$ to the inner product modulo 2 of y and $G(s)$, where $s \in \{0,1\}^k$ is selected uniformly.
(Hint: reduce the problem to the special case in which $y = 0^{\ell(k)}$.)

Exercise 8.28 (bias versus statistical difference from uniform) Let X be a random variable assuming values in $\{0,1\}^t$. Prove that if X has bias at most ε over any non-empty set then the statistical difference between X and U_t is at most $2^{t/2} \cdot \varepsilon$, and that for every $x \in \{0,1\}^t$ it holds that $\mathsf{Pr}[X = x] = 2^{-t} \pm \varepsilon$.

Guideline: Consider the probability function $p : \{0,1\}^t \to [0,1]$ defined by $p(x) \stackrel{\text{def}}{=} \mathsf{Pr}[X = x]$, and let $\delta(x) \stackrel{\text{def}}{=} p(x) - 2^{-t}$ denote the deviation of p from the uniform probability function. Viewing the set of real functions over $\{0,1\}^t$ as a 2^t-dimensional vector space, consider two orthonormal bases for this space. The first basis consists of the (Kroniker) functions $\{k_\alpha\}_{\alpha \in \{0,1\}^t}$ such that $k_\alpha(x) = 1$ if $x = \alpha$ and $k_\alpha(x) = 0$ otherwise. The second basis consists of the (normalize Fourier) functions $\{f_S\}_{S \subseteq [t]}$ defined by $f_S(x) \stackrel{\text{def}}{=} 2^{-t/2} \prod_{i \in S} (-1)^{x_i}$ (where $f_\emptyset \equiv 2^{-t/2}$).[47] Note that the bias of X over any

[47] Verify that both bases are indeed orthogonal (i.e., $\sum_x k_\alpha(x) k_\beta(x) = 0$ for every $\alpha \neq \beta$ and $\sum_x f_S(x) f_T(x) = 0$ for every $S \neq T$) and normal (i.e., $\sum_x k_\alpha(x)^2 = 1$ and $\sum_x f_S(x)^2 = 1$).

$S \neq \emptyset$ equals $|\sum_x p(x) \cdot 2^{t/2} f_S(x)|$, which in turn equals $2^{t/2}|\sum_x \delta(x) f_S(x)|$. Thus, for every S (including the empty set), we have $|\sum_x \delta(x) f_S(x)| \leq 2^{-t/2}\varepsilon$, which means that the representation of δ in the normalize Fourier basis is by coefficients that have each an absolute value of at most $2^{-t/2}\varepsilon$. It follows that the Norm-2 of this vector of coefficients is upper-bounded by $\sqrt{2^t \cdot (2^{-t/2}\varepsilon)^2} = \varepsilon$, and the two claims follow by noting that they refer to norms of δ according to the Kroniker basis. In particular, Norm-2 is preserved under orthonormal bases, the max-norm is upper-bounded by Norm-2, and Norm-1 is upper-bounded by $\sqrt{2^t}$ times the value of the Norm-2.

Exercise 8.29 (The LFSR small-bias generator (following [9])) Using the following guidelines (and letting $t = k/2$), analyze the construction outlined following Theorem 8.26 (and depicted in Figure 8.5):

1. Prove that $r_i = \sum_{j=0}^{t-1} c_j^{(f,i)} \cdot s_j$, where $c_j^{(f,i)}$ is the coefficient of z^j in the (degree $t-1$) polynomial obtained by reducing z^i modulo the polynomial $f(z)$ (i.e., $z^i \equiv \sum_{j=0}^{t-1} c_j^{(f,i)} z^j \pmod{f(z)}$).

   (Hint: Recall that $z^t \equiv \sum_{j=0}^{t-1} f_j z^j \pmod{f(z)}$, and thus $z^i \equiv \sum_{j=0}^{t-1} f_j z^{i-t+j} \pmod{f(z)}$. Note the correspondence to $r_i = \sum_{j=0}^{t-1} f_j \cdot r_{i-t+j}$.)

2. For any non-empty $S \subseteq \{0, ..., \ell(k)-1\}$, evaluate the bias of the sequence $r_0, ..., r_{\ell(k)-1}$ over S, where f is a random irreducible polynomial of degree t and $s = (s_0, ..., s_{t-1}) \in \{0,1\}^t$ is uniformly distributed. Specifically:

   (a) For a fixed f and random $s \in \{0,1\}^t$, prove that $\sum_{i \in S} r_i$ has non-zero bias if and only if $f(z)$ divides $\sum_{i \in S} z^i$.

   (Hint: Note that $\sum_{i \in S} r_i = \sum_{j=0}^{t-1} \sum_{i \in S} c_j^{(f,i)} s_j$, and use Item 1.)

   (b) Prove that the probability that a random irreducible polynomial of degree t divides $\sum_{i \in S} z^i$ is $\Theta(\ell(k)/2^t)$.

   (Hint: A polynomial of degree n can be divided by at most n/d different irreducible polynomials of degree d. On the other hand, the number of irreducible polynomials of degree d over GF(2) is $\Theta(2^d/d)$.)

   Conclude that for random f and s, the sequence $r_0, ..., r_{\ell(k)-1}$ has bias $O(\ell(k)/2^t)$.

Note that an implementation of the LFSR generator requires a mapping of random $k/2$-bit long string to *almost* random irreducible polynomials of degree $k/2$. Such a mapping can be constructed in $\exp(k)$ time, which is $\text{poly}(\ell(k))$ if $\ell(k) = \exp(\Omega(k))$. A more efficient mapping that uses a $O(k)$-bit long seek is described in [9, Sec. 8].

Exercise 8.30 (limitations on small-bias generators) Let G be an ε-bias generator with stretch ℓ, and view G as a mapping from $\text{GF}(2)^k$ to $\text{GF}(2)^{\ell(k)}$. As such, each bit in the output of G can be viewed as a polynomial[48] in the k input variables (each ranging in GF(2)). Prove that if $\varepsilon(k) < 1$ and each of these polynomials has total degree at most d then $\ell(k) \leq \sum_{i=1}^{d} \binom{k}{i}$.

[48] Recall that every Boolean function over GF(p) can be expressed as a polynomial of individual degree at most $p-1$.

Guideline: Note that, without loss of generality, all polynomials have a free term equal to zero (and has individual degree at most 1 in each variable). Next, consider the vector space spanned by all d-monomials over k variables (i.e., monomial having at most d variables). Since $\varepsilon(k) < 1$, the polynomials representing the output bits of G must correspond to a sequence of independent vectors in this space.
Derive the following corollaries:

1. If $\varepsilon(k) < 1$ then $\ell(k) < 2^k$ (regardless of d).
2. If $\varepsilon(k) < 1$ and $\ell(k) > k$ then G cannot be a linear transformation.

We note that, in contrast to Item 1, (efficient) ε-bias generators of stretch $\ell(k) = \text{poly}(\varepsilon(k)) \cdot 2^k$ do exists (see [159]). Also, in contrast to Item 2, note that $G(s) = (s, b(s))$, where $b(s_1, ..., s_k) = \sum_{i=1}^{k/2} s_i s_{(k/2)+i} \bmod 2$, is an ε-bias generator with $\varepsilon(k) = \exp(-\Omega(k))$.
(Hint: Focusing on bias over sets that include the last output bit, prove that without loss of generality it suffices to analyze the bias of $b(U_k)$.)

Exercise 8.31 (a sanity check for pseudorandomness) The following fact is suggested as a sanity check for candidate pseudorandom generators with respect to space-bounded machines. The fact (to be proven as an exercise) is that, for every $\varepsilon(\cdot)$ and $s(\cdot)$ such that $s(k) \geq 1$ for every k, if G is (s, ε)-pseudorandom (as per Definition 8.20), then G is an ε-bias generator.

Exercise 8.32 (approximate t-wise independent generators (following [159])) Combining a small-bias generator as in Theorem 8.26 with the t-wise independent generator of Eq. (8.12), and relying on the linearity of the latter, construct a generator producing ℓ-bit long sequences in which any t positions are at most ε-away from uniform (in variation distance), while using a seed of length $O(t + \log(1/\varepsilon) + \log\log \ell)$. (For max-norm a seed of length $O(\log(t/\varepsilon) + \log\log \ell)$ suffices.)

Guideline: First note that, for any t, ℓ' and b, the transformation of Eq. (8.12) can be implemented by a fixed linear (over GF(2)) transformation of a $t \cdot b$-bit seed into an ℓ-bit long sequence, where $\ell = \ell' \cdot b$. It follows that there exists a fixed GF(2)-linear transformation T of a random seed of length $t \cdot b$, where $b = \log_2 \ell'$, into a t-wise independent bit sequence of the length ℓ (i.e., $T U_{t \cdot b}$ is t-wise independent over $\{0,1\}^\ell$). Thus, every t rows of T are linearly independent. The key observation is that when we replace the aforementioned random seed by an ε'-bias sequence, every $i \leq t$ positions in the output sequence have bias at most ε' (because they define a non-zero linear test on the bits of the ε'-bias sequence). Note that the length of the new seed (used to produce ε'-bias sequence of length $t \cdot b$) is $O(\log tb/\varepsilon')$. Applying Exercise 8.28, we conclude that any t positions are at most $2^{t/2} \cdot \varepsilon'$-away from uniform (in variation distance). Recall that this was obtained using a seed of length $O(\log(t/\varepsilon') + \log\log \ell)$, and the claim follows by using $\varepsilon' = 2^{-t/2} \cdot \varepsilon$.

Exercise 8.33 (small-bias generator and error-correcting codes) Show a correspondence between ε-bias generators of stretch ℓ and binary linear error-correcting codes (cf. Appendix E.1) mapping $\ell(k)$-bit long strings to 2^k-bit long strings such that every two codewords are at distance $(1 \pm \varepsilon(k)) \cdot 2^k$ apart.

Guideline: Associate $\{0,1\}^k$ with $[2^k]$. Then, a generator $G : [2^k] \to \{0,1\}^{\ell(k)}$ corresponds to the code $C : \{0,1\}^{\ell(k)} \to \{0,1\}^{2^k}$ such that, for every $i \in [\ell(k)]$ and $j \in [2^k]$, the i^{th} bit of $G(j)$ equals the j^{th} bit of $C(0^{i-1}10^{\ell(k)-i})$.

Exercise 8.34 (on the bias of sequences over a finite field) For a prime p, let ζ be a random variable assigned values in $\mathrm{GF}(p)$ and $\delta(v) \stackrel{\text{def}}{=} \Pr[\zeta = v] - (1/p)$. Prove that $\max_{v\in\mathrm{GF}(p)}\{|\delta(v)|\}$ is upper-bounded by $b \stackrel{\text{def}}{=} \max_{c\in\{1,\ldots,p-1\}}\{\|\mathsf{E}[\omega^{c\zeta}]\|\}$, where ω denotes the p^{th} (complex) root of unity, and that $\sum_{v\in\mathrm{GF}(p)}|\delta(v)|$ is upper-bounded by $\sqrt{p}\cdot b$.

Guideline: Analogously to Exercise 8.28, view probability distributions over $\mathrm{GF}(p)$ as p-ary vectors, and consider two bases for the set of complex functions over $\mathrm{GF}(p)$: the Kroniker basis (i.e., $k_i(x) = 1$ if $x = i$ and $k_i(x) = 0$) and the (normalize) Fourier basis (i.e., $f_i(x) = p^{-1/2}\cdot\omega^{ix}$). Note that the biases of ζ corresponds to the inner products of δ with the non-constant Fourier functions, whereas the distances of ζ from the uniform distribution correspond to the inner products of δ with the Kroniker functions.

Exercise 8.35 (a version of the Expander Random Walk Theorem) Using notations as in Theorem 8.28, prove that the probability that a random walk of length ℓ' stays in W is at most $(\rho + (\lambda/d)^2)^{\ell'/2}$. In fact, prove a more general claim that refers to the probability that a random walk of length ℓ' intersects $W_0 \times W_1 \times \cdots \times W_{\ell'-1}$. The claimed upper-bound is

$$\sqrt{\rho_0}\cdot\prod_{i=1}^{\ell'-1}\sqrt{\rho_i + (\lambda/d)^2}, \tag{8.19}$$

where $\rho_i \stackrel{\text{def}}{=} |W_i|/|V|$.

Guideline: View the random walk as the evolution of a corresponding probability vector under suitable transformations. The transformations correspond to taking a random step in the graph and to passing through a "sieve" that keeps only the entries that correspond to the current set W_i. The key observation is that the first transformation shrinks the component that is orthogonal to the uniform distribution (which is the first eigenvalue of the adjacency matrix of the expander), whereas the second transformation shrinks the component that is in the direction of the uniform distribution. For further details, see §E.2.1.3.

Exercise 8.36 Using notations as in Theorem 8.28, prove that the probability that a random walk of length ℓ' visits W more than $\alpha\ell'$ times is smaller than $\binom{\ell'}{\alpha\ell'}\cdot(\rho + (\lambda/d)^2)^{\alpha\ell'/2}$. For example, for $\alpha = 1/2$ and $\lambda/d < \sqrt{\rho}$, we get an upper-bound of $(32\rho)^{\ell'/4}$. We comment that much better bounds can be obtained (cf. [82]).

(Hint: Use a union bound on all possible sequences of $m = \alpha\ell'$ visits, and upper-bound the probability of visiting W in steps $j_1, \ldots, j_m$ by applying Eq. (8.19) with $W_i = W$ if $i \in \{j_1, \ldots, j_m\}$ and $W = V$ otherwise.)

Chapter 9

Probabilistic Proof Systems

A proof is whatever convinces me.

Shimon Even (1935–2004)

Various types of *probabilistic* proof systems have played a central role in the development of computer science in the last couple of decades. In this chapter, we concentrate on three such proof systems: *interactive proofs*, *zero-knowledge proofs*, and *probabilistic checkable proofs*. These proof systems share a common (untraditional) feature – they carry a probability of error (which is explicitly bounded and can be reduced by successive application of the proof system). The gain in allowing this untraditional relaxation is substantial, as demonstrated by the three results mentioned in the summary.

Summary: The association of efficient procedures with *deterministic* polynomial-time procedures is the basis for viewing NP-proof systems as the canonical formulation of proof systems (with efficient verification procedures). Allowing *probabilistic* verification procedures and, moreover, ruling by statistical evidence gives rise to various types of probabilistic proof systems. These probabilistic proof systems carry an explicitly bounded probability of error, but they offer various advantages over the traditional (deterministic and errorless) proof systems.

Randomized and interactive verification procedures, giving rise to *interactive proof systems*, seem much more powerful than their deterministic counterparts. In particular, such interactive proof systems exist for any set in $\mathcal{PSPACE} \supseteq \text{co}\mathcal{NP}$ (e.g., for the set of unsatisfied propositional formulae), whereas it is widely believed that some sets in co$\mathcal{NP}$ do *not* have NP-proof systems (i.e., $\mathcal{NP} \neq \text{co}\mathcal{NP}$). We stress that a "proof" in this context is not a fixed and static object, but rather a randomized (and dynamic) process in which the verifier interacts with the prover. Intuitively, one may think of this interaction as consisting of questions asked by the verifier, to which the prover has to reply convincingly.

Such randomized and interactive verification procedures allow for the meaningful conceptualization of *zero-knowledge proofs*, which are of great conceptual and practical interest (especially in cryptography). Loosely speaking, zero-knowledge proofs are interactive proofs that yield nothing (to the verifier) beyond the fact that the assertion is indeed valid. For example, a zero-knowledge proof that a certain propositional formula is satisfiable does not reveal a satisfying assignment to the formula nor any partial information regarding such an assignment (e.g., whether the first variable can assume the value `true`). Thus, the successful verification of a zero-knowledge proof exhibit an extreme contrast between being convinced of the validity of a statement and learning anything in addition (while receiving such a convincing proof). It turns out that, under reasonable complexity assumptions (i.e., assuming the existence of one-way functions), every set in $\mathcal{NP}$ has a zero-knowledge proof system.

NP-proofs can be efficiently transformed into a (redundant) form that offers a trade-off between the number of locations (randomly) examined in the resulting proof and the confidence in its validity. It particular, it is known that any set in $\mathcal{NP}$ has an NP-proof system that supports probabilistic verification such that the error probability decreases exponentially with the number of bits read from the alleged proof. These redundant NP-proofs are called *probabilistically checkable proofs* (or PCPs). In addition to their conceptually fascinating nature, PCPs have played a key role in the study of the complexity of approximation problems.

Prerequisites: We assume a basic familiarity with elementary probability theory (see Appendix D.1) and randomized algorithms (see Section 6.1).

Introduction and Preliminaries

The glory attached to the creativity involved in finding proofs, makes us forget that it is the less glorified process of verification that gives proofs their value. Conceptually speaking, proofs are secondary to the verification process; whereas technically speaking, proof systems are defined in terms of their verification procedures.

The notion of a verification procedure presumes the notion of computation and furthermore the notion of efficient computation. This implicit stipulation is made explicit in the definition of $\mathcal{NP}$ (cf. Definition 2.5), in which efficient computation is associated with (deterministic) polynomial-time algorithms.[1] Thus, NP provides the ultimate formulation of proof systems (with efficient verification procedures)

[1] Recall that the formulation of NP-proof systems explicitly restricts the length of proofs to be polynomial in the length of the assertion. Thus, verification is performed in a number of steps that is polynomial in the length of the assertion. We comment that deterministic proof systems that allow for longer proofs (but require that verification is efficient in terms of the length of the alleged proof) can be modeled as NP-proof systems by adequate padding (of the assertion).

as long as one associates efficient procedures with *deterministic* polynomial-time algorithms. However, we can gain a lot if we are willing to take a somewhat non-traditional step and allow *probabilistic* verification procedures. In particular:

- Interactive proof systems, which employ randomized and interactive verification procedures, seem much more powerful than their deterministic counterparts.
- Such interactive proof systems allow for the construction of (meaningful) zero-knowledge proofs, which are of great theoretical and practical interest.
- NP-proofs can be efficiently transformed into a (redundant) form that supports super-fast probabilistic verification via very few random probes into the alleged proof.

In all these cases, explicit bounds are imposed on the computational complexity of the verification procedure, which in turn is personified by the notion of a verifier. Furthermore, in all these proof systems, the verifier is allowed to toss coins and rule by statistical evidence. Thus, all these proof systems carry a probability of error; yet, this probability is explicitly bounded and, furthermore, can be reduced by successive application of the proof system.

One important convention. When presenting a proof system, we state all complexity bounds in terms of the length of the assertion to be proven (which is viewed as an input to the verifier). Namely, when we say "polynomial-time" we mean time that is polynomial in the length of this assertion. Actually, as will become evident, this is *the* natural choice in all the cases that we consider. Note that this convention is consistent with the definition of NP-proof systems (cf. Definition 2.5), because $\text{poly}(|(x,y)|) = \text{poly}(|x|)$ for $|y| = \text{poly}(|x|)$.

Notational Conventions. Denote by `poly` the set of all integer functions bounded by a polynomial and by `log` the set of all integer functions bounded by a logarithmic function (i.e., $f \in \texttt{log}$ iff $f(n) = O(\log n)$). All complexity measures mentioned in the subsequent exposition are assumed to be constructible in polynomial-time.

Organization. In Section 9.1 we present the basic definitions and results regarding interactive proof systems. The definition of an interactive proof systems is the starting point for a discussion of zero-knowledge proofs, which is provided in Section 9.2. Section 9.3, which presents the basic definitions and results regarding probabilistically checkable proofs (PCP), can be read independently of the other sections.

9.1 Interactive Proof Systems

In light of the growing acceptability of randomized and distributed computations, it is only natural to associate the notion of efficient computation with probabilistic and interactive polynomial-time computations. This leads naturally to the notion of an interactive proof system in which the verification procedure is interactive and randomized, rather than being non-interactive and deterministic. Thus, a "proof" in this context is not a fixed and static object, but rather a randomized (dynamic) process in which the verifier interacts with the prover. Intuitively, one may think of this interaction as consisting of questions asked by the verifier, to which the prover has to reply convincingly. The foregoing discussion, as well as the definition provided in Section 9.1.1, makes explicit reference to a prover, whereas a prover is only implicit in the traditional definitions of proof systems (e.g., NP-proof systems). Before turning to the actual definition, we highlight and further discuss some of the foregoing issues.

A static object versus an interactive process. Traditionally in mathematics, a "proof" is a *fixed* sequence consisting of statements that are either self-evident or are derived from previous statements via self-evident rules. Actually, both conceptually and technically, it is more accurate to substitute the phrase "self-evident" by the phrase "commonly agreed" (because, at the last account, self-evidence is a matter of common agreement). In fact, in the formal study of proofs (i.e., logic), the commonly agreed statements are called *axioms*, whereas the commonly agreed rules are referred to as *derivation rules*. We highlight *a key property of mathematics proofs: proofs are viewed as fixed* (*static*) *objects*. In contrast, in other areas of human activity, the notion of a "proof" has a much wider interpretation. In particular, a proof is not a fixed object but rather a process by which the validity of an assertion is established. For example, in the context of Law, the cross-examination of a witness in court (including its non-verbal components) may be considered a proof (or a refutation) of some claim. Likewise, debates that take place in daily life have an analogous potential of establishing claims and are then perceived as proofs. This perception is quite common in philosophical and political debates, and arise also in scientific debates. Furthermore, some technical "proofs by contradiction" appeal to this daily experience by emulating an imaginary debate with a potential (generic) skeptic.

We note that, in mathematics, proofs are often considered more fundamental than their consequence (i.e., the theorem). In contrast, in many daily situations, proofs are considered secondary (in importance) to their consequence. These conflicting attitudes are well-coupled with the difference between written proofs and "interactive" proofs: If one values the proof itself then one may insist on having it archived, whereas if one only cares about the consequence then the way in which it is reached is immaterial.

Interestingly, the set of daily attitudes will be adequate in the current chapter, where *proofs are viewed merely as a vehicle for the verification of the validity of claims*. (This attitude gets to an extreme in the case of zero-knowledge proofs, where we actually require that the proofs themselves be useless beyond being convincing of the validity of the claimed assertion.) In general, we will be interested in modeling various forms of proofs, focusing on proofs that can be verified by automated procedures. These verification procedures are designed to check the validity of potential proofs, and are oblivious of additional features that appeal to humans such as beauty, insightfulness, etc. In the current section we will consider the most general form of proof systems that still allow efficient verification.

We note that the proof systems that we study refer to mundane theorems (e.g., asserting that a *specific* propositional formula is not satisfiable or that a party sent a message as instructed by a predetermined protocol). We stress that the (meta) theorems that we shall state regarding these proof systems will be proven in the traditional mathematical sense.

Prover and Verifier. The notion of a prover is implicit in all discussions of proofs, be it in mathematics or in other situations: the prover is the (sometimes hidden or transcendental) entity providing the proof. In contrast, the notion of a verifier tends to be more explicit in such discussions, which typically emphasize the *verification process*, or in other words the role of the verifier. Both in mathematics and in daily situations, proofs are defined in terms of the verification procedure. The verification procedure is considered to be relatively simple, and the burden is placed on the party/person supplying the proof (i.e., the prover). The asymmetry between the complexity of the verification task and the complexity of the theorem-proving task is captured by the definition of NP-proof systems (i.e., verification is required to be efficient, whereas $\mathcal{P} \neq \mathcal{NP}$ implies that in some cases finding adequate proofs is infeasible).

We highlight the "distrustful attitude" towards the prover, which underlies any proof system. If the verifier trusts the prover then no proof is needed. Hence, whenever discussing a proof system one considers a setting in which the verifier is not trusting the prover, and furthermore is skeptic of anything that the prover says. In such a setting the prover's goal is to convince the verifier, while the verifier should make sure it is not fooled by the prover.

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

We note that not every set of true statements has a "reasonable" proof system in which each of these statements can be proven (while no false statement can be "proven"). This fundamental phenomenon is given a precise meaning in results such as *Gödel's Incompleteness Theorem* and Turing's theorem regarding the *undecidability of the Halting Problem*. In contrast, recall that $\mathcal{NP}$ was defined as the class of sets having proof systems that support efficient deterministic verification (of "written proofs"). This section is devoted to the study of a more liberal notion of efficient verification procedures (allowing both randomization and interaction).

9.1.1 Definition

Loosely speaking, an interactive proof is a game between a computationally bounded verifier and a computationally unbounded prover whose goal is to convince the verifier of the validity of some assertion. Specifically, the verifier employs a probabilistic polynomial-time strategy. It is required that if the assertion holds then the verifier always accepts (i.e., when interacting with an appropriate prover strategy). On the other hand, if the assertion is false then the verifier must reject with probability at least $\frac{1}{2}$, no matter what strategy is being employed by the prover. (The error probability can be reduced by running such a proof system several times.)

Formally, a strategy for a party describes the *party's next move* (i.e., its next message or its final decision) *as a function of the common input* (i.e., the aforementioned assertion), *its internal coin tosses, and all messages it has received so far.* That is, we assume that each party records the outcomes of its past coin tosses as well as all the messages it has received, and determines its moves based on these. Thus, an interaction between two parties, employing strategies A and B respectively, is determined by the common input, denoted x, and the randomness of both parties, denoted r_A and r_B. Assuming that A takes the first move (and B takes the last one), the corresponding interaction transcript (on common input x and randomness r_A and r_B) is $\alpha_1, \beta_1, ..., \alpha_t, \beta_t$, where $\alpha_i = A(x, r_A, \beta_1, ..., \beta_{i-1})$ and $\beta_i = B(x, r_B, \alpha_1, ..., \alpha_i)$. The corresponding final decision of A is defined as $A(x, r_A, \beta_1, ..., \beta_t)$.

We say that a party employs a probabilistic polynomial-time strategy if its next move can be computed in a number of steps that is *polynomial in the length of the common input.* In particular, this means that, on input common input x, the strategy may only consider a polynomial in $|x|$ many messages, which are each of poly$(|x|)$ length.[2] Intuitively, if the other party exceeds an a priori (polynomial in $|x|$) bound on the total length of the messages that it is allowed to send, then the execution is suspended. Thus, referring to the aforementioned strategies, we say that A is a probabilistic polynomial-time strategy if, for every i and $r_A, \beta_1, ..., \beta_i$, the value of $A(x, r_A, \beta_1, ..., \beta_i)$ can be computed in time polynomial in $|x|$. Again, in proper use, it must hold that $|r_A|, t$ and the $|\beta_i|$'s are all polynomial in $|x|$.

Definition 9.1 (Interactive Proof systems – IP):[3] *An* interactive proof system for a set S *is a two-party game, between a* verifier *executing a* probabilistic polynomial-time strategy*, denoted* V*, and a* prover *that executes a* (computationally unbounded) *strategy, denoted* P*, satisfying the following two conditions:*

- Completeness: *For every* $x \in S$*, the verifier* V *always accepts after interacting with the prover* P *on common input* x*.*

[2] Needless to say, the number of internal coin tosses fed to a polynomial-time strategy must also be bounded by a polynomial in the length of x.

[3] We follow the convention of specifying strategies for both the verifier and the prover. An alternative presentation only specifies the verifier's strategy, while rephrasing the completeness condition as follows: *There exists a prover strategy* P *so that, for every* $x \in S$*, the verifier* V *always accepts after interacting with* P *on common input* x*.*

- Soundness: *For every $x \notin S$ and every strategy P^*, the verifier V rejects with probability at least $\frac{1}{2}$ after interacting with P^* on common input x.*

We denote by $\mathcal{IP}$ the class of sets having interactive proof systems.

The error probability (in the soundness condition) can be reduced by successive applications of the proof system. (This is easy to see in the case of sequential repetitions, but holds also for parallel repetitions; see Exercise 9.1.) In particular, repeating the proving process for k times, reduces the probability that the verifier is fooled (i.e., accepts a false assertion) to 2^{-k}, and we can afford doing so for any $k = \text{poly}(|x|)$. (Variants on the basic definition are discussed in Section 9.1.3.)

The role of randomness. Randomness is essential to the power of interactive proofs; that is, restricting the verifier to deterministic strategies yields a class of interactive proof systems that has no advantage over the class of NP-proof systems. The reason being that, in case the verifier is deterministic, the prover can predict the verifier's part of the interaction. Thus, the prover can just supply its own sequence of answers to the verifier's sequence of (predictable) questions, and the verifier can just check that these answers are convincing. Actually, we establish that soundness error (and not merely randomized verification) is essential to the power of interactive proof systems (i.e., their ability to reach beyond NP-proofs).

Proposition 9.2 *Suppose that S has an interactive proof system (P, V) with* no soundness error*; that is, for every $x \notin S$ and every potential strategy P^*, the verifier V rejects with probability* one *after interacting with P^* on common input x. Then $S \in \mathcal{NP}$.*

Proof: We may assume, without loss of generality, that V is deterministic (by just fixing arbitrarily the contents of its random-tape (e.g., to the all-zero string) and noting that both (perfect) completeness and perfect (i.e., errorless) soundness still hold). Since V is deterministic, the prover can predict each message sent by V (because each such message is uniquely determined by the common input and the previous prover messages). Thus, a sequence of optimal prover's messages (i.e., a sequence of messages leading V to accept x) can be (pre)determined (without interacting with V) *based solely on the common input x*. (Note that we do not care about the complexity of determining such a sequence, since no computational bounds are placed on the prover.) Formally, $x \in S$ if and only if there exists a sequence of (prover's) messages that make (the deterministic) V accept x, where the question of whether a specific sequence makes V accept x depends only on the sequence and on the common input x (because V tosses no coins that may affect this decision). It follows that $S \in \mathcal{NP}$. ■

Indeed, the punch-line of the foregoing proof is that the prover gains nothing from interacting with an easily predictable verifier (i.e., a verifier that determines its messages in deterministic polynomial-time based on the common input and the prover's prior messages).[4] The prover can just produce the entire interaction by itself (and send it to the verifier for verification). *The moral is is that there is no point to interact with a party whose moves are easily predictable.* This moral represents the prover's point of view (regarding deterministic verifiers). Certainly, from the verifier's point of view it is beneficial to interact with the prover, because the latter is computationally stronger (and thus its moves may not be easily predictable by the verifier even in case they are predictable in an information theoretic sense).

9.1.2 The Power of Interactive Proofs

We have seen that randomness is essential to the power of interactive proof systems in the sense that without randomness interactive proofs are not more powerful than NP-proofs. Indeed, the power of interactive proof arises from the combination of randomization and interaction. We first demonstrate this point by a simple proof system for a specific coNP-set that is not known to have an NP-proof system, and next prove the celebrated result $\mathcal{IP} = \mathcal{PSPACE}$, which suggests that interactive proofs are much stronger than NP-proofs.

9.1.2.1 A simple example

> *One day on the Olympus, bright-eyed Athena claimed that Nectar poured out of the new silver-coated jars tastes less good than Nectar poured out of the older gold-decorated jars. Mighty Zeus, who was forced to introduce the new jars by the practically oriented Hera, was annoyed at the claim. He ordered that Athena be served one hundred glasses of Nectar, each poured at random either from an old jar or from a new one, and that she tell the source of the drink in each glass. To everybody's surprise, wise Athena correctly identified the source of each serving, to which the Father of the Gods responded "my child, you are either right or extremely lucky." Since all gods knew that being lucky was not one of the attributes of Pallas-Athena, they all concluded that the impeccable goddess was right in her claim.*

The foregoing story illustrates the main idea underlying the interactive proof for Graph Non-Isomorphism, presented in Construction 9.3. Informally, this interactive proof system is designed for proving dissimilarity of two given objects (in the foregoing story these are the two brands of Nectar, whereas in Construction 9.3 these are two non-isomorphic graphs). We note that, typically, proving similarity between objects is easy, because one can present a mapping (of one object to the other) that demonstrates this similarity. In contrast, proving dissimilarity seems harder, because in general there seems to be no succinct proof of dissimilarity. More generally, it is typically easy to prove the existence of an easily verifiable structure in

[4]Note that knowledge of the verifier's messages may be essential for answering these questions convincingly. In the case that V is deterministic its messages can be determined by the prover, but this may not be possible in the general case (i.e., when V is randomized).

the given object by merely presenting this structure, but proving the non-existence of such a structure seems hard. Formally, membership in an NP-set is proved by presenting an NP-witness, but it is not clear how to prove the non-existence of such witness. Indeed, recall that the common belief is that co$\mathcal{NP} \neq \mathcal{NP}$.

Two graphs, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, are called isomorphic if there exists a 1-1 and onto mapping, ϕ, from the vertex set V_1 to the vertex set V_2 such that $\{u, v\} \in E_1$ if and only if $\{\phi(v), \phi(u)\} \in E_2$. This ("edge preserving") mapping ϕ, in case it exists, is called an *isomorphism* between the graphs. The following protocol specifies a way of proving that two graphs are not isomorphic, while it is not known whether such a statement can be proven via a non-interactive process (i.e., via an NP-proof system).

Construction 9.3 (Interactive proof for Graph Non-Isomorphism):

- Common Input: *A pair of graphs,* $G_1 = (V_1, E_1)$ *and* $G_2 = (V_2, E_2)$.
- Verifier's first step (V1): *The verifier selects at random one of the two input graphs, and sends to the prover a random isomorphic copy of this graph. Namely, the verifier selects uniformly* $\sigma \in \{1, 2\}$, *and a random permutation* π *from the set of permutations over the vertex set* V_σ. *The verifier constructs a graph with vertex set* V_σ *and edge set*

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

and sends (V_σ, E) *to the prover.*

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

The verifier's strategy in Construction 9.3 is easily implemented in probabilistic polynomial-time. We do not known of a probabilistic polynomial-time implementation of the prover's strategy, but this is not required. The motivating remark justifies the claim that Construction 9.3 constitutes an interactive proof system for the set of pairs of non-isomorphic graphs.[5] Recall that the latter is a co$\mathcal{NP}$-set (which is not known to be in $\mathcal{NP}$).

9.1.2.2 The full power of interactive proofs

The interactive proof system of Construction 9.3 refers to a specific coNP-set that is not known to be in $\mathcal{NP}$. It turns out that interactive proof systems are powerful enough to prove membership in *any* coNP-set (e.g., prove that a graph is not 3-colorable). Thus, assuming that $\mathcal{NP} \neq \text{co}\mathcal{NP}$, this establishes that interactive proof systems are more powerful than NP-proof systems. Furthermore, the class of sets having interactive proof systems coincides with the class of sets that can be decided using a polynomial amount of work-space.

Theorem 9.4 (The IP Theorem): $\mathcal{IP} = \mathcal{PSPACE}$.

Recall that it is widely believed that $\mathcal{NP}$ is a *proper* subset of $\mathcal{PSPACE}$. Thus, under this conjecture, interactive proofs are more powerful than NP-proofs.

Sketch of the Proof of Theorem 9.4

Theorem 9.4, was established using algebraic methods (see details below). In particular, the following approach – unprecedented in complexity theory – was employed: In order to demonstrate that a particular set is in a particular class, an arithmetic generalization of the Boolean problem is presented, and (elementary) algebraic methods are applied for showing that the arithmetic problem is solvable within the class. Following is a sketch of the proof. We first show that co$\mathcal{NP} \subseteq \mathcal{IP}$, by presenting an interactive proof system for the co$\mathcal{NP}$-complete set of non-satisfiable CNF formulae. Next we extend this proof system to obtain one for the $\mathcal{PSPACE}$-complete set of non-satisfiable Quantified Boolean Formulae. Finally, we observe that $\mathcal{IP} \subseteq \mathcal{PSPACE}$.

Teaching note: Our presentation focuses on the main ideas, and neglects various implementation details (which can be found in [151, 192]). Furthermore, we devote most of the presentation to establishing that co$\mathcal{NP} \subseteq \mathcal{IP}$, and recommend doing the same in class.

Arithmetization of Boolean (CNF) formulae: Given a Boolean (CNF) formula, we replace the Boolean variables by integer variables, and replace the logical operations by corresponding arithmetic operations. In particular, OR-clauses are replaced by sums, and the top level conjunction is replaced by a product. Then, we consider the formal summation of the resulting arithmetic expression, where

[5] In case G_1 is not isomorphic to G_2, no graph can be isomorphic to both input graphs (i.e., both to G_1 and to G_2). In this case the graph G' sent in Step (V1) uniquely determines the bit σ. On the other hand, if G_1 and G_2 are isomorphic then, for every G' sent in Step (V1), the number of isomorphisms between G_1 and G' equals the number of isomorphisms between G_2 and G'. It follows that, in this case G', yields no information about σ (chosen by the verifier), and so no prover may convince the verifier with probability exceeding $1/2$.

summation is taken over all 0-1 assignments to its variables. For example, the Boolean formula

$$(x_3 \vee \neg x_5 \vee x_{17}) \wedge (x_5 \vee x_9) \wedge (\neg x_3 \vee \neg x_4)$$

is replaces by the arithmetic expression

$$(x_3 + (1 - x_5) + x_{17}) \cdot (x_5 + x_9) \cdot ((1 - x_3) + (1 - x_4))$$

and *the Boolean formula is non-satisfiable if and only if the sum of the arithmetic expression, taken over all choices of* $x_1, x_2, ..., x_{17} \in \{0,1\}$*, equals 0.* Thus, proving that the original Boolean formula is non-satisfiable reduces to proving that the corresponding arithmetic summation evaluates to 0. We highlight two additional observations regarding the resulting arithmetic expression:

1. The arithmetic expression is a low degree polynomial over the integers; specifically, its (total) degree equals the number of clauses in the original Boolean formula.

2. For any Boolean formula, the value of the corresponding arithmetic expression (for any choice of $x_1, ..., x_n \in \{0,1\}$) resides within the interval $[0, v^m]$, where v is the maximum number of variables in a clause, and m is the number of clauses. Thus, summing over all 2^n possible 0-1 assignments, where $n \leq vm$ is the number of variables, the result resides in $[0, 2^n v^m]$.

Moving to a Finite Field: Whenever we need to check equality between two integers in $[0, M]$, it suffices to check their equality mod q, where $q > M$. The benefit is that the arithmetic is now in a finite field (mod q), and so certain things are "nicer" (e.g., uniformly selecting a value). Thus, proving that a CNF formula is not satisfiable reduces to proving an equality of the following form

$$\sum_{x_1=0,1} \cdots \sum_{x_n=0,1} \phi(x_1, ..., x_n) \equiv 0 \pmod{q}, \tag{9.1}$$

where ϕ is a low degree multi-variate polynomial. In the rest of this exposition, all arithmetic operations refer to the finite field of q elements, denoted $\mathrm{GF}(q)$.

Overview of the actual protocol: stripping summations in iterations. Given a formal expression as in Eq. (9.1), we strip off summations in iterations, stripping a single summation at each iteration, and instantiate the corresponding free variable as follows. At the beginning of each iteration the prover is supposed to supply the univariate polynomial representing the residual expression as a function of the (single) currently stripped variable. (By Observation 1, this is a low degree polynomial and so it has a short description.)[6] The verifier checks that the

[6] We also use Observation 2, which implies that we may use a finite field with elements having a description length that is polynomial in the length of the original Boolean formula (i.e., $\log_2 q = O(vm)$).

polynomial (say, p) is of low degree, and that it corresponds to the current value (say, v) being claimed (i.e., it verifies that $p(0)+p(1) \equiv v$). Next, the verifier randomly instantiates the currently free variable (i.e., it selects uniformly $r \in \mathrm{GF}(q)$), yielding a new value to be claimed for the resulting expression (i.e., the verifier computes $v \leftarrow p(r)$, and expects a proof that the residual expression equals v). The verifier sends the uniformly chosen instantiation (i.e., r) to the prover, and the parties proceed to the next iteration (which refers to the residual expression and to the new value v). At the end of the last iteration, the verifier has a closed form expression (i.e., an expression without formal summations), which can be easily checked against the claimed value.

A single iteration (detailed): The i^{th} iteration is aimed at proving a claim of the form

$$\sum_{x_i=0,1} \cdots \sum_{x_n=0,1} \phi(r_1,...,r_{i-1},x_i,x_{i+1},...,x_n) \equiv v_{i-1} \pmod{q}, \tag{9.2}$$

where $v_0 = 0$, and $r_1,...,r_{i-1}$ and v_{i-1} are as determined in previous iterations. The i^{th} iteration consists of two steps (messages): a prover step followed by a verifier step. The prover is supposed to provide the verifier with the univariate polynomial p_i that satisfies

$$p_i(z) \stackrel{\text{def}}{=} \sum_{x_{i+1}=0,1} \cdots \sum_{x_n=0,1} \phi(r_1,...,r_{i-1},z,x_{i+1},...,x_n) \bmod q\,. \tag{9.3}$$

Denote by p_i' the actual polynomial sent by the prover (i.e., the honest prover sets $p_i' = p_i$). Then, the verifier first checks if $p_i'(0)+p_i'(1) \equiv v_{i-1} \pmod{q}$, and next uniformly selects $r_i \in \mathrm{GF}(q)$ and sends it to the prover. Needless to say, the verifier will reject if the first check is violated. The claim to be proven in the next iteration is

$$\sum_{x_{i+1}=0,1} \cdots \sum_{x_n=0,1} \phi(r_1,...,r_{i-1},r_i,x_{i+1},...,x_n) \equiv v_i \pmod{q}, \tag{9.4}$$

where $v_i \stackrel{\text{def}}{=} p_i'(r_i) \bmod q$.

Completeness of the protocol: When the initial claim (i.e., Eq. (9.1)) holds, the prover can supply the correct polynomials (as determined in Eq. (9.3)), and this will lead the verifier to always accept.

Soundness of the protocol: It suffices to upper-bound the probability that, for a particular iteration, the entry claim (i.e., Eq. (9.2)) is false while the ending claim (i.e., Eq. (9.4)) is valid. Both claims refer to the current summation expression being equal to the current value, where 'current' means either at the beginning of the iteration or at its end. Let $p(\cdot)$ be the actual polynomial representing the expression when stripping the current variable, and let $p'(\cdot)$ be any potential answer by the prover. We may assume that $p'(0)+p'(1) \equiv v \pmod{q}$ and that p' is of low-degree (as otherwise the verifier will reject). Using our hypothesis (that the entry claim of Eq. (9.2) is false), we know that $p(0)+p(1) \not\equiv v \pmod q$. Thus, p' and p are different low-degree polynomials, and so they may agree on very few points (if at all). In case the verifier instantiation (i.e., its choice of random r) does not happen to be one of these few points, the ending claim (i.e., Eq. (9.4)) is false too (because $p(r) \not\equiv p'(r) \pmod q$, whereas the new value is set to $p'(r) \bmod q$ and the residual expression evaluates to $p(r)$). Details are left as an exercise (see Exercise 9.2).

This establishes that the set of non-satisfiable CNF formulae has an interactive proof system. Actually, a similar proof system (which uses a related arithmetization – see Exercise 9.4) can be used to prove that a given formula has a given number of satisfying assignment; i.e., prove membership in the ("counting") set

$$\{(\phi, k) : |\{\tau : \phi(\tau) = 1\}| = k\}. \tag{9.5}$$

Using adequate reductions, it follows that every problem in $\#\mathcal{P}$ has an interactive proof system (i.e., for every $R \in \mathcal{PC}$, the set $\{(x,k) : |\{y : (x,y) \in R\}| = k\}$ is in $\mathcal{IP}$). Proving that $\mathcal{PSPACE} \subseteq \mathcal{IP}$ requires a little more work.

Interactive Proofs for PSPACE (basic idea). We present an interactive proof for the set of satisfied Quantified Boolean Formulae (QBF), which is complete for $\mathcal{PSPACE}$ (see Theorem 5.15).[7] Recall that the number of quantifiers in such formulae is unbounded (e.g., it may be polynomially related to the length of the input), that there are both existential and universal quantifiers, and furthermore these quantifiers may alternate. In the arithmetization of these formulae, we replace existential quantifiers by summations and universal quantifiers by products. Two difficulties arise when considering the application of the forgoing protocol to the resulting arithmetic expression. Firstly, the value of the expression (which may involve a big number of nested formal products) is only upper-bounded by a double exponential function (in the length of the input). Secondly, when stripping a summation (or a product), the expression may be a polynomial of high degree (due to nested formal products that may appear in the remaining expression). For example, both phenomena occur in the following expression

$$\sum_{x=0,1} \prod_{y_1=0,1} \cdots \prod_{y_n=0,1} (x+y_n),$$

which equals $\sum_{x=0,1} x^{2^{n-1}} \cdot (1+x)^{2^{n-1}}$. The first difficulty is easy to resolve by using the fact (to be established in Exercise 9.6) that if two integers in $[0, M]$ are different then they must be different modulo most of the primes in the interval $[3, \mathrm{poly}(\log M)]$. Thus, we let the verifier selects a random prime q of length that is linear in the length of the original formula, and the two parties consider the arithmetic expression reduced modulo this q. The second difficulty is resolved

[7]Actually, the following extension of the foregoing proof system yields a proof system for the set of *unsatisfied* Quantified Boolean Formulae (which is also complete for $\mathcal{PSPACE}$). Alternatively, one may extend the related proof system presented in Exercise 9.4.

by noting that $\mathcal{PSPACE}$ is actually reducible to a special form of QBF in which no variable appears both to the left and to the right of more than one universal quantifier (see the proof of Theorem 5.15 or alternatively Exercise 9.5). It follows that when arithmetizing and stripping summations (or products) from the resulting arithmetic expression, the corresponding univariate polynomial is of low degree (i.e., at most twice the length of the original formula, where the factor of two is due to the single universal quantifier that has this variable quantified on its left and appearing on its right).

IP is contained in PSPACE: We shall show that, for every interactive proof system, there exists an *optimal prover strategy* that can be implemented in polynomial-space, where an optimal prover strategy is one that maximizes the probability that the prescribed verifier accepts the common input. It follows that $\mathcal{IP} \subseteq \mathcal{PSPACE}$, because (for every $S \in \mathcal{IP}$) we can emulate the interaction of the prescribed verifier with an optimal prover strategy in polynomial space.

Proposition 9.5 *Let V be a probabilistic polynomial-time interactive machine. Then, there exists a polynomial-space computable prover strategy f that, for every x maximizes the probability that V accepts x. That is, for every P^* and every x it holds that the probability that V accepts x after interacting with P^* is upper-bounded by the probability that V accepts x after interacting with f.*

Proof Sketch: For every common input x and any possible partial transcript γ of the interaction so far, the strategy[8] f determines an optimal next message for the prover by considering all possible coin tosses of the verifier that are consistent with (x, γ). Specifically, f is determined recursively such that $f(x, \gamma) = m$ if m maximizes the number of verifier coins that are consistent with (x, γ) and lead the verifier to accept when subsequent prover moves are determined by f (which is where recursion is used). That is, coins r support the setting $f(x, \gamma) = m$, where $\gamma = (\alpha_1, \beta_1, ..., \alpha_t, \beta_t)$, if the following two conditions hold:

1. r is consistent with (x, γ), which means that for every $i \in \{1, ..., t\}$ it holds that $\beta_i = V(x, r, \alpha_1, ..., \alpha_i)$.
2. r leads V to accept (when subsequent prover moves are determined by f), which means that $V(x, r, \alpha_1, ..., \alpha_t, m, \alpha_{t+2}, ..., \alpha_T) = 1$, where for every $i \in \{t+1, ..., T-1\}$ it holds that $\alpha_{i+1} = f(x, \gamma, m, \beta_{t+1}, ..., \alpha_i, \beta_i)$ and $\beta_i = V(x, r, \alpha_1, ..., \alpha_t, m, \alpha_{t+2}, ..., \alpha_i)$.

That is, $f(x, \gamma) = m$ if m maximizes the value of $\mathsf{E}[f(x, \gamma, m, V(x, R_\gamma, m))]$, where R_γ is selected uniformly among the r's that are consistent with (x, γ). Thus, the value $f(x, \gamma)$ can be computed in polynomial-space when given oracle access to $f(x, \gamma, \cdot, \cdot)$, and the proposition follows by standard composition of space-bounded computations. $\square$

[8] For sake of convenience, when describing the strategy f, we refer to the entire partial transcript of the interaction with V (rather than merely to the sequence of previous messages sent by V).

9.1.3 Variants and finer structure: an overview

In this subsection we consider several variants on the basic definition of interactive proofs as well as finer complexity measures. This is an advanced subsection, which only provides an overview of the various notions and results (as well as pointers to proofs of the latter).

9.1.3.1 Arthur-Merlin games a.k.a public-coin proof systems

The verifier's messages in a general interactive proof system are determined arbitrarily (but efficiently) based on the verifier's view of the interaction so far (which includes its internal coin tosses, which without loss of generality can take place at the onset of the interaction). Thus, the verifier's past coin tosses are not necessarily revealed by the messages that it sends. In contrast, in public-coin proof systems (a.k.a Arthur-Merlin proof systems), the verifier's messages contain the outcome of any coin that it tosses at the current round. Thus, these messages reveal the randomness used towards generating them (i.e., this randomness becomes public). Actually, without loss of generality, the verifier's messages can be identical to the outcome of the coins tossed at the current round (because any other string that the verifier may compute based on these coin tosses is actually determined by them). Note that the proof systems presented in the proof of Theorem 9.4 are of the public-coin type, whereas this is not the case for the Graph Non-Isomorphism proof system (of Construction 9.3). Thus, although not all natural proof systems are of the public-coin type, every set having an interactive proof system also has a public-coin interactive proof system. This means that, *in the context of interactive proof systems, asking random questions is as powerful as asking clever questions.*

Indeed, public-coin proof systems are a syntactically restricted type of interactive proof systems. This restriction may make the design of such systems more complex, but potentially facilitates their analysis (and especially the analysis of a generic system). Another advantage of public-coin proof systems is that the verifier's actions (except for its final decision) are oblivious of the prover's messages. This property is used in the proof of Theorem 9.12.

9.1.3.2 Interactive proof systems with two-sided error

In Definition 9.1 error probability is allowed in the soundness condition but not in the completeness condition. In such a case, we say that the proof system has perfect completeness (or one-sided error probability). A more general definition allows an error probability (upper-bounded by, say, $1/3$) in both the completeness and soundness conditions. Note that sets having such generalized (two-sided error) interactive proofs are also in $\mathcal{PSPACE}$, and thus allowing two-sided error does not increase the power of interactive proofs. See further discussion at the end of §9.1.3.3.

9.1.3.3 A hierarchy of interactive proof systems

Definition 9.1 only refers to the *total* computation time of the verifier, and thus allows an arbitrary (polynomial) number of messages to be exchanged. A finer definition refers to the number of messages being exchanged (also called the number of rounds).[9]

Definition 9.6 (The round-complexity of interactive proof):

- *For an integer function m, the complexity class $\mathcal{IP}(m)$ consists of sets having an interactive proof system in which, on common input x, at most $m(|x|)$ messages are exchanged between the parties.*[10]
- *For a set of integer functions, M, we let $\mathcal{IP}(M) \stackrel{\rm def}{=} \bigcup_{m\in M} \mathcal{IP}(m)$. Thus, $\mathcal{IP} = \mathcal{IP}(\texttt{poly})$.*

For example, interactive proof systems in which the verifier sends a single message that is answered by a single message of the prover corresponds to $\mathcal{IP}(2)$. Clearly, $\mathcal{NP} \subseteq \mathcal{IP}(1)$, yet the inclusion may be strict because in $\mathcal{IP}(1)$ the verifier may toss coins after receiving the prover's single message. (Also note that $\mathcal{IP}(0) = \text{co}\mathcal{RP}$.) Concerning the finer structure of the IP-hierarchy, the following is known:

- A linear speed-up (see Appendix F.2 (or [20] and [107])): For every integer function, f, such that $f(n) \geq 2$ for all n, the class $\mathcal{IP}(O(f(\cdot)))$ collapses to the class $\mathcal{IP}(f(\cdot))$. In particular, $\mathcal{IP}(O(1))$ collapses to $\mathcal{IP}(2)$.
- The class $\mathcal{IP}(2)$ contains sets not known to be in $\mathcal{NP}$; e.g., Graph Non-Isomorphism (see Construction 9.3). However, under plausible intractability assumptions, $\mathcal{IP}(2) = \mathcal{NP}$ (see [156]).
- If $\text{co}\mathcal{NP} \subseteq \mathcal{IP}(2)$ then the Polynomial-Time Hierarchy collapses (see [42]).

It is conjectured that $\text{co}\mathcal{NP}$ is *not* contained in $\mathcal{IP}(2)$, and consequently that interactive proofs with an unbounded number of message exchanges are more powerful than interactive proofs in which only a bounded (i.e., constant) number of messages are exchanged.[11] The class $\mathcal{IP}(1)$ (also denoted $\mathcal{MA}$) seems to be *the* "real" randomized (and yet non-interactive) version of $\mathcal{NP}$: Here the prover supplies a candidate (polynomial-size) "proof", and the verifier assesses its validity probabilistically (rather than deterministically).

The IP-hierarchy (i.e., $\mathcal{IP}(\cdot)$) equals an analogous hierarchy, denoted $\mathcal{AM}(\cdot)$, that refers to public-coin (a.k.a Arthur-Merlin) interactive proofs. That is, for every integer function f, it holds that $\mathcal{AM}(f) = \mathcal{IP}(f)$. For $f \geq 2$, it is also the case that $\mathcal{AM}(f) = \mathcal{AM}(O(f))$; actually, the aforementioned linear speed-up for $\mathcal{IP}(\cdot)$ is established by combining the following two results:

[9] An even finer structure emerges when considering also the total length of the messages sent by the prover (see [102]).

[10] We count the total number of messages exchanged regardless of the direction of communication.

[11] Note that the linear speed-up cannot be applied for an unbounded number of times, because each application may increase (e.g., square) the time-complexity of verification.

1. Emulating $\mathcal{IP}(\cdot)$ by $\mathcal{AM}(\cdot)$ (see §F.2.1 or [107]): $\mathcal{IP}(f) \subseteq \mathcal{AM}(f+3)$.
2. Linear speed-up for $\mathcal{AM}(\cdot)$ (see §F.2.2 or [20]): $\mathcal{AM}(2f) \subseteq \mathcal{AM}(f)$.

In particular, $\mathcal{IP}(O(1)) = \mathcal{AM}(2)$, even if $\mathcal{AM}(2)$ is restricted such that the verifier tosses no coins after receiving the prover's message. (Note that $\mathcal{IP}(1) = \mathcal{AM}(1)$ and $\mathcal{IP}(0) = \mathcal{AM}(0)$ are trivial.) We comment that it is common to denote $\mathcal{AM}(2)$ by $\mathcal{AM}$, which is indeed inconsistent with the convention of using $\mathcal{IP}$ to denote $\mathcal{IP}(\texttt{poly})$.

The fact that $\mathcal{IP}(O(f)) = \mathcal{IP}(f)$ is proved by establishing an analogous result for $\mathcal{AM}(\cdot)$ demonstrates the advantage of the public-coin setting for the study of interactive proofs. A similar phenomenon occurs when establishing that the IP-hierarchy equals an analogous two-sided error hierarchy (see Exercise 9.7).

9.1.3.4 Something completely different

We stress that although we have relaxed the requirements from the verification procedure (by allowing it to interact with the prover, toss coins, and risk some (bounded) error probability), we did not restrict the validity of its assertions by assumptions concerning the potential prover. This should be contrasted with other notions of proof systems, such as computationally-sound ones (see §9.1.4.2), in which the validity of the verifier's assertions depends on assumptions concerning the potential prover(s).

9.1.4 On computationally bounded provers: an overview

Recall that our definition of interactive proofs (i.e., Definition 9.1) makes no reference to the computational abilities of the potential prover. This fact has two conflicting consequences:

1. The completeness condition does not provide any upper bound on the complexity of the corresponding proving strategy (which convinces the verifier to accept valid assertions).
2. The soundness condition guarantees that, regardless of the computational effort spend by a cheating prover, the verifier cannot be fooled to accept invalid assertions (with probability exceeding the soundness error).

Note that providing an upper-bound on the complexity of the (prescribed) prover strategy P of a specific interactive proof system (P, V) only strengthens the claim that (P, V) is a proof system for the corresponding set (of valid assertions). We stress that the prescribed prover strategy is referred to only in the completeness condition (and is irrelevant to the soundness condition). On the other hand, relaxing the definition of interactive proofs such that soundness holds only for a specific class of cheating prover strategies (rather than for all cheating prover strategies) weakens the corresponding claim. In this advanced section we consider both possibilities.

Teaching note: Indeed, this is an advanced subsection, which is best left for independent reading. It merely provides an overview of the various notions, and the reader is directed to the chapter's notes for further detail (i.e., pointers to the relevant literature).

9.1.4.1 How powerful should the prover be?

Assume that a set S is in $\mathcal{IP}$. This means that there is a verifier V that can be convinced to accept any input in S but cannot be fooled to accept any input not in S (except with small probability). One may ask how powerful should a prover be such that it can convince the verifier V to accept any input in S. Note that Proposition 9.5 asserts that an optimal prover strategy can be implemented in polynomial-space (and that we cannot expect better for a generic set in $\mathcal{PSPACE} = \mathcal{IP}$), but we will seek better upper-bounds on the complexity of the prover that convinces a specific verifier (which in turn corresponds to a specific set S). More interestingly, considering all possible verifiers that give rise to interactive proof systems for S, we ask what is the minimum power required from a prover that satisfies the completeness requirement with respect to one of these verifiers?

We stress that, unlike the case of computationally-sound proof systems (see §9.1.4.2), we do not restrict the power of the prover in the soundness condition, but rather consider the minimum complexity of provers meeting the completeness condition. Specifically, we are interested in *relatively efficient* provers that meet the completeness condition. The term "relatively efficient prover" has been given three different interpretations, which are briefly surveyed next.

1. A prover is considered *relatively efficient* if, when given an auxiliary input (in addition to the common input in S), it works in (probabilistic) polynomial-time. Specifically, in case $S \in \mathcal{NP}$, the auxiliary input maybe an NP-proof that the common input is in the set. Still, even in this case the interactive proof need not consist of the prover sending the auxiliary input to the verifier; for example, an alternative procedure may allow the prover to be zero-knowledge (see Construction 9.10).

   This interpretation is adequate and in fact crucial for applications in which such an auxiliary input is available to the otherwise polynomial-time parties. Typically, such auxiliary input is available in cryptographic applications in which parties wish to prove in (zero-knowledge) that they have correctly conducted some computation. In these cases the NP-proof is just the transcript of the computation by which the claimed result has been generated, and thus the auxiliary input is available to the proving party.

2. A prover is considered *relatively efficient* if it can be implemented by a probabilistic polynomial-time oracle machine with oracle access to the set S itself. (Note that the prover in Construction 9.3 has this property.)

   This interpretation generalizes the notion of self-reducibility of NP-sets. (Recall that by self-reducibility of an NP-set we mean that the search problem of finding an NP-witness is polynomial-time reducible to deciding membership in the set (cf. Definition 2.13).)

3. A prover is considered *relatively efficient* if it can be implemented by a probabilistic machine that runs in time that is polynomial in the deterministic complexity of the set. This interpretation relates the difficulty of convincing a "lazy verifier" to the complexity of finding the truth alone.

   Hence, in contrast to the first interpretation, which is adequate in settings where assertions are generated along with their NP-proofs, the current interpretation is adequate in settings in which the prover is given only the assertion and has to find a proof to it by itself (before trying to convince a lazy verifier of its validity).

9.1.4.2 Computational-soundness

Relaxing the soundness condition such that it only refers to relatively-efficient ways of trying to fool the verifier (rather than to all possible ways) yields a fundamentally different notion of a proof system. Assertions proven in such a system are not necessarily correct; they are correct only if the potential cheating prover does not exceed the presumed complexity limits. As in §9.1.4.1, the notion of "relative efficiency" can be given different interpretations, the most popular one being that the cheating prover strategy can be described by a (non-uniform) family of polynomial-size circuits. The latter interpretation coincides with the first interpretation used in §9.1.4.1 (i.e., a probabilistic polynomial-time strategy that is given an auxiliary input (of polynomial length)). Specifically, the soundness condition is replaced by the following computational soundness condition that asserts that it is infeasible to fool the verifier into accepting false statements. Formally:

> For every prover strategy that is implementable by a family of polynomial-size circuits $\{C_n\}$, and every sufficiently long $x \in \{0,1\}^* \setminus S$, the probability that V accepts x when interacting with $C_{|x|}$ is less than $1/2$.

As in case of standard soundness, the computational-soundness error can be reduced by repetitions. We warn, however, that unlike in the case of standard soundness (where both sequential and parallel repetitions will do), the computational-soundness error cannot always be reduced by parallel repetitions.

It is common and natural to consider proof systems in which the prover strategies considered both in the completeness and soundness conditions satisfy the same notion of relative efficiency. Protocols that satisfy these conditions with respect to the foregoing interpretation are called arguments. We mention that argument systems may be more efficient (e.g., in terms of their communication complexity) than interactive proof systems.

9.2 Zero-Knowledge Proof Systems

Zero-Knowledge proofs are fascinating and extremely useful constructs. Their fascinating nature is due to their seemingly contradictory definition: zero-knowledge proofs are both convincing and yet yield nothing beyond the validity of the assertion being proven. Their applicability in the domain of cryptography is vast; they are typically used to force malicious parties to behave according to a predetermined protocol. In addition to their direct applicability in Cryptography, zero-knowledge proofs serve as a good bench-mark for the study of various problems regarding cryptographic protocols. In this section we focus on the conceptual contents of zero-knowledge, and relegate their cryptographic applications to Appendix C.

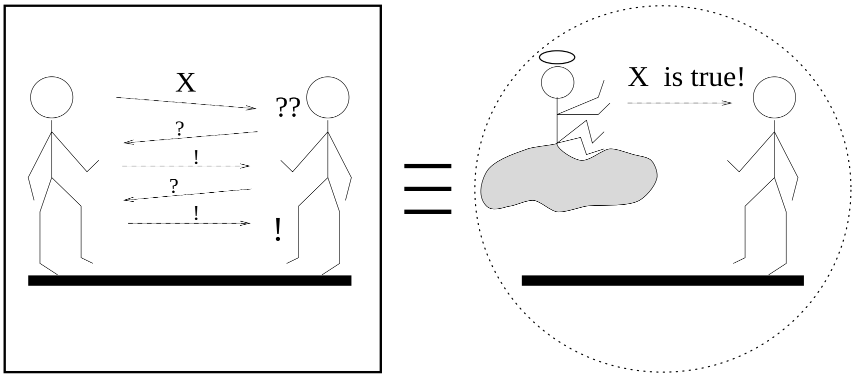


Figure 9.1: Zero-knowledge proofs – an illustration.

Turning back to the conceptual angle, we highlight the fact that standard proofs are believed to yield knowledge and not merely establish the validity of the assertion being proven. Indeed, it is commonly believed that (good) proofs provide a deeper understanding of the theorem being proved. At the technical level, an NP-proof of membership in some set $S \in \mathcal{NP} \setminus \mathcal{P}$ yields something (i.e., the NP-proof itself) that is typically hard to compute (even when assuming that the input is in S). For example, a 3-coloring of a graph is an NP-proof that the graph is 3-colorable, but it yields information (i.e., the coloring) that is infeasible to compute (when given an arbitrary 3-colorable graph). In contrast to such NP-proofs, which seem to yield a lot of knowledge, zero-knowledge proofs yield no knowledge at all; that is, the latter exhibit an extreme contrast between being convincing (of the validity of a statement) and teaching anything on top of the validity of the statement.

Teaching note: We believe that the treatment of zero-knowledge proofs provided in this section suffices for the purpose of a course in complexity theory. For an extensive treatment of zero-knowledge proofs, the interested reader is referred to [87, Chap. 4].

9.2.1 Definitional Issues

Loosely speaking, zero-knowledge proofs are proofs that yield nothing beyond the validity of the assertion; that is, a verifier obtaining such a proof only gains conviction in the validity of the assertion. This is formulated by saying that anything that can be feasibly obtained from a zero-knowledge proof is also feasibly computable from the (valid) assertion itself. The latter formulation follows the simulation paradigm, which is discussed next.

9.2.1.1 A wider perspective: the simulation paradigm

In defining zero-knowledge proofs, we view the verifier as a potential adversary that tries to gain knowledge from the (prescribed) prover.[12] We wish to state that no (feasible) adversary strategy for the verifier can gain anything from the prover (beyond conviction in the validity of the assertion). Let us consider the desired formulation from a wide perspective.

A key question regarding the modeling of security concerns is how to express the intuitive requirement that an adversary "gains nothing substantial" by deviating from the prescribed behavior of an honest user. Our approach is that the adversary *gains nothing* if whatever it can obtain by unrestricted adversarial behavior can be obtained within essentially the same computational effort by a benign behavior. The definition of the "benign behavior" captures what we want to achieve in terms of security, and is specific to the security concern to be addressed. For example, in the context of zero-knowledge, a benign behavior is any computation that is based (only) on the assertion itself (while assuming that the latter is valid). Thus, a zero-knowledge proof is an interactive proof in which no feasible adversarial verifier strategy can obtain from the interaction more than a "benign verifier" (which believes the assertion) can obtain from the assertion itself. We comment that the simulation paradigm is pivotal to many definitions in cryptography (e.g., it underlies the definition of security of encryption schemes and cryptographic protocols); for further details see Appendix C.

9.2.1.2 The basic definitions

Zero-knowledge is a property of some prover strategies. More generally, zero-knowledge is a property of some interactive machines. Fixing an interactive machine (e.g., a prescribed prover), we consider what can be gained (i.e., computed) by an *arbitrary feasible adversary* (e.g., a verifier) *that interacts with the aforementioned fixed machine* on a common input taken from a predetermined set (in our case the set of valid assertions). This gain is compared against what can be computed by an *arbitrary feasible algorithm* (called a simulator) that is only given the input itself. The fixed machine is zero-knowledge if the "computational power" of these two (fundamentally different settings) is essentially equivalent. Details follow.

The formulation of the zero-knowledge condition refers to two types of probability ensembles, where each ensemble associates a single probability distribution to each relevant input (e.g., a valid assertion). Specifically, in the case of interactive proofs, the first ensemble represents the output distribution of the verifier after interacting with the specified prover strategy P (on some common input), where

[12] Recall that when defining a proof system (e.g., an interactive proof system), we view the prover as a potential adversary that tries to fool the (prescribed) verifier (into accepting invalid assertions).

the verifier is employing an arbitrary efficient strategy (not necessarily the specified one). The second ensemble represents the output distribution of some probabilistic polynomial-time algorithm (which is only given the corresponding input but does not interact with anyone). The basic paradigm of zero-knowledge asserts that for every ensemble of the first type there exist a "similar" ensemble of the second type. The specific variants differ by the interpretation given to the notion of *similarity*. The most strict interpretation, leading to perfect zero-knowledge, is that similarity means equality.

Definition 9.7 (perfect zero-knowledge, over-simplified):[13] *A prover strategy, P, is said to be* perfect zero-knowledge over a set S *if for every probabilistic polynomial-time verifier strategy, V^*, there exists a probabilistic polynomial-time algorithm, M^*, such that*

$$(P, V^*)(x) \equiv M^*(x), \qquad \textit{for every } x \in S$$

where $(P, V^*)(x)$ is a random variable representing the output of verifier V^* after interacting with the prover P on common input x, and $M^*(x)$ is a random variable representing the output of machine M^* on input x.

We comment that any set in co$\mathcal{RP}$ has a perfect zero-knowledge proof system in which the prover keeps silence and the verifier decides by itself. The same holds for $\mathcal{BPP}$ provided that we relax the definition of interactive proof system to allow two-sided error. Needless to say, our focus is on non-trivial proof systems; that is, proof systems for sets outside of $\mathcal{BPP}$.

A somewhat more relaxed interpretation (of the notion of similarity), leading to almost-perfect zero-knowledge (a.k.a statistical zero-knowledge), is that similarity means statistical closeness (i.e., negligible difference between the ensembles). The most liberal interpretation, leading to the standard usage of the term zero-knowledge (and sometimes referred to as computational zero-knowledge), is that similarity means computational indistinguishability (i.e., failure of any efficient procedure to tell the two ensembles apart). Combining the foregoing discussion with the relevant definition of computational indistinguishability (i.e., Definition C.5), we obtain the following definition.

Definition 9.8 (zero-knowledge, somewhat simplified): *A prover strategy, P, is said to be* zero-knowledge over a set S *if for every probabilistic polynomial-time verifier strategy, V^*, there exists a probabilistic polynomial-time* simulator, M^*, *such that for every probabilistic polynomial-time* distinguisher, D, *it holds that*

$$d(n) \stackrel{\text{def}}{=} \max_{x \in S \cap \{0,1\}^n} \{|\mathsf{Pr}[D(x, (P, V^*)(x)) = 1] - \mathsf{Pr}[D(x, M^*(x)) = 1]|\}$$

[13] In the actual definition one relaxes the requirement in one of the following two ways. The first alternative is allowing M^* to run for *expected* (rather than strict) polynomial-time. The second alternative consists of allowing M^* to have no output with probability at most $1/2$ and considering the value of its output conditioned on it having output at all. The latter alternative implies the former, but the converse is not known to hold.

is a negligible function.[14] *We denote by $\mathcal{ZK}$ the class of sets having zero-knowledge interactive proof systems.*

Definition 9.8 is a simplified version of the actual definition, which is presented in Appendix C.4.2. Specifically, in order to guarantee that zero-knowledge is preserved under sequential composition it is necessary to slightly augment the definition (by providing V^* and M^* with the same value of an arbitrary (poly($|x|$)-bit long) auxiliary input). Other definitional issues and related notions are briefly discussed in Appendix C.4.4.

On the role of randomness and interaction. It can be shown that only sets in $\mathcal{BPP}$ have zero-knowledge proofs in which the verifier is deterministic (see Exercise 9.9). The same holds for deterministic provers, provided that we consider "auxiliary-input" zero-knowledge (as in Definition C.9). It can also be shown that only sets in $\mathcal{BPP}$ have zero-knowledge proofs in which a single message is sent (see Exercise 9.10). Thus, both randomness and interaction are essential to the non-triviality of zero-knowledge proof systems. (For further details, see [87, Sec. 4.5.1].)

Advanced Comment: Knowledge Complexity. Zero-knowledge is the lowest level of a knowledge-complexity hierarchy which quantifies the "knowledge revealed in an interaction." Specifically, the knowledge complexity of an interactive proof system may be defined as the minimum number of oracle-queries required in order to efficiently simulate an interaction with the prover. (See [86, Sec. 2.3.1] for references.)

9.2.2 The Power of Zero-Knowledge

When faced with a definition as complex (and seemingly self-contradictory) as the definition of zero-knowledge, one should indeed wonder whether the definition can be met (in a non-trivial manner).[15] It turns out that the existence of non-trivial zero-knowledge proofs is related to the existence of intractable problems in $\mathcal{NP}$. In particular, we will show that if one-way functions exist then every NP-set has a zero-knowledge proof system. (For the converse, see [87, Sec. 4.5.2] or [214].) We first demonstrate the scope of zero-knowledge by a presenting a simple (perfect) zero-knowledge proof system for a specific NP-set that is not known to be in $\mathcal{BPP}$. In this case we make no intractability assumptions, but the result is significant only if $\mathcal{NP}$ is not contained in $\mathcal{BPP}$.

9.2.2.1 A simple example

> *A story not found in the Odyssey refers to the not so famous Labyrinth of the Island of Aeaea. The Sorceress Circe, daughter of Helius, chal-*

[14] That is, d vanishes faster that the reciprocal of any positive polynomial (i.e., for every positive polynomial p and for sufficiently large n, it holds that $d(n) < 1/p(n)$). Needless to say, $d(n) \stackrel{\text{def}}{=} 0$ if $S \cap \{0,1\}^n = \emptyset$.

[15] Note that any set in $\mathcal{BPP}$ has a trivial zero-knowledge (two-sided error) proof system in which the verifier just determines membership by itself.

lenged godlike Odysseus to traverse the Labyrinth from its North Gate to its South Gate. Canny Odysseus doubted whether such a path existed at all and asked beautiful Circe for a proof, to which she replied that if she showed him a path this would trivialize for him the challenge of traversing the Labyrinth. "Not necessarily," clever Odysseus replied, "you can use your magic to transport me to a random place in the labyrinth, and then guide me by a random walk to a gate of my choice. If we repeat this enough times then I'll be convinced that there is a labyrinth-path between the two gates, while you will not reveal to me such a path." "Indeed," wise Circe thought to herself, "showing this mortal a random path from a random location in the labyrinth to the gate he chooses will not teach him more than his taking a random walk from that gate."

The foregoing story illustrates the main idea underlying the zero-knowledge proof for Graph Isomorphism presented next. Recall that the set of pairs of isomorphic graphs is not known to be in $\mathcal{BPP}$, and thus the straightforward NP-proof system (in which the prover just supplies the isomorphism) may not be zero-knowledge. Furthermore, assuming that Graph Isomorphism is not in $\mathcal{BPP}$, this set has no zero-knowledge NP-proof system, but as we shall shortly see it does have a zero-knowledge interactive proof system.

Construction 9.9 (zero-knowledge proof for Graph Isomorphism):

- Common Input: *A pair of graphs, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Let ϕ be an isomorphism between the input graphs; namely, ϕ is a 1-1 and onto mapping of the vertex set V_1 to the vertex set V_2 such that $\{u, v\} \in E_1$ if and only if $\{\phi(v), \phi(u)\} \in E_2$.*

- Prover's first Step (P1): *The prover selects a random isomorphic copy of G_2, and sends it to the verifier. Namely, the prover selects at random, with uniform probability distribution, a permutation π from the set of permutations over the vertex set V_2, and constructs a graph with vertex set V_2 and edge set*

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

  The prover sends (V_2, E) to the verifier.

- Motivating Remark: *If the input graphs are isomorphic, as the prover claims, then the graph sent in Step P1 is isomorphic to both input graphs. However, if the input graphs are* not *isomorphic then no graph can be isomorphic to both of them.*

- Verifier's first Step (V1): *Upon receiving a graph, $G' = (V', E')$, from the prover, the verifier asks the prover to show an isomorphism between G' and one of the input graphs, chosen at random by the verifier. Namely, the verifier uniformly selects $\sigma \in \{1, 2\}$, and sends it to the prover (who is supposed to answer with an isomorphism between G_σ and G').*

- Prover's second Step (P2): *If the message, σ, received from the verifier equals 2 then the prover sends π to the verifier. Otherwise (i.e., $\sigma \neq 2$), the prover sends $\pi \circ \phi$ (i.e., the composition of π on ϕ, defined as $\pi \circ \phi(v) \stackrel{\text{def}}{=} \pi(\phi(v))$) to the verifier.*

  (Indeed, the prover treats any $\sigma \neq 2$ as $\sigma = 1$. In the analysis we shall assume, without loss of generality, that $\sigma \in \{1, 2\}$ always holds.)

- Verifier's second Step (V2): *If the message, denoted ψ, received from the prover is an isomorphism between G_σ and G' then the verifier outputs 1, otherwise it outputs 0.*

The verifier strategy in Construction 9.9 is easily implemented in probabilistic polynomial-time. In case the prover is given an isomorphism between the input graphs as auxiliary input, also the prover's program can be implemented in probabilistic polynomial-time. The motivating remark justifies the claim that Construction 9.9 constitutes an interactive proof system for the set of pairs of isomorphic graphs. As for the zero-knowledge property, consider first the special case in which the verifier actually follows the prescribed strategy (and selects σ at random, and in particular obliviously of the graph G' it receives). The view of this verifier can be easily simulated by selecting σ and ψ at random, constructing G' as a random isomorphic copy of G_σ (via the isomorphism ψ), and outputting the triplet (G', σ, ψ). Indeed (even in this case), the simulator behaves differently from the prescribed prover (which selects G' as a random isomorphic copy of G_2, via the isomorphism π), but its output distribution is identical to the verifier's view in the real interaction. However, the forgoing description assumes that the verifier follows the prescribed strategy, while in general the verifier may (adversarially) select σ depending on the graph G'. Thus, a slightly more complicated simulation (described next) is required.

A general clarification may be in place. Recall that we wish to simulate the interaction of an arbitrary verifier strategy with the prescribed prover. Thus, this simulator must depend on the corresponding verifier strategy, and indeed we shall describe the simulator while referring to such a generic verifier strategy. Formally, this means that the simulator's program incorporates the program of the corresponding verifier strategy. (Actually, the following simulator uses the generic verifier strategy as a subroutine.)

Turning back to the specific protocol of Construction 9.9, the basic idea is that simulator tries to guess σ and can complete a simulation if its guess turns out to be correct. Specifically, the simulator selects $\tau \in \{1, 2\}$ uniformly (hoping that the verifier will later select $\sigma = \tau$), and constructs G' by randomly permuting G_τ (and thus being able to present an isomorphism between G_τ and G'). Recall that the simulator is analyzed only on yes-instances (i.e., the input graphs G_1 and G_2 are isomorphic). The point is that if G_1 and G_2 are isomorphic, then the graph G' does not yield any information regarding the simulator's guess (i.e., τ).[16] Thus,

[16]Indeed, this observation is identical to the one made in the analysis of the soundness of Construction 9.3.

the value σ selected by the adversarial verifier may depend on G' but not on σ, which implies that $\Pr[\sigma = \tau] = 1/2$. In other words, the simulator's guess (i.e., τ) is correct (i.e., equals σ) with probability $1/2$. Now, if the guess is correct then the simulator can produce an output that has the correct distribution, and otherwise the entire process is repeated.

Useful conventions. We wish to highlight three conventions that were either used (implicitly) in the foregoing analysis or can be used to simplify the description of (this and/or) other zero-knowledge simulators.

1. Without loss of generality, we may assume that the cheating verifier strategy is implemented by a *deterministic* polynomial-size circuit (or, equivalently, by a deterministic polynomial-time algorithm with an auxiliary input).[17]

   This is justified by fixing any outcome of the verifier's coins, and observing that our (uniform) simulation of the various (residual) deterministic strategies yields a simulation of the original probabilistic strategy.

2. Without loss of generality, it suffices to consider cheating verifiers that (only) output their view of the interaction (i.e., the common input, their internal coin tosses, and the messages that they have received). In other words, it suffices to simulate the view that cheating verifiers have of the real interaction.

   This is justified by noting that the final output of any verifier can be obtained from its view of the interaction, where the complexity of the transformation is upper-bounded by the complexity of the verifier's strategy.

3. Without loss of generality, it suffices to construct a "weak simulator" that produces output with some noticeable[18] probability such that whenever an output is produced it is distributed "correctly" (i.e., similarly to the distribution occuring in real interactions with the prescribed prover).

   This is justified by repeatedly invoking such a weak simulator (polynomially) many times and using the first output produced by any of these invocations. Note that by using an adequate number of invocations, we fail to produce an output with negligible probability. Furthermore, note that a simulator that fails to produce output with negligible probability can be converted to a simulator that always produces an output, while incurring a negligible statistic deviation in the output distribution.

9.2.2.2 The full power of zero-knowledge proofs

The zero-knowledge proof system presented in Construction 9.9 refers to one specific NP-set that is not known to be in $\mathcal{BPP}$. It turns out that, under reasonable

[17] This observation is not crucial, but it does simplify the analysis (by eliminating the need to specify a sequence of coin tosses in each invocation of the verifier's strategy).

[18] Recall that a probability is called noticeable if it is greater than the reciprocal of some positive polynomial (in the relevant parameter).

assumptions, zero-knowledge can be used to prove membership in *any* NP-set. Intuitively, it suffices to establish this fact for a single NP-complete set, and thus we focus on presenting a zero-knowledge proof system for the set of 3-colorable graphs.

It is easy to prove that a given graph G is 3-colorable by just presenting a 3-coloring of G (and the same holds for membership in any set in $\mathcal{NP}$), but this NP-proof is not a zero-knowledge proof (unless $\mathcal{NP} \subseteq \mathcal{BPP}$). In fact, assuming $\mathcal{NP} \not\subseteq \mathcal{BPP}$, graph 3-colorability has no zero-knowledge NP-proof system, but as we shall shortly see it does have a zero-knowledge interactive proof system. This interactive proof system will be described while referring to "boxes" in which information can be hidden and later revealed. Such boxes can be implemented using one-way functions (see, e.g., Theorem 9.11).

Construction 9.10 (Zero-knowledge proof of 3-colorability, abstract description): *The description refers to abstract non-transparent boxes that can be perfectly locked and unlocked such that these boxes perfectly hide their contents while being locked.*

- Common Input: *A simple graph* $G=(V,E)$.
- Prover's first step: *Let* ψ *be a 3-coloring of* G. *The prover selects a random permutation,* π, *over* $\{1,2,3\}$, *and sets* $\phi(v) \stackrel{\rm def}{=} \pi(\psi(v))$, *for each* $v \in V$. *Hence, the prover forms a random relabeling of the 3-coloring* ψ. *The prover sends to the verifier a sequence of* $|V|$ *locked and non-transparent boxes such that the* $v^{\rm th}$ *box contains the value* $\phi(v)$.
- Verifier's first step: *The verifier uniformly selects an edge* $\{u,v\} \in E$, *and sends it to the prover.*
- Motivating Remark: *The boxes are supposed to contain a 3-coloring of the graph, and the verifier asks to inspect the colors of vertices* u *and* v. *Indeed, for the zero-knowledge condition, it is crucial that the prover only responds to pairs that correspond to edges of the graph.*
- Prover's second step: *Upon receiving an edge* $\{u,v\} \in E$, *the prover sends to the verifier the keys to boxes* u *and* v.

  For simplicity of the analysis, if the verifier sends $\{u,v\} \notin E$ *then the prover behaves as if it has received a fixed* (or random) *edge in* E, *rather than suspending the interaction, which would have been the natural thing to do.*
- Verifier's second step: *The verifier unlocks and opens boxes* u *and* v, *and accepts if and only if they contain two different elements in* $\{1,2,3\}$.

The verifier strategy in Construction 9.10 is easily implemented in probabilistic polynomial-time. The same holds with respect to the prover's strategy, provided that it is given a 3-coloring of G as auxiliary input. Clearly, if the input graph is 3-colorable then the verifier accepts with probability 1 when interacting with the prescribed prover. On the other hand, if the input graph is not 3-colorable, then any contents put in the boxes must be invalid with respect to at least one edge, and consequently the verifier will reject with probability at least $\frac{1}{|E|}$. Hence, the foregoing protocol exhibits a non-negligible gap in the accepting probabilities between the case of 3-colorable graphs and the case of non-3-colorable graphs. To increase the gap, the protocol may be repeated sufficiently many times (of course, using independent coin tosses in each repetition).

In the abstract setting of Construction 9.10, the zero-knowledge property follows easily, because one can simulate the real interaction by placing a random pair of different colors in the boxes indicated by the verifier. This indeed demonstrates that the verifier learns nothing from the interaction, because it expects to see a random pair of different colors (and indeed this is what it sees). Note that the aforementioned expectation relies on the fact that the boxes correspond to vertices that are connected by an edge.

This simple demonstration of the zero-knowledge property is not possible in the digital implementation (discussed next), because in that case the boxes are not totally unaffected by their contents (but are rather effected, yet in an indistinguishable manner). Instead, we simulate the interaction as follows. We first guess (at random) which pair of boxes (corresponding to an edge) the verifier would ask to open, and place a random pair of distinct colors in these boxes (and garbage in the rest).[19] Then, we hand all boxes to the verifier, which asks us to open a pair of boxes (corresponding to an edge). If the verifier asks for the pair that we chose (i.e., our guess is successful), then we can complete the simulation by opening these boxes. Otherwise, we try again (with a new random guess and random colors). Thus, it suffices to use boxes that hide their contents almost perfectly (rather than being perfectly opaque). Such boxes can be implemented digitally.

Teaching note: Indeed, we recommend presenting and analyzing in class only the foregoing abstract protocol. It suffices to briefly comment about the digital implementation, rather than presenting a formal proof of Theorem 9.11 (which can be found in [96] (or [87, Sec. 4.4])).

Digital implementation. We implement the abstract boxes (referred to in Construction 9.10) by using adequately defined commitment schemes. Loosely speaking, such a scheme is a two-phase game between a sender and a receiver such that after the first phase the sender is "committed" to a value and yet, at this stage, it is infeasible for the receiver to find out the committed value (i.e., the commitment is "hiding"). The committed value will be revealed to the receiver in the second phase and it is guaranteed that the sender cannot reveal a value other than the one committed (i.e., the commitment is "binding"). Such commitment schemes can be implemented assuming the existence of one-way functions (as in Definition 7.3).

Zero-knowledge proofs for other NP-sets. Using the fact that 3-colorability is NP-complete, one can derive (from Construction 9.10) zero-knowledge proof sys-

[19] An alternative (and more efficient) simulation consists of putting random independent colors in the various boxes, hoping that the verifier asks for an edge that is properly colored. The latter event occurs with probability (approximately) 2/3, provided that the boxes hide their contents (almost) perfectly.

tems for any NP-set.[20] Furthermore, NP-witnesses can be efficiently transformed into polynomial-size circuits that implement the corresponding (prescribed zero-knowledge) prover strategies.

Theorem 9.11 (The ZK Theorem): *Assuming the existence of* (non-uniformly hard) *one-way functions, any NP-proof can be efficiently transformed into a* (computational) *zero-knowledge interactive proof. In particular,* $\mathcal{NP} \subseteq \mathcal{ZK}$.

The hypothesis of Theorem 9.11 (i.e., the existence of one-way functions) seems unavoidable, because the existence of zero-knowledge proofs for "hard on the average" problems implies the existence of one-way functions (and, likewise, the existence of zero-knowledge proofs for sets outside $\mathcal{BPP}$ implies the existence of "auxiliary-input one-way functions").

Theorem 9.11 has a dramatic effect on the design of cryptographic protocols (see Appendix C). In a different vein we mention that, under the same assumption, any interactive proof can be transformed into a zero-knowledge one. (This transformation, however, is not efficient.)

Theorem 9.12 (The ultimate ZK Theorem): *Assuming the existence of* (non-uniformly hard) *one-way functions,* $\mathcal{IP} = \mathcal{ZK}$.

Loosely speaking, Theorem 9.12 can be proved by recalling that $\mathcal{IP} = \mathcal{AM}(\mathtt{poly})$ and modifying any public-coin protocol as follows: the modified prover sends commitments to its messages rather than the messages themselves, and once the original interaction is completed it proves (in zero-knowledge) that the corresponding transcript would have been accepted by the original verifier. Indeed, the latter assertion is of the "NP type", and thus the zero-knowledge proof system guaranteed in Theorem 9.11 can be invoked for proving it.

Reflection. The proof of Theorem 9.11 uses the fact that 3-colorability is NP-complete in order to obtain a zero-knowledge proofs for any set in $\mathcal{NP}$ by using such a protocol for 3-colorability (i.e., Construction 9.10). Thus, an NP-completeness result is used here in a "positive" way; that is, in order to construct something rather than in order to derive a hardness result. This was probably the first positive application of NP-completeness. Subsequent positive uses of completeness results have appeared in the context of interactive proofs (see the proof of Theorem 9.4), probabilistically checkable proofs (see the proof of Theorem 9.16), and the "hardness versus randomness paradigm" (see, e.g., [122]).

Perfect and Statistical Zero-Knowledge. The foregoing results may be contrasted with the results regarding the complexity of statistical zero-knowledge proof systems: Statistical zero-knowledge proof systems exist only for sets in $\mathcal{IP}(2) \cap \mathrm{co}\mathcal{IP}(2)$, and thus are unlikely to exist for all NP-sets. On the other

[20] Actually, we should either rely on the fact that the standard Karp-reductions are invertible in polynomial time or on the fact that the 3-colorability protocol is actually zero-knowledge with respect to auxiliary inputs (as in Definition C.9).

hand, the class Statistical Zero-Knowledge is known to contain some hard problems, and turns out to have interesting complexity theoretic properties (e.g., being closed under complementation, and having very natural complete problems). The interested reader is referred to [213].

9.2.3 Proofs of Knowledge – a parenthetical subsection

Teaching note: Technically speaking, this topic belongs to Section 9.1, but its more interesting demonstrations refer to zero-knowledge proofs of knowledge – hence its current positioning.

Loosely speaking, "proofs of knowledge" are interactive proofs in which the prover asserts "knowledge" of some object (e.g., a 3-coloring of a graph), and not merely its existence (e.g., the existence of a 3-coloring of the graph, which in turn is equivalent to the assertion that the graph is 3-colorable).

What do we mean by saying that a *machine* knows something? Any standard dictionary suggests several meanings for the verb `to know`, but these are typically phrased with reference to the notion of *awareness*, a notion which is certainly inapplicable in the context of machines. Instead, we should look for a *behavioristic* interpretation of the verb `to know`. Indeed, it is reasonable to link knowledge with the ability to do something (e.g., the ability to write down whatever one knows). Hence, we will say that a machine knows a string α if it *can* output the string α. But this seems as total non-sense too: a machine has a well defined output – either the output equals α or it does not. So what can be meant by saying that *a machine can do something?* Loosely speaking, it may mean that the machine can be *easily modified* so that it does whatever is claimed. More precisely, it may mean that there exists an *efficient* machine that, using the original machine as a black-box (or given its code as an input), outputs whatever is claimed.

Technically speaking, using a machine as a black-box seems more appealing when the said machine is interactive (i.e., implements an interactive strategy). Indeed, this will be our focus here. Furthermore, conceptually speaking, whatever a machine knows (or does not know) is its own business, whereas what can be of interest and reference *to the outside* is whatever can be deduced about the knowledge of a machine by interacting with it. Hence, we are interested in proofs of knowledge (rather than in mere knowledge).

For sake of simplicity let us consider a concrete question: *how can a machine prove that it knows a 3-coloring of a graph?* An obvious way is just sending the 3-coloring to the verifier. Yet, we claim that applying the protocol in Construction 9.10 (i.e., the zero-knowledge proof system for 3-Colorability) is an alternative way of proving knowledge of a 3-coloring of the graph.

The definition of a *verifier of knowledge of 3-coloring* refers to any possible prover strategy and links the ability to "extract" a 3-coloring (of a given graph) from such a prover to the probability that this prover convinces the verifier. That is, the definition postulates the existence of an efficient universal way of "extracting" a 3-coloring of a given graph by using any prover strategy that convinces this verifier to accept this graph with probability 1 (or, more generally, with some noticeable probability). On the other hand, we should no expect this extractor to obtain much from prover strategies that fail to convince the verifier (or, more generally, convince it with negligible probability). A robust definition should allow a smooth transition between these two extremes (and in particular between provers that convince the verifier with noticeable probability and those that convince it with negligible probability). Such a definition should also support the intuition by which the following strategy of Alice is zero-knowledge: *Alice sends Bob a 3-coloring of a given graph provided that Bob has successfully convinced her that he knows this coloring.*[21] We stress that the zero-knowledge property of Alice's strategy should hold regardless of the proof-of-knowledge system used for proving Bob's knowledge of a 3-coloring.

Loosely speaking, we say that an interactive machine, V, constitutes a verifier for knowledge of 3-coloring if, for any prover strategy P, the complexity of extracting a 3-coloring of G when using machine P as a "black box"[22] is inversely proportional to the probability that V is convinced by P (to accept the graph G). Namely, the extraction of the 3-coloring is done by an oracle machine, called an extractor, that is given access to a function specifying the behavior P (i.e., the messages it sends in response to particular messages it may receive). We require that the (*expected*) *running time of the extractor, on input G and access to an oracle specifying P's behavior, be inversely related* (by a factor polynomial in $|G|$) *to the probability that P convinces V to accept G*. In particular, if P always convinces V to accept G, then the extractor runs in expected polynomial-time. The same holds in case P convinces V to accept with noticeable probability. On the other hand, if P never convinces V to accept, then nothing is required of the extractor. We stress that the latter special cases do not suffice for a satisfactory definition; see discussion in [87, Sec. 4.7.1].

Proofs of knowledge, and in particular zero-knowledge proofs of knowledge, have many applications to the design of cryptographic schemes and cryptographic protocols. These are enabled by the following general result.

Theorem 9.13 (Theorem 9.11, revisited): *Assuming the existence of* (non-uniformly hard) *one-way functions, any NP-relation has a zero-knowledge proof of knowledge* (of a corresponding NP-witnesses)*. Furthermore, the prescribed prover strategy can be implemented in probabilistic polynomial-time, provided it is given such an NP-witness.*

[21] For simplicity, the reader may consider graphs that have a unique 3-coloring (upto a relabeling). In general, we refer here to instances that have unique solution (cf. Section 6.2.3), which arise naturally in some (cryptographic) applications.

[22] Indeed, one may consider also non-black-box extractors.

9.3 Probabilistically Checkable Proof Systems

> **Teaching note:** Probabilistically checkable proof (PCP) systems may be viewed as a restricted type of interactive proof systems in which the prover is memoryless and responds to each verifier message as if it were the first such message. This perspective creates a tighter link with previous sections, but is somewhat contrived. However, such a memoryless prover may be viewed as a static object that the verifier may query at locations of its choice. But then it is more appealing to present the model using the (more traditional) terminology of oracle machines rather than using (and degenerating) the terminology of interactive machines.

Probabilistically checkable proof systems can be viewed as standard (deterministic) proof systems that are augmented with a probabilistic procedure capable of evaluating the validity of the assertion by examining few locations in the alleged proof. In fact, we focus on the latter probabilistic procedure, which is given direct access to the individual bits of the alleged proof (and need not scan it bit-by-bit). Thus, the alleged proof is a string, as in the case of a traditional proof system, but we are interested in probabilistic verification procedures that access only few locations in the proof, and yet are able to make a meaningful probabilistic verdict regarding the validity of the alleged proof. Specifically, the verification procedure should accept any valid proof (with probability 1), but rejects with probability at least 1/2 any alleged proof for a false assertion.

The main complexity measure associated with probabilistically checkable proof systems is indeed their query complexity. Another complexity measure of natural concern is the length of the proofs being employed, which in turn is related to the randomness complexity of the system. The randomness complexity of PCPs plays a key role in numerous applications (e.g., in composing PCP systems as well as when applying PCP systems to derive inapproximability results), and thus we specify this parameter rather than the proof length.

> **Teaching note:** Indeed, PCP systems are most famous for their role in deriving numerous inapproximability results (see Section 9.3.3), but our view is that the latter is merely one extremely important application of the fundamental notion of a PCP system. Our presentation is organized accordingly.

9.3.1 Definition

Loosely speaking, a probabilistically checkable proof system consists of a probabilistic polynomial-time verifier having access to an oracle that represents an alleged proof (in redundant form). Typically, the verifier accesses only few of the oracle bits, and these bit positions are determined by the outcome of the verifier's coin tosses. As in the case of interactive proof systems, it is required that if the assertion holds then the verifier always accepts (i.e., when given access to an adequate oracle); whereas, if the assertion is false then the verifier must reject with probability at least $\frac{1}{2}$, no matter which oracle is used. The basic definition of the PCP setting is given in Part (1) of the following definition. Yet, the complexity measures introduced in Part (2) are of key importance for the subsequent discussions.

Definition 9.14 (Probabilistically Checkable Proofs – PCP):

1. *A* probabilistically checkable proof system (PCP) *for a set S is a probabilistic polynomial-time oracle machine, called* verifier *and denoted V, that satisfies the following two conditions:*
   - Completeness: *For every $x \in S$ there exists an oracle π_x such that, on input x and access to oracle π_x, machine V always accepts x.*
   - Soundness: *For every $x \notin S$ and every oracle π, on input x and access to oracle π, machine V rejects x with probability at least $\frac{1}{2}$.*
2. *We say that a probabilistically checkable proof system has* query complexity *$q:\mathbb{N}\to\mathbb{N}$ if, on any input of length n, the verifier makes at most $q(n)$ oracle queries.*[23] *Similarly, the* randomness complexity *$r:\mathbb{N}\to\mathbb{N}$ upper-bounds the number of coin tosses performed by the verifier on a generic n-bit long input.*

   For integer functions r and q, we denote by $\mathcal{PCP}(r,q)$ the class of sets having probabilistically checkable proof systems of randomness complexity r and query complexity q. For sets of integer functions, R and Q,

$$\mathcal{PCP}(R,Q) \stackrel{\text{def}}{=} \bigcup_{r\in R\,,\,q\in Q} \mathcal{PCP}(r,q)\,.$$

We note that the oracle π_x referred to in the completeness condition a PCP system constitutes a proof in the standard mathematical sense (with respect to a verification procedure that examines all possible outcomes of V's internal coin tosses). Furthermore, the oracles in PCP systems of logarithmic randomness complexity constitute NP-proofs. However, these oracles have the extra remarkable property of enabling a lazy verifier to toss coins, take its chances and "assess" the validity of the proof without reading all of it (but rather by reading a tiny portion of it). Potentially, this allows the verifier to utilize very long proofs (i.e., of super-polynomial length) or alternatively examine very few bits of an NP-proof.

We note that the error probability (in the soundness condition) of PCP systems can be reduced by successive applications of the proof system. In particular, repeating the process for k times, reduces the probability that the verifier is fooled by a false assertion to 2^{-k}, whereas all complexities increase by at most a factor of k. Thus, PCP systems provide a trade-off between the number of locations examined in the proof and the confidence in the validity of the assertion.

Adaptive versus non-adaptive verifiers. Definition 9.14 allows the verifier to be adaptive; that is, the verifier may determine its queries based on the answers it has received to previous queries (in addition to their dependence on the input and the verifier's internal coin tosses). In contrast, non-adaptive verifiers determine all their queries based solely on their input and internal coin tosses. We comment that most constructions of PCP systems use non-adaptive verifiers, and in fact in many sources PCP systems are defined as non-adaptive.

[23] As usual in complexity theory, the oracle answers are always binary (i.e., either 0 or 1).

Randomness versus proof length. Note that the "effective" length of proofs for any PCP system is related to its query and randomness complexities, where the effective length means the number of locations in a generic proof-oracle that may be examined on a fixed input and any possible sequence of internal coin tosses. Specifically, if the PCP system has query complexity q and randomness complexity r then its effective proof length is upper-bounded by 2^{q+r}, whereas a bound of $2^r \cdot q$ holds for non-adaptive systems (see Exercise 9.11). On the other hand, in some sense, the randomness complexity of a PCP can be upper-bounded by the logarithm of the length of the proofs employed (provided we allow non-uniform verifiers; see Exercise 9.13).

On the role of randomness. The PCP Theorem (i.e., $\mathcal{NP} = \mathcal{PCP}(\texttt{log}, O(1))$) exhibits a trade-off between the number of bits examined in the alleged proof and the confidence in the validity of the assertion. We note that such a trade-off is impossible if one requires the verifier to be deterministic. This is due to the fact that every set in $\mathcal{PCP}(r,q)$ has an NP-proof system that employs proofs of length $2^r q$ (see Exercise 9.12). Thus, $\mathcal{PCP}(r,q) \subseteq \textsc{Dtime}(2^{2^r q} \cdot \text{poly})$, and, in particular, $\mathcal{PCP}(0,\texttt{log}) = \mathcal{P}$. Furthermore, since it is unlikely that all NP-sets have NP-proof systems that employs proofs of (say) linear length, it follows that for $2^{r(n)}q(n) \leq n$ (or for any other fixed polynomial that bounds $2^r q$) the class $\mathcal{PCP}(r,q)$ is unlikely to contain $\mathcal{NP}$. Actually, $\mathcal{P} \neq \mathcal{NP}$ implies that $\mathcal{NP}$ is not contained in $\mathcal{PCP}(o(\log), o(\log))$ (see Exercise 9.15).

9.3.2 The Power of Probabilistically Checkable Proofs

The celebrated PCP Theorem asserts that $\mathcal{NP} = \mathcal{PCP}(\texttt{log}, O(1))$, and this result is indeed the focus of the current section. But before getting to it we make several simple observations regarding the PCP Hierarchy.

We first note that $\mathcal{PCP}(\texttt{poly}, 0)$ equals co$\mathcal{RP}$, whereas $\mathcal{PCP}(0, \texttt{poly})$ equals $\mathcal{NP}$. It is easy to prove an upper bound on the non-deterministic time complexity of sets in the PCP hierarchy (see Exercise 9.12):

Proposition 9.15 (upper-bounds on the power of PCPs): *For every polynomially bounded integer function r, it holds that $\mathcal{PCP}(r, \texttt{poly}) \subseteq \textsc{Ntime}(2^r \cdot \texttt{poly})$. In particular, $\mathcal{PCP}(\texttt{log}, \texttt{poly}) \subseteq \mathcal{NP}$.*

The focus on PCP systems of logarithmic randomness complexity reflects an interest in PCP systems that utilize proof oracles of polynomial length (see discussion in Section 9.3.1). We stress that such PCP systems (i.e., $\mathcal{PCP}(\texttt{log}, q)$) are NP-proof systems with a (potentially amazing) extra property: the validity of the assertion can be "probabilistically evaluated" by examining a (small) portion (i.e., $q(n)$ bits) of the proof. Thus, for any fixed polynomially bounded function q, a result of the form

$$\mathcal{NP} \subseteq \mathcal{PCP}(\texttt{log}, q) \tag{9.6}$$

is interesting (because it applies also to NP-sets having witnesses of length exceeding q), and the smaller q – the better. The PCP Theorem asserts the amazing fact by which q can be made a constant.

Theorem 9.16 (The PCP Theorem): $\mathcal{NP} \subseteq \mathcal{PCP}(\texttt{log}, O(1))$.

Thus, probabilistically checkable proofs in which the verifier tosses only logarithmically many coins and makes only a constant number of queries exist for every set in $\mathcal{NP}$. Furthermore, the proof of Theorem 9.16 is constructive in the sense that it allows to efficiently transform any NP-witness (for an instance of a set in $\mathcal{NP}$) into an oracle that makes the PCP verifier accept (with probability 1). Thus, NP-proofs can be transformed into NP-proofs that offer a trade-off between the portion of the proof being read and the confidence it offers. Specifically, for every $\varepsilon > 0$, if one is willing to tolerate an error probability of ε then it suffices to examine $O(\log(1/\varepsilon))$ bits of the (transformed) NP-proof. Indeed (as discussed in Section 9.3.1), these bit locations need to be selected at random.

A new characterization of NP: Combining Theorem 9.16 with Proposition 9.15 we obtain the following characterization of $\mathcal{NP}$.

Corollary 9.17 (The PCP characterization of NP): $\mathcal{NP} = \mathcal{PCP}(\texttt{log}, O(1))$.

The proof of the PCP Theorem: Theorem 9.16 is a culmination of a sequence of remarkable works, each establishing meaningful and increasingly stronger versions of Eq. (9.6). A presentation of the full proof of Theorem 9.16 is beyond the scope of the current work (and is, in our opinion, unsuitable for a basic course in complexity theory). Instead, we present an overview of the original proof (see §9.3.2.2) as well as of an alternative proof (see §9.3.2.3) that was found more than a decade later. We will start, however, by presenting a weaker result that is used in both proofs of Theorem 9.16 and is also of independent interest. This weaker result (see §9.3.2.1) asserts that every NP-set has a PCP system with constant query complexity (albeit with polynomial randomness complexity); that is, $\mathcal{NP} \subseteq \mathcal{PCP}(\texttt{poly}, O(1))$.

Teaching note: In our opinion, presenting in class any part of the proof of the PCP Theorem should be given low priority. In particular, presenting the connections between PCP and the complexity of approximation should be given a higher priority. As for relative priorities among the following three subsections, we recommend giving §9.3.2.1 the highest priority, because it offers a direct demonstration of the power of PCPs. As for the two alternative proofs of the PCP Theorem itself, our recommendation depends on the intended goal. On one hand, for the purpose of merely giving a taste of the ideas involved in the proof, we prefer an overview of the original proof (provided in §9.3.2.2). On the other hand, for the purpose of actually providing a full proof, we definitely prefer the new proof (which is only outlined in §9.3.2.3).

9.3.2.1 Proving that $\mathcal{NP} \subseteq \mathcal{PCP}(\texttt{poly}, O(1))$

The fact that every NP-set has a PCP system with constant query complexity (regardless of its randomness complexity) already testifies to the power of PCP systems. It asserts that *probabilistic verification of proofs is possible by inspecting very few locations in a* (potentially huge) *proof.* Indeed, the PCP systems presented next utilize exponentially long proofs, but they do so while inspecting these proofs at a constant number of (randomly selected) locations.

We start with a brief overview of the construction. We first note that it suffices to construct a PCP for proving the satisfiability of a given system of quadratic equations over GF(2), because this problem is NP-complete.[24] For inputs consisting of quadratic equations with n variables, the oracle (of this PCP) is supposed to provide the values of all quadratic expressions in these n variables evaluated at some fixed assignment to these variables. This assignment is supposed to satisfy the system of quadratic equations that is given as input. We distinguish two tables in the oracle: The first table corresponding to the 2^n linear expressions and the second table to the 2^{n^2} quadratic expressions. Each table is tested for self-consistency (via a "linearity test"), and the two tables are tested to be consistent with each other (via a "matrix-equality" test, which utilizes "self-correction"). Each of these tests utilizes a constant number of Boolean queries, and randomness that is logarithmic in the size of the corresponding table (and is thus $O(n^2)$). Finally, we test (again via self-correction) the value assigned by these tables to a quadratic expression obtained by a random linear combination of the quadratic expressions that appear in the quadratic system that is given as input. Details follow.

The starting point. We construct a PCP system for the set of satisfiable quadratic equations over GF(2). The input is a sequence of such equations over the variables $x_1, ..., x_n$, and the proof oracle consist of two parts (or tables), which are supposed to provide information regarding some satisfying assignment $\tau = \tau_1 \cdots \tau_n$ (also viewed as an n-ary vector over GF(2)). The first part, denoted T_1, is supposed to provide a Hadamard encoding of the said satisfying assignment; that is, for every $\alpha \in \mathrm{GF}(2)^n$ this table is supposed to provide the inner product mod 2 of the n-ary vectors α and τ (i.e., $T_1(\alpha)$ is supposed to equal $\sum_{i=1}^n \alpha_i\tau_i$). The second part, denoted T_2, is supposed to provide all linear combinations of the values of the $\tau_i\tau_j$'s; that is, for every $\beta \in \mathrm{GF}(2)^{n^2}$ (viewed as an n-by-n matrix over GF(2)), the value of $T_2(\beta)$ is supposed to equal $\sum_{i,j} \beta_{i,j}\tau_i\tau_j$. (Indeed T_1 is contained in T_2, because $\sigma^2 = \sigma$ for any $\sigma \in \mathrm{GF}(2)$.) The PCP verifier will use the two tables for checking that the input (i.e., a sequence of quadratic equations) is satisfied by the assignment that is encoded in the two tables. Needless to say, these tables may not be a valid encoding of any n-ary vector (let alone one that satisfies the input), and so the verifier also needs to check that the encoding is (close to being) valid. We will focus on this task first.

[24] Here and elsewhere, we denote by GF(2) the 2-element field.

Testing the Hadamard Code. Note that T_1 is supposed to encode a linear function; that is, there must be some $\tau = \tau_1 \cdots \tau_n \in \mathrm{GF}(2)^n$ such that $T_1(\alpha) = \sum_{i=1}^n \tau_i \alpha_i$ holds for every $\alpha = \alpha_1 \cdots \alpha_n \in \mathrm{GF}(2)^n$. This can be tested by selecting uniformly $\alpha', \alpha'' \in \mathrm{GF}(2)^n$ and checking whether $T_1(\alpha') + T_1(\alpha'') = T_1(\alpha' + \alpha'')$, where $\alpha' + \alpha''$ denotes addition of vectors over GF(2). The analysis of this natural tester turns out to be quite complex. Nevertheless, it is indeed the case that any table that is 0.01-far from being linear is rejected with probability at least 0.02 (see Exercise 9.16), where T is ε-far from being linear if T disagrees with any linear function f on more than an ε fraction of the domain (i.e., $\mathsf{Pr}_r[T(r) = f(r)] > \varepsilon$).

By repeating the linearity test for a constant number of times, we may reject each table that is 0.01-far from being a codeword of the Hadamard Code with probability at least 0.99. Thus, using a constant number of queries, the verifier rejects any T_1 that is 0.01-far from being a Hadamard encoding of any $\tau \in \mathrm{GF}(2)^n$, and likewise rejects any T_2 that is 0.01-far from being a Hadamard encoding of any $\tau' \in \mathrm{GF}(2)^{n^2}$. We may thus assume that T_1 (resp., T_2) is 0.01-close to the Hadamard encoding of some τ (resp., τ'). (This does not mean, however, that τ' equals the outer produce of τ with itself.)

In the rest of the analysis, we fix $\tau \in \mathrm{GF}(2)^n$ and $\tau' \in \mathrm{GF}(2)^{n^2}$, and denote the Hadamard encoding of τ (resp., τ') by $f_\tau : \mathrm{GF}(2)^n \to \mathrm{GF}(2)$ (resp., $f_{\tau'} : \mathrm{GF}(2)^{n^2} \to \mathrm{GF}(2)$). Recall that T_1 (resp., T_2) is 0.01-close to f_τ (resp., $f_{\tau'}$).

Self-correction of the Hadamard Code. Suppose that T is ε-close to a linear function $f : \mathrm{GF}(2)^n \to \mathrm{GF}(2)$ (i.e., $\mathsf{Pr}_r[T(r) = f(r)] \leq \varepsilon$). Then, we can recover the value of f at any desired point x, by making two (random) queries to T. Specifically, for a uniformly selected $r \in \mathrm{GF}(2)^n$, we use the value $T(x+r) - T(r)$. Note that the probability that we recover the correct value is at least $1-2\varepsilon$, because $\mathsf{Pr}_r[T(x+r) - T(r) = f(x+r) - f(r)] \geq 1 - 2\varepsilon$ and $f(x+r) - f(r) = f(x)$ by linearity of f. (Needless to say, for $\varepsilon < 1/4$, the function T cannot be ε-close to two different linear functions.)[25] Thus, assuming that T_1 is 0.01-close to f_τ (resp., T_2 is 0.01-close to $f_{\tau'}$) we may correctly recover (i.e., with error probability 0.02) the value of f_τ (resp., $f_{\tau'}$) at any desired point by making 2 queries to T_1 (resp., T_2).

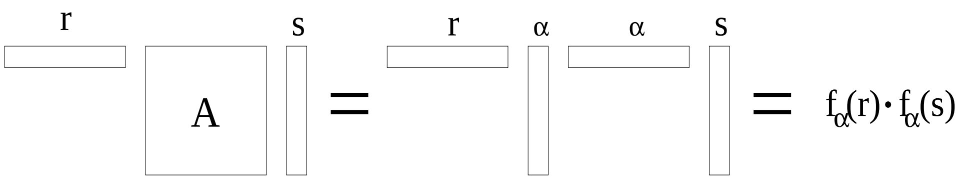


Figure 9.2: Detail for testing consistency of linear and quadratic forms.

[25]Indeed, this fact follows from the self-correction argument, but a simpler proof merely refers to the fact that the Hadamard code has relative distance 1/2.

Checking consistency of f_τ and $f_{\tau'}$. Suppose that we are given access to $f_\tau : \mathrm{GF}(2)^n \to \mathrm{GF}(2)$ and $f_{\tau'} : \mathrm{GF}(2)^{n^2} \to \mathrm{GF}(2)$, where $f_\tau(\alpha) = \sum_i \tau_i\alpha_i$ and $f_{\tau'}(\alpha') = \sum_{i,j} \tau'_{i,j}\alpha'_{i,j}$, and that we wish to verify that $\tau'_{i,j} = \tau_i\tau_j$ for every $i,j \in \{1,...,n\}$. In other words, we are given a (somewhat weird) encoding of two matrices, $A = (\tau_i\tau_j)_{i,j}$ and $A' = (\tau'_{i,j})_{i,j}$, and we wish to check whether or not these matrices are identical. It can be shown (see Exercise 9.18) that if $A \neq A'$ then $\mathsf{Pr}_{r,s}[r^\top As \neq r^\top A's] \geq 1/4$, where r and s are uniformly distributed n-ary vectors. Note that, in our case (where $A = (\tau_i\tau_j)_{i,j}$ and $A' = (\tau'_{i,j})_{i,j}$), it holds that $r^\top As = \sum_j(\sum_i r_i\tau_i\tau_j)s_j = f_\tau(r)f_\tau(s)$ (see Figure 9.2) and $r^\top A's = \sum_j(\sum_i r_i\tau'_{i,j})s_j = f_{\tau'}(rs^\top)$, where $rs^\top$ is the outer-product of s and r. Thus, (for $(\tau_i\tau_j)_{i,j} \neq (\tau'_{i,j})_{i,j}$) we have $\mathsf{Pr}_{r,s}[f_\tau(r)f_\tau(s) \neq f_{\tau'}(rs^\top)] \geq 1/4$. Using self-correction (to obtain the desired value of $f_{\tau'}$ at $rs^\top$, since $rs^\top$ is not uniformly distributed in $\mathrm{GF}(2)^{n^2}$), we test the consistency of f_τ and $f_{\tau'}$; that is, we select uniformly $r,s \in \mathrm{GF}(2)^n$ and $R \in \mathrm{GF}(2)^{n^2}$ and check that $T_1(r)T_1(s) = T_2(rs^\top + R) - T_2(R)$.

By repeating the consistency test for a constant number of times, we may reject an inconsistent pair of tables with probability at least 0.99. Thus, in the rest of the analysis, we may assume that $(\tau_i\tau_j)_{i,j} = (\tau'_{i,j})_{i,j}$.

Checking that τ satisfies the quadratic system. Suppose that we are given access to f_τ and $f_{\tau'}$ as in the foregoing (where, in particular, $\tau' = \tau\tau^\top$). A key observation is that if τ does not satisfy a system of quadratic equations then, with probability 1/2, it does not satisfy a random linear combination of these equations. Thus, in order to check whether τ satisfies the quadratic system, we create a single quadratic equation (by taking such a random linear combination) and compare the value of the resulting quadratic expression to the corresponding value, by recovering the value of $f_{\tau'}$ at a single point (which corresponds to the quadratic equation). That is, to test whether τ satisfies the quadratic equation $Q(x) = \sigma$, we test whether $f_{\tau'}(Q) = \sigma$. The actual checking is implemented by the verifier using self-correction (of the table T_2).

To summarize, the verifier performs a constant number of queries and uses randomness that is quadratic in the number of variables. If the quadratic system is satisfiable (by some τ), then the verifier accepts the corresponding tables T_1 and T_2 (i.e., $T_1 = f_\tau$ and $T_2 = f_{\tau\tau^\top}$) with probability 1. On the other hand, if the quadratic system is unsatisfiable, then any pair of tables (T_1, T_2) will be rejected with constant probability (by one of the foregoing tests). It follows that $\mathcal{NP} \subseteq \mathcal{PCP}(r, O(1))$, where $r(n) = O(n^2)$.

9.3.2.2 Overview of the first proof of the PCP Theorem

The original proof of the PCP Theorem (Theorem 9.16) consists of three main conceptual steps, which we briefly sketch first and further discuss later.

1. Constructing a (non-adaptive) PCP system for $\mathcal{NP}$ having *logarithmic randomness and polylogarithmic query complexity*. Furthermore, this proof system has additional properties that enable proof composition as in the following Step (3).

2. Constructing a PCP system for $\mathcal{NP}$ having *polynomial randomness and constant query complexity* (indeed, as in §9.3.2.1). This proof system too has additional properties enabling proof composition as in Step (3).

3. The proof composition paradigm:[26] In general, this paradigm allows to compose two proof systems such that the "inner" one is used for probabilistically verifying the acceptance criteria of the "outer" verifier. The aim is to conduct this ("composed") verification using much fewer queries than the query complexity of the "outer" proof system. In particular, the inner verifier cannot afford to read its input, which makes composition more subtle than the term suggests.

   Loosely speaking, the *outer* verifier should be robust in the sense that its soundness condition guarantee that with high probability the oracle answers are "far" from satisfying the residual decision predicate (rather than merely not satisfy it). (Furthermore, the latter predicate, which is well-defined by the non-adaptive nature of the outer verifier, must have a circuit of size bounded by a polynomial in the number of queries.) The *inner* verifier is given oracle access to its input and is charged for each query made to it, but is only required to reject with high probability inputs that are far from being valid (and, as usual, accept inputs that are valid). That is, the inner verifier is actually a verifier of proximity.

   Composing two such PCPs yields a new PCP for $\mathcal{NP}$, where the new proof oracle consists of the proof oracle of the "outer" system and a sequence of proof oracles for the "inner" system (one "inner" proof per each possible random-tape of the "outer" verifier). Thus, composing an outer verifier of randomness complexity r' and query complexity q' with an inner verifier of randomness complexity r'' and query complexity q'' yields a PCP of randomness complexity $r(n) = r'(n) + r''(q'(n))$ and query complexity $q(n) = q''(q'(n))$, because $q'(n)$ represents the length of the input (oracle) that is accessed by the inner verifier. Recall that the outer verifier is non-adaptive, and thus if the inner verifier is non-adaptive (resp., robust) then so is the verifier resulting from the composition, which is important in case we wish to compose the latter verifier with another inner verifier.

In particular, the proof system of Step (1) is composed with itself [using $r'(n) = r''(n) = O(\log n)$ and $q'(n) = q''(n) = \text{poly}(\log n)$] yielding a PCP system (for $\mathcal{NP}$) of randomness complexity $r(n) = r'(n) + r''(q'(n)) = O(\log n)$ and query complexity $q(n) = q''(q'(n)) = \text{poly}(\log \log n)$. Composing the latter system (used as an "outer" system) with the the PCP system of Step (2), yields a PCP system (for $\mathcal{NP}$) of randomness complexity $r(n) + \text{poly}(q(n)) = O(\log n)$ and query complexity $O(1)$, thus establishing the PCP Theorem.

A more detailed overview – the plan. The foregoing description uses two (non-trivial) PCP systems and refers to additional properties such as robustness

[26]Our presentation of the composition paradigm follows [33], rather than the original presentation of [14, 13].

and verification of proximity. A PCP system of polynomial randomness complexity and constant query complexity (as postulated in Step 2) is outlined in §9.3.2.1. We thus start by discussing the notions of verifying proximity and being robust, while demonstrating their applicability to the said PCP. Finally, we outline the other PCP system that is used (i.e., the one postulated in Step 1).

PCPs of Proximity. Recall that a standard PCP verifier gets an explicit input and is given oracle access to an alleged proof (for membership of the input in a predetermined set). In contrast, a PCP of proximity verifier is given oracle access to two oracles, one representing an input and the other being an alleged proof. Typically, the query complexity of this verifier is lower than the length of the input oracle, and hence this verifier cannot afford reading the entire input and cannot be expected to make absolute statements about it. Indeed, instead of deciding whether or not the input is in a predetermined set, the verifier is only required to distinguish the case that the input is in the set from the case that the input is *far* from the set (where far means being at *relative* Hamming distance at least 0.01 (or any other constant)).

For example, consider a variant of the system of §9.3.2.1 in which the quadratic system is fixed[27] and the verifier needs to determine whether the assignment appearing in the input oracle satisfies the said system or is far from any assignment that satisfies it. The proof oracle is as in §9.3.2.1, and a PCP of proximity may proceed as in §9.3.2.1 and in addition perform a proximity test to verify that the input oracle is close to the assignment encoded in the proof oracle. Specifically, the verifier may read a uniformly selected bit of the input oracle and compare this value to the self-corrected value obtained from the proof oracle (i.e., for a uniformly selected $i \in \{1, ..., n\}$, we compare the i^{th} bit of the input oracle to the self-correction of the value $T_1(0^{i-1}10^{n-i})$, obtained from the proof oracle).

Robust PCPs. Composing an "outer" PCP verifier with an "inner" PCP verifier of proximity makes sense provided that the *outer* verifier rejects in a "robust" manner. That is, the soundness condition of a robust verifier requires that (with probability at least 1/2) the oracle answers are *far* from any sequence that is acceptable by the residual predicate (rather than merely that the answers are rejected by this predicate). Indeed, if the outer verifier is (non-adaptive and) robust, then it suffices that the inner verifier distinguish (with the help of an adequate proof) answers that are valid from answers that are far from being valid.

For example, if robustness is defined as referring to *relative constant distance* (which is indeed the case), then the PCP of §9.3.2.1 (as well as any PCP of constant query complexity) is trivially robust. However, we will not care about the robustness of this PCP, because we only use this PCP as an inner verifier in proof composition. In contrast, we will care about the robustness of PCPs that are used as outer verifiers (e.g., the PCP presented next).

[27] Indeed, in our applications the quadratic system will be "known" to the ("inner") verifier, because it is determined by the ("outer") verifier.

Teaching note: Unfortunately, the construction of a PCP of logarithmic randomness and polylogarithmic query complexity for NP involves many technical details. Furthermore, obtaining a robust version of this PCP is beyond the scope of the current text. Thus, the following description should be viewed as merely providing a flavor of the underlying ideas.

PCP of logarithmic randomness and polylogarithmic query complexity for NP. We start by showing that $NP \subseteq \mathcal{PCP}(f,f)$, for $f(n) = \text{poly}(\log n)$. The proof system is based on an arithmetization of CNF formulae, which is different from the one used in §9.1.2.2 (for constructing an interactive proof system for co$\mathcal{NP}$). In the current arithmetization, the names of the variables (resp., clauses) of the input formula ϕ are represented by binary strings of logarithmic (in $|\phi|$) length, and a *generic* variable (resp., clause) of ϕ is represented by a logarithmic number of *new variables* (which are assigned values in a finite field $\mathrm{F} \supset \{0,1\}$). The (structure of the) input 3CNF formula $\phi(x_1, ..., x_n)$ is represented by a Boolean function $C_\phi : \{0,1\}^{O(\log n)} \to \{0,1\}$ such that $C_\phi(\alpha, \beta_1, \beta_2, \beta_3) = 1$ if and only if, for $i = 1,2,3$, the i^{th} literal in the α^{th} clause has index $\beta_i = (\gamma_i, \sigma_i)$ that is viewed as a variable name augmented by its sign. Thus, for every $\alpha \in \{0,1\}^{\log|\phi|}$ there is a unique $(\beta_1, \beta_2, \beta_3) \in \{0,1\}^{3\log 2n}$ such that $C_\phi(\alpha, \beta_1, \beta_2, \beta_3) = 1$ holds. Next, we consider a multi-linear extension of C_ϕ over F, denoted Φ; that is, Φ is the (unique) multi-linear polynomial that agrees with C_ϕ on $\{0,1\}^{O(\log n)} \subset \mathrm{F}^{O(\log n)}$. Thus, on input ϕ, the verifier first constructs C_ϕ and Φ. Part of the proof oracle of this verifier is viewed as function $A : \mathrm{F}^{\log n} \to \mathrm{F}$, which is supposed to be a multi-linear extension of a truth assignment that satisfies ϕ (i.e., for every $\gamma \in \{0,1\}^{\log n} \equiv [n]$, the value $A(\gamma)$ is supposed to be the value of the γ^{th} variable in such an assignment). Thus, we wish to check whether, for every $\alpha \in \{0,1\}^{\log|\phi|}$, it holds that

$$\sum_{\beta_1\beta_2\beta_3 \in \{0,1\}^{3\log 2n}} \Phi(\alpha, \beta_1, \beta_2, \beta_3) \cdot \prod_{i=1}^{3} (1 - A'(\beta_i)) = 0 \tag{9.7}$$

where $A'(\beta)$ is the value of the β^{th} literal under the (variable) assignment A; that is, for $\beta = (\gamma, \sigma)$, where $\gamma \in \{0,1\}^{\log n}$ is a variable name and $\sigma \in \{0,1\}$ is the literal's type, it holds that $A'(\beta) = \sigma \cdot A(\gamma) + (1-\sigma) \cdot (1 - A(\gamma))$. Thus, Eq. (9.7) holds if and only if the α^{th} clause is satisfied by the assignment induced by A (because $A'(\beta) = 1$ must hold for at least one of the three literals β that appear in this clause).[28] Note that, as in §9.3.2.1, we cannot afford to verify all n instances of Eq. (9.7). Furthermore, unlike in §9.3.2.1, we cannot afford to take a random linear combination of these n instances either (because this requires too much randomness). Fortunately, taking a "pseudorandom" linear combination of these equations is good enough. Specifically, using an adequate (efficiently constructible) small-bias probability space (cf. §8.6.2.3) will do. Denoting such a space (of size $\text{poly}(|\phi| \cdot |F|)$ and bias at most $1/6$) by $S \subset \mathrm{F}^{|\phi|}$, we may select uniformly $(s_1, ..., s_{|\phi|}) \in S$ and

[28] Note that, for this α there exists a unique triple $(\beta_1, \beta_2, \beta_3) \in \{0,1\}^{3\log 2n}$ such that $\Phi(\alpha, \beta_1, \beta_2, \beta_3) \neq 0$. This triple $(\beta_1, \beta_2, \beta_3)$ encodes the literals appearing in the α^{th} clause, and this clause is satisfied by A if and only if $\exists i \in [3]$ s.t. $A'(\beta_i) = 1$.

check whether

$$\sum_{\alpha\beta_1\beta_2\beta_3 \in \{0,1\}^\ell} s_\alpha \cdot \Phi(\alpha, \beta_1, \beta_2, \beta_3) \cdot \prod_{i=1}^{3} (1 - A'(\beta_i)) = 0 \qquad (9.8)$$

where $\ell \stackrel{\rm def}{=} \log|\phi| + 3\log 2n$. The small-bias property guarantees that if A fails to satisfy any of the equations of type Eq. (9.7) then, with probability at least 1/3 (taken over the choice of $(s_1, ..., s_{|\phi|}) \in S$), it is the case that A fails to satisfy Eq. (9.8). Since $|S| = \text{poly}(|\phi| \cdot |F|)$ rather that $|S| = 2^{|\phi|}$, we can select a sample in S using $O(\log|\phi|)$ coin tosses. Thus, we have reduced the original problem to checking whether, for a random $(s_1, ..., s_{|\phi|}) \in S$, Eq. (9.8) holds.

Assuming (for a moment) that A is a low-degree polynomial, we can probabilistically verify Eq. (9.8) by applying a summation test (as in the interactive proof for co$\mathcal{NP}$). Indeed, the verifier obtains the relevant univariate polynomials by making adequate queries (which specify the entire sequence of choices made so far in the summation test). Note that after stripping the ℓ summations, the verifier end-ups up with an expression that contains three unknown values of A', which it may obtain by making corresponding queries to A. The summation test involves tossing $\ell \cdot \log|\mathrm{F}|$ coins and making $(\ell + 3) \cdot O(\log|\mathrm{F}|)$ Boolean queries (which correspond to ℓ queries that are each answered by a univariate polynomial of constant degree (over F), and three queries to A (each answered by an element of F)). Soundness of the summation test follows by setting $|F| \gg O(\ell)$. Needless to say, we must also check that A is indeed a multi-variate polynomial of low degree (or rather that it is close to such a polynomial). A low-degree test of complexities similar to those of the summation text does exist. Using a finite field F of $\text{poly}(\log(n))$ elements, this yields $\mathcal{NP} \subseteq \mathcal{PCP}(f, f)$ for $f(n) \stackrel{\rm def}{=} O(\log(n) \cdot \log\log(n))$.

To obtain the desired PCP system of logarithmic randomness complexity, we represent the names of the original variables and clauses by $\frac{O(\log n)}{\log\log n}$-long sequences over $\{1, ..., \log n\}$, rather than by logarithmically-long binary sequences. This requires using low degree polynomial extensions (i.e., polynomial of degree $(\log n) - 1$), rather than multi-linear extensions. We can still use a finite field of $\text{poly}(\log(n))$ elements, and so we need only $\frac{O(\log n)}{\log\log n} \cdot O(\log\log n)$ random bits for the summation and low-degree tests. However, the number of queries (needed for obtaining the answers in these tests) grows, because now the polynomials involved have individual degree $(\log n) - 1$ rather than constant individual degree. This merely means that the query complexity increases by a factor of $O(\log n / \log\log n)$. Thus, we obtain $\mathcal{NP} \subseteq \mathcal{PCP}(\texttt{log}, q)$ for $q(n) \stackrel{\rm def}{=} O(\log^2 n)$.

Recall that, in order to use the latter PCP system in composition, we need to guarantee that it (or a version of it) is robust as well as to present a version that is a PCP of proximity. The latter version is relatively easy to obtain (using ideas as applied to the PCP of §9.3.2.1), whereas obtaining robustness is too complex to be described here. We comment that one way of obtaining a robust PCP system is by a generic application of a (randomness-efficient) "parallelization" of PCP systems (cf. [13]), which in turn depends heavily on highly efficient low-degree tests. A alternative approach (cf. [33]) capitalizes of the specific structure of the summation test (as well as on the evident robustness of a simple low-degree test).

Digest. Assuming that $\mathcal{P} \neq \mathcal{NP}$, the PCP Theorem asserts a PCP system that obtains simultaneously the minimal possible randomness and query complexity (up to a multiplicative factor). The forgoing construction obtains this remarkable result by combining two different PCPs: the first PCP obtains logarithmic randomness but uses polylogarithmically many queries, whereas the second PCP uses a constant number of queries but has polynomial randomness complexity. We stress that each of the two PCP systems is highly non-trivial and very interesting by itself. We highlight the fact that these PCPs can be composed using a very simple composition method that refers to auxiliary properties such as robustness and proximity testing. (Composition of PCP systems that lack these extra properties is possible, but is far more cumbersome and complex.)

9.3.2.3 Overview of the second proof of the PCP Theorem

The original proof of the PCP Theorem focuses on the construction of two PCP systems that are highly non-trivial and interesting by themselves, and combines them in a natural manner. Loosely speaking, this combination (via proof composition) *preserves* the good features of each of the two systems; that is, it yields a PCP system that inherits the (logarithmic) randomness complexity of one system and the (constant) query complexity of the other. In contrast, the following alternative proof is focused at the "amplification" of PCP systems, via a gradual process of logarithmically many steps. We start with a trivial "PCP" system that has the desired complexities but rejects false assertions with probability inversely proportional to their length, and double the rejection probability in each step while essentially maintaining the initial complexities. That is, in each step, the constant query complexity of the verifier is preserved and its randomness complexity is increased only by a constant term. Thus, the process gradually transforms an extremely weak PCP system into a remarkably strong PCP system as postulated in the PCP Theorem.

In order to describe the aforementioned process we need to redefine PCP systems so to allow arbitrary soundness error. In fact, for technical reasons, it is more convenient to describe the process as an iterated reduction of a "constraint satisfaction" problem to itself. Specifically, we refer to systems of 2-variable constraints, which are readily represented by (labeled) graphs such that the vertices correspond to (non-Boolean) variables and the edges are associated with constraints.

Definition 9.18 (CSP with 2-variable constraints): *For a fixed finite set Σ, an instance of* CSP *consists of a graph $G = (V, E)$* (which may have parallel edges and self-loops) *and a sequence of 2-variable constraints $\Phi = (\phi_e)_{e \in E}$ associated with the edges, where each constraint has the form $\phi_e : \Sigma^2 \to \{0,1\}$. The value of an assignment $\alpha : V \to \Sigma$ is the number of constraints satisfied by α; that is, the value of α is $|\{(u,v) \in E : \phi_{(u,v)}(\alpha(u), \alpha(v)) = 1\}|$. We denote by $\mathtt{vlt}(G, \Phi)$* (standing for violation) *the fraction of unsatisfied constraints under the best possible* *assignment; that is,*

$$\mathtt{vlt}(G,\Phi) = \min_{\alpha:V\to\Sigma}\{|\{(u,v)\in E : \phi_{(u,v)}(\alpha(u),\alpha(v))=0\}|/|E|\}. \tag{9.9}$$

For various functions $\tau:\mathbb{N}\to(0,1]$, *we will consider the promise problem* $\mathtt{gapCSP}^{\Sigma}_{\tau}$, *having instances as in the foregoing, such that the* yes-*instances are fully satisfiable instances* (i.e., $\mathtt{vlt}=0$) *and the* no-*instances are pairs* (G,Φ) *for which* $\mathtt{vlt}(G,\Phi)\ge\tau(|G|)$ *holds, where* $|G|$ *denotes the number of edges in* G.

Note that 3SAT is reducible to $\mathtt{gapCSP}^{\{1,\ldots,7\}}_{\tau}$ for $\tau(m)=1/m$; see Exercise 9.19. Our goal is to reduce 3SAT (or rather $\mathtt{gapCSP}^{\{1,\ldots,7\}}_{\tau}$) to $\mathtt{gapCSP}^{\Sigma}_{c}$, for some fixed finite Σ and constant $c>0$. The PCP Theorem will follow by showing a simple PCP system for $\mathtt{gapCSP}^{\Sigma}_{c}$; see Exercise 9.21. (The relationship between constraint satisfaction problems and the PCP Theorem is further discussed in Section 9.3.3.) The desired reduction of $\mathtt{gapCSP}^{\Sigma}_{1/m}$ to $\mathtt{gapCSP}^{\Sigma}_{\Omega(1)}$ is obtained by iteratively applying the following reduction logarithmically many times.

Lemma 9.19 (amplifying reduction of gapCSP to itself): *For some finite* Σ *and constant* $c>0$, *there exists a polynomial-time reduction of* $\mathtt{gapCSP}^{\Sigma}$ *to itself such that the following conditions hold with respect to the mapping of any instance* (G,Φ) *to the instance* (G',Φ').

1. *If* $\mathtt{vlt}(G,\Phi)=0$ *then* $\mathtt{vlt}(G',\Phi')=0$.
2. $\mathtt{vlt}(G',\Phi')\ge\min(2\cdot\mathtt{vlt}(G,\Phi),c)$.
3. $|G'|=O(|G|)$.

Proof Outline:[29] The reduction consists of three steps. We first apply a pre-processing step that makes the underlying graph suitable for further analysis. The value of vlt may decrease during this step by a constant factor. The heart of the reduction is the second step in which we may increase vlt by any desired constant factor. The latter step also increases the alphabet Σ, and thus a post-processing step is employed to regain the original alphabet (by using any inner PCP systems; e.g., the one presented in §9.3.2.1). Details follow.

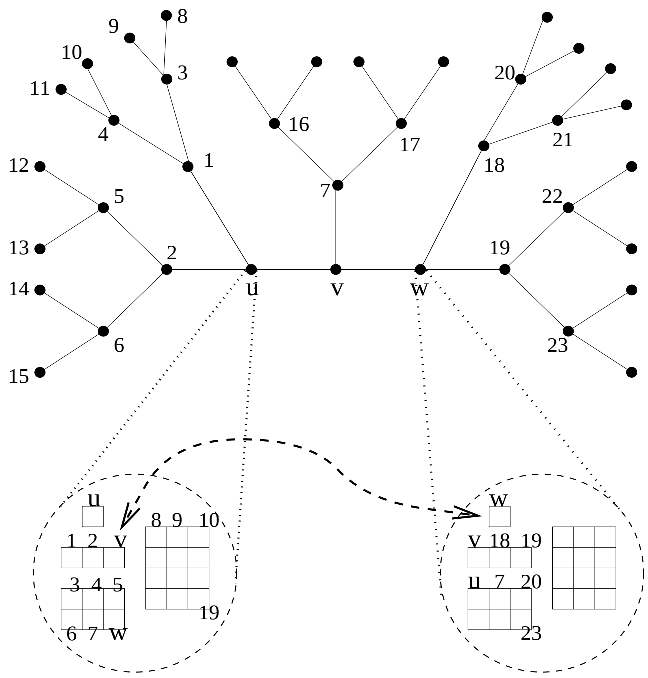
We first stress that the aforementioned Σ and c, as well as the auxiliary parameters d and t (to be introduced in the following two paragraphs), are fixed constants that will be determined such that various conditions (which arise in the course of our argument) are satisfied. Specifically, t will be the last parameter to be determined (and it will be made greater than a constant that is determined by all the other parameters).

We start with the pre-processing step. Our aim in this step is to reduce the input (G,Φ) of $\mathtt{gapCSP}^{\Sigma}$ to an instance (G_1,Φ_1) such that G_1 is a d-regular expander graph.[30] Furthermore, each vertex in G_1 will have at least $d/2$ self-loops,

[29] For details, see [63].

[30] A d-**regular graph** is a graph in which each vertex is incident to exactly d edges. Loosely speaking, an expander graph has the property that each moderately balanced cut (i.e., partition of its vertex set) has relatively many edges crossing it. An equivalent definition, also used in the actual analysis, is that the second eigenvalue of the corresponding adjacency matrix has absolute value that is bounded away from d. For further details, see §E.2.1.1.

the number of edges is preserved up to a constant factor (i.e., $|G_1| = O(|G|)$), and $\mathtt{vlt}(G_1,\Phi_1) = \Theta(\mathtt{vlt}(G,\Phi))$. This step is quite simple: see Exercise 9.22. Intuitively, with respect to intersecting a fixed set of edges, a random (t-edge long) walk on the resulting graph G_1 behave like a random sample of (t) edges, while $|G_1| = O(|G|)$ and $\mathtt{vlt}(G_1,\Phi_1) = \Omega(\mathtt{vlt}(G,\Phi))$.



The alphabet Σ' as a labeling of the distance $t = 3$ neighborhoods, when repetitions are omitted. In this case $d = 6$ but the self-loops are not shown. The two-sided arrow indicates one of the edges in G_1 that will contribute to the edge constraint between u and w in (G_2,Φ_2).

Figure 9.3: The amplifying reduction in the second proof of the PCP Theorem.

The main step is aimed at increasing the fraction of violated constraints by a sufficiently large constant factor. This is done by reducing the instance (G_1,Φ_1) of $\mathtt{gapCSP}^{\Sigma}$ to an instance (G_2,Φ_2) of $\mathtt{gapCSP}^{\Sigma'}$ such that $\Sigma' = \Sigma^{d^t}$. Specifically, the vertex set of G_2 is identical to the vertex set of G_1, and each t-edge long path in G_1 is replaced by a corresponding edge in G_2, which is thus a d^t-regular graph. The constraints in Φ_2 are the natural ones, viewing each element of Σ' as a Σ-labeling of the ("distance $\leq t$") neighborhood of a vertex (see Figure 9.3), and checking that two such labelings are consistent as well as satisfy Φ_1. That is, suppose that there is a path of length at most t in G_1 going from vertex u to vertex w and passing through vertex v. Then, there is an edge in G_2 between vertices u and w, and the constraint associated with it with mandates that the entries corresponding to vertex v in the Σ'-labeling of vertices u and w are identical. In addition, if the G_1-edge (v,v') is on a path of length at most t starting at u then the corresponding edge in G_2 is associated a constraint that enforces the constraint that is associated to (v,v') in Φ_1.

Clearly, if $\mathtt{vlt}(G_1,\Phi_1) = 0$ then $\mathtt{vlt}(G_2,\Phi_2) = 0$. The interesting fact is that the fraction of violated constraints increases by a factor of $\Omega(\sqrt{t})$; that is, $\mathtt{vlt}(G_2,\Phi_2) \geq \min(\Omega(\sqrt{t}\cdot \mathtt{vlt}(G_1,\Phi_1)), c)$. Here we merely provide a rough intuition and refer the interested reader to [63]. The intuition is that any Σ'-labeling to the vertices of G_2 must either be consistent with a Σ-labeling of G_1 or violate the "equality constraints" of many edges in G_2. Focusing on the first case and relying on the hypothesis that G_1 is an expander, it follows that the set of violated edge-constraints (of Φ_1) with respect to the aforementioned Σ-labeling causes many more edge-constraints of Φ_2 to be violated (by virtue of the latter enforcing many edge-constraints of Φ_1). The point is that *any set F of edges of G_1 is likely to appear on a* $\min(\Omega(t)\cdot|F|/|G_1|, \Omega(1))$ *fraction of the edges of* G_2 (i.e., t-paths of G_1). (Note that the claim would have been obvious if G_1 were a complete graph, but it also holds for an expander.)[31]

The factor of $\Omega(\sqrt{t})$ gained in the second step makes up for the constant factor lost in the first step (as well as the constant factor to be lost in the last step). Furthermore, for a suitable choice of the constant t, the aforementioned gain yields an overall constant factor amplification (of $\mathtt{vlt}$). However, so far we obtained an instance of $\mathtt{gapCSP}^{\Sigma'}$ rather than an instance of $\mathtt{gapCSP}^{\Sigma}$, where $\Sigma' = \Sigma^{d^t}$. The purpose of the last step is to reduce the latter instance to an instance of $\mathtt{gapCSP}^{\Sigma}$. This is done by viewing the instance of $\mathtt{gapCSP}^{\Sigma'}$ as a (weak) PCP system (analogously to Exercise 9.21), and composing it with an inner-verifier using the proof composition paradigm outlined in §9.3.2.2. We stress that the inner-verifier used here needs only handle instances of constant size (i.e., having description length $O(d^t \log|\Sigma|)$), and so the verifier presented in §9.3.2.1 will do. The resulting PCP-system uses randomness $r \stackrel{\rm def}{=} \log_2|G_2| + O(d^t\log|\Sigma|)^2$ and a constant number of binary queries, and has rejection probability $\Omega(\mathtt{vlt}(G_2,\Phi_2))$, which is independent of the choice of the constant t. As in Exercise 9.19, for $\Sigma = \{0,1\}^{O(1)}$, we can easily obtain an instance of $\mathtt{gapCSP}^{\Sigma}$ that has a $\Omega(\mathtt{vlt}(G_2,\Phi_2))$ fraction of violated constraints. Furthermore, the size of the resulting instance is $O(2^r) = O(|G_2|)$, because d and t are constants. This completes the last step as well as the (outline of the) proof of the entire lemma. $\square$

9.3.3 PCP and Approximation

The characterization of $\mathcal{NP}$ in terms of probabilistically checkable proofs plays a central role in the study of the complexity of approximation problems (cf., Section 10.1.1). To demonstrate this relationship, we first note that a PCP system V gives rise to a natural approximation problem; that is, on input x, the task is approximating the probability that V accepts x when given oracle access to

[31] We also note that due to a technical difficulty it is easier to establish the claimed bound of $\Omega(\sqrt{t}\cdot\mathtt{vlt}(G_1,\Phi_1))$ rather than $\Omega(t\cdot\mathtt{vlt}(G_1,\Phi_1))$.

the best possible π (i.e., we wish to approximate $\max_\pi\{\mathsf{Pr}[V^\pi(x) = 1]\}$). Thus, *if* $S \in \mathcal{PCP}(r, q)$ *then deciding membership in* S *is reducible to approximating the maximum among* $\exp(2^{r+q})$ *quantities* (corresponding to all effective oracles), where each quantity can be evaluated in time $2^r \cdot \text{poly}$. Note that *an approximation up to a constant factor* (of 2) *will do.*

Note that the foregoing approximation problem is parameterized by a PCP verifier V, and its instances are given their value with respect to this verifier (i.e., the instance x has value $\max_\pi\{\mathsf{Pr}[V^\pi(x) = 1]\}$). This per se does not yield a "natural" approximation problem. In order to link PCP systems with natural approximation problems, we take a closer look at the approximation problem associated with $\mathcal{PCP}(r, q)$. For simplicity, we focus on the case of non-adaptive PCP systems (i.e., all the queries are determined beforehand based on the input and the internal coin tosses of the verifier). Fixing an input x for such a system, we consider the $2^{r(|x|)}$ formulae that represent the decision of the verifier on each of the possible outcomes of its coin tosses after inspecting the corresponding bits in the proof oracle. That is, each of these $2^{r(|x|)}$ formulae depends on $q(|x|)$ Boolean variables that represent the values of the corresponding bits in the proof oracle. Thus, if x is a yes-instance then there exists a truth assignment (to these variables) that satisfies all $2^{r(|x|)}$ formulae, whereas if x is a no-instance then there exists no truth assignment that satisfies more than $2^{r(|x|)-1}$ formulae. Furthermore, in the case that $r(n) = O(\log n)$, given x, we can construct the corresponding sequence of formulae in polynomial-time. Hence, the PCP Theorem (i.e., Theorem 9.16) yields *NP-hardness results regarding the approximation of the number of simultaneously satisfiable Boolean formulae.* When focusing on the case that q is constant, this motivates the following definition.

Definition 9.20 (gap problems for SAT and generalized-SAT): *For constants* $q \in \mathbb{N}$ *and* $\varepsilon > 0$*, the promise problem* $\mathtt{gapGSAT}_\varepsilon^q$ *consists of instances that are each a sequence of* q*-variable Boolean formulae. The* yes*-instances are sequences that are simultaneously satisfiable, whereas the* no*-instances are sequences for which no Boolean assignment satisfies more than a* $1 - \varepsilon$ *fraction of the formulae in the sequence. The promise problem* $\mathtt{gapSAT}_\varepsilon^q$ *is defined analogously, except that in this case each instance is a sequence of formulae that are each a single disjunctive clause.*

Indeed, each instance of $\mathtt{gapSAT}_\varepsilon^q$ is naturally viewed as q-CNF formulae, and we consider an assignment that satisfies as many clauses (of the input CNF) as possible. As hinted, $\mathcal{NP} \subseteq \mathcal{PCP}(\mathtt{log}, O(1))$ implies that $\mathtt{gapGSAT}_{1/2}^{O(1)}$ is NP-complete, which in turn implies that for some constant $\varepsilon > 0$ the problem $\mathtt{gapSAT}_\varepsilon^3$ is NP-complete. The converses hold too. All these claims are stated and proved next.

Theorem 9.21 (equivalent formulations of the PCP Theorem). *The following three conditions are equivalent:*

1. The PCP Theorem: *there exists a constant* q *such that* $\mathcal{NP} \subseteq \mathcal{PCP}(\mathtt{log}, q)$.
2. *There exists a constant* q *such that* $\mathtt{gapGSAT}_{1/2}^q$ *is* $\mathcal{NP}$*-hard.*

3. *There exists a constant $\varepsilon > 0$ such that* $\texttt{gapSAT}^3_\varepsilon$ *is $\mathcal{NP}$-hard.*

Note that Items 2 and 3 make no reference to PCP. Their equivalence to Item 1 manifests that the hardness of approximating natural optimization problems lies at the heart of the PCP Theorem. In general, probabilistically checkable proof systems for $\mathcal{NP}$ yield strong inapproximability results for various classical optimization problems (cf., Exercise 9.14 and Section 10.1.1).

Proof: We first show that the PCP Theorem implies the NP-hardness of $\texttt{gapGSAT}$. We may assume, without loss of generality, that, for some constant q and every $S \in \mathcal{NP}$, it holds that $S \in \mathcal{PCP}(O(\log), q)$ via a non-adaptive verifier (because q adaptive queries can be emulated by 2^q non-adaptive queries). We reduce S to $\texttt{gapGSAT}$ as follows. On input x, we scan all $2^{O(\log |x|)}$ possible sequence of outcomes of the verifier's coin tosses, and for each such sequence of outcomes we determine the queries made by the verifier as well as the residual decision predicate (where this predicate determines which sequences of answers lead this verifier to accept). That is, for each random-outcome $\omega \in \{0,1\}^{O(\log |x|)}$, we consider the residual predicate, determined by x and ω, that specifies which q-bit long sequence of oracle answers makes the verifier accept x on coins ω. Indeed, this predicate depends only on q variables (which represent the values of the q corresponding oracle answers). Thus, we map x to a sequence of poly$(|x|)$ formulae, each depending on q variables, obtaining an instance of $\texttt{gapGSAT}^q$. This mapping can be computed in polynomial-time, and indeed $x \in S$ (resp., $x \notin S$) is mapped to a yes-instance (resp., no-instance) of $\texttt{gapGSAT}^q_{1/2}$.

Item 2 implies Item 3 by a standard reduction of GSAT to 3SAT. Specifically, $\texttt{gapGSAT}^q_{1/2}$ reduces to $\texttt{gapSAT}^q_{2^{-(q+1)}}$, which in turn reduces to $\texttt{gapSAT}^3_\varepsilon$ for $\varepsilon = 2^{-(q+1)}/(q-2)$. Note that Item 3 implies Item 2 (e.g., given an instance of $\texttt{gapSAT}^3_\varepsilon$, consider all possible conjunctions of $1/\varepsilon$ disjunctive clauses in the given instance).

We complete the proof by showing that Item 3 implies Item 1. (The same proof shows that Item 2 implies Item 1.) In fact, we show that $\texttt{gapGSAT}^q_\varepsilon$ is in $\mathcal{PCP}(O(\varepsilon^{-1}\log), O(q/\varepsilon))$, and do so by presenting a very natural PCP system. In this PCP system the proof oracle is supposed to be an satisfying assignment, and the verifier selects at random one of the (q-variable) formulae in the input sequence, and checks whether it is satisfied by the (assignment given by the) oracle. This amounts to tossing logarithmically many coins and making q queries. This verifier always accepts yes-instances (when given access to an adequate oracle), whereas each no-instances is rejected with probability at least ε (no matter which oracle is used). To amplify the rejection probability (to the desired threshold of 1/2), we invoke the foregoing verifier $O(\varepsilon^{-1})$ times. ■

Gap amplifying reductions – a reflection. Items 2 and 3 of Theorem 9.21 assert the existence of "gap amplifying" reductions of problems like 3SAT to themselves. These reductions map yes-instances to yes-instances (as usual), while mapping no-instances to no-instances of a special type such that a "gap" is created between the yes-instances and no-instances at the image of the reduction. For example, in the case of 3SAT, unsatisfiable formulae are mapped to formulae that are not merely unsatisfiable but rather have no assignment that satisfies more than a $1-\varepsilon$ fraction of the clauses. Thus, PCP constructions are essentially "gap amplifying" reductions.

9.3.4 More on PCP itself: an overview

We start by discussing variants of the PCP characterization of NP, and next turn to PCPs having expressing power beyond NP. Needless to say, the latter systems have super-logarithmic randomness complexity.

9.3.4.1 More on the PCP characterization of NP

Interestingly, the two complexity measures in the PCP-characterization of $\mathcal{NP}$ can be traded off such that at the extremes we get $\mathcal{NP} = \mathcal{PCP}(\mathtt{log}, O(1))$ and $\mathcal{NP} = \mathcal{PCP}(0, \mathtt{poly})$, respectively.

Proposition 9.22 *For every $S \in \mathcal{NP}$, there exists a logarithmic function ℓ such that, for every integer function k that satisfies $0 \leq k(n) \leq \ell(n)$, it holds that $S \in \mathcal{PCP}(\ell - k, O(2^k)) \subseteq \mathcal{NP}$.*

Proof Sketch: By Theorem 9.16, we have $S \in \mathcal{PCP}(\ell, O(1))$. Consider an emulation of the corresponding verifier in which we try all possibilities for the $k(n)$-bit long prefix of its random-tape. Lastly, recall that $\mathcal{PCP}(\mathtt{log}, \mathtt{poly}) \subseteq \mathcal{NP}$. $\square$

Following the establishment of Theorem 9.16, numerous variants of the PCP Characterization of NP were explored. These variants refer to a finer evaluation of various parameters of probabilistically checkable proof systems (for sets in $\mathcal{NP}$). Following is a brief summary of some of these studies.[32]

The length of PCPs. Recall that the effective length of the oracle in any $\mathcal{PCP}(\mathtt{log}, \mathtt{log})$ system is polynomial (in the length of the input). Furthermore, in the PCP systems underlying the proof of Theorem 9.16 the queries refer only to a polynomially long prefix of the oracle, and so the actual length of these PCPs for $\mathcal{NP}$ is polynomial. Remarkably, *the length of PCPs for $\mathcal{NP}$ can be made nearly-linear* (in the combined length of the input and the standard NP-witness), *while maintaining constant query complexity, where by nearly-linear we mean linear up to a poly-logarithmic factor.* (For details see [34, 63].) This means that a *relatively modest amount of redundancy* in the proof oracle suffices for supporting probabilistic verification via a constant number of probes.

The number of queries in PCPs. Theorem 9.16 asserts that a constant number of queries suffice for PCPs with logarithmic randomness and soundness error of 1/2 (for NP). It is currently known that this constant is at most *five*, whereas with *three* queries one may get arbitrary close to a soundness error of 1/2. The

[32] With the exception of works that appeared after [86], we provide no references for the results quoted here. We refer the interested reader to [86, Sec. 2.4.4].

obvious trade-off between the number of queries and the soundness error gives rise to the robust notion of amortized query complexity, defined as the ratio between the number of queries and (minus) the logarithm (to based 2) of the soundness error. *For every $\varepsilon > 0$, any set in $\mathcal{NP}$ has a PCP system with logarithmic randomness and amortized query complexity* $1+\varepsilon$ (cf. [114]), whereas only sets in $\mathcal{P}$ have PCPs of logarithmic randomness and amortized query complexity 1 (or less).

The free-bit complexity. The motivation to the notion of free bits came from the PCP–to–MaxClique connection (see Exercise 9.14 and [27, Sec. 8]), but we believe that this notion is of independent interest. Intuitively, this notion distinguishes between queries for which the acceptable answer is determined by previously obtained answers (i.e., the verifier compares the answer to a value determined by the previous answers) and queries for which the verifier only records the answer for future usage. The latter queries are called free (because any answer to them is "acceptable"). For example, in the linearity test (see §9.3.2.1) the first two queries are free and the third is not (i.e., the test accepts if and only if $f(x) + f(y) = f(x + y)$). The amortized free-bit complexity is define analogously to the amortized query complexity. Interestingly, *$\mathcal{NP}$ has PCPs with logarithmic randomness and amortized free-bit complexity less than any positive constant.*

Adaptive versus non-adaptive verifiers. Recall that a PCP verifier is called non-adaptive if its queries are determined solely based on its input and the outcome of its coin tosses. (A general verifier, called adaptive, may determine its queries also based on previously received oracle answers.) Recall that the PCP Characterization of NP (i.e., Theorem 9.16) is established using a non-adaptive verifier; however, it turns out that *adaptive verifiers are more powerful than non-adaptive ones in terms of quantitative results*: Specifically, for PCP verifiers making *three* queries and having logarithmic randomness complexity, adaptive queries provide for soundness error at most 0.51 (actually $0.5 + \varepsilon$ for any $\varepsilon > 0$) for any set in $\mathcal{NP}$, whereas *non-adaptive* queries provide soundness error 5/8 (or less) only for sets in $\mathcal{P}$.

Non-binary queries. Our definition of PCP allows only binary queries. Certainly, non-binary queries can always be coded as binary ones, but the converse is not necessarily valid, in particular in adversarial settings. Note that the soundness condition constitutes an implicit adversarial setting, where a bad proof may be thought of as being selected by an adversary. Thus, when several binary queries are packed into one non-binary query, the adversary need not respect the packing (i.e., it may answer inconsistently on the same binary query depending on the other queries packed with it). For this reason, "parallel repetition" is highly non-trivial in the PCP setting. Still, a Parallel Repetition Theorem that refers to independent invocations of the same PCP is known, but it is not applicable for obtaining soundness error smaller than a constant (while preserving logarithmic randomness). Nevertheless, using adequate "consistency tests" one may construct PCP systems for $\mathcal{NP}$ using logarithmic randomness, a constant number of (non-binary) queries and *soundness error exponential in the length of the answers.* (Currently, this is known only for sub-logarithmic answer lengths.)

9.3.4.2 PCP with super-logarithmic randomness

Our focus in §9.3.4.1 was on the important case where the verifier tosses logarithmically many coins, and hence the "effective proof length" is polynomial. Here we mention that the PCP Theorem scales up.[33]

Theorem 9.23 (Theorem 9.16 – Generalized): *Let $t(\cdot)$ be an integer function such that $n<t(n)<2^{\mathrm{poly}(n)}$. Then,* $\mathrm{N{\small TIME}}(t) \subseteq \mathcal{PCP}(O(\log t), O(1))$.

Recall that $\mathcal{PCP}(r,q) \subseteq \mathrm{N{\small TIME}}(t)$, for $t(n) = \mathrm{poly}(n) \cdot 2^{r(n)}$. Thus, the NTIME Hierarchy implies a hierarchy of $\mathcal{PCP}(\cdot, O(1))$ classes, for randomness complexity ranging between logarithmic and polynomial functions.

Chapter Notes

(The following historical notes are quite long and still they fail to properly discuss several important technical contributions that played an important role in the development of the area. For further details, the reader is referred to [86, Sec. 2.6.2].)

Motivated by the desire to formulate the most general type of "proofs" that may be used within cryptographic protocols, Goldwasser, Micali and Rackoff [105] introduced the notion of an *interactive proof system.* Although the main thrust of their work was the introduction of a special type of interactive proofs (i.e., ones that are *zero-knowledge*), the possibility that interactive proof systems may be more powerful from NP-proof systems was pointed out in [105]. Independently of [105], Babai [16] suggested a different formulation of interactive proofs, which he called *Arthur-Merlin Games.* Syntactically, Arthur-Merlin Games are a restricted form of interactive proof systems, yet it was subsequently shown that these restricted systems are as powerful as the general ones (cf., [107]). The speed-up result (i.e., $\mathcal{AM}(2f) \subseteq \mathcal{AM}(f)$) is due to [20] (improving over [16]).

The first evidence of the power of interactive proofs was given by Goldreich, Micali, and Wigderson [96], who presented an interactive proof system for Graph Non-Isomorphism (Construction 9.3). More importantly, they demonstrated the *generality and wide applicability of zero-knowledge proofs*: Assuming the existence of one-way function, they showed how to construct zero-knowledge interactive proofs for any set in $\mathcal{NP}$ (Theorem 9.11). This result has had a dramatic impact on the design of cryptographic protocols (cf., [97]). For further discussion of zero-knowledge and its applications to cryptography, see Appendix C. Theorem 9.12 (i.e., $\mathcal{ZK} = \mathcal{IP}$) is due to [30, 123].

[33] This scaling up is not straightforward, since we wish to maintain polynomial-time verification. The key point is that the CNF formulae that represent the computation of NTIME are highly uniform, and thus the corresponding Boolean functions (and their low degree extensions) can be evaluated in polynomial-time.

Probabilistically checkable proof (PCP) systems are related to *multi-prover interactive proof systems*, a generalization of interactive proofs that was suggested by Ben-Or, Goldwasser, Kilian and Wigderson [31]. Again, the main motivation came from the zero-knowledge perspective; specifically, introducing multi-prover zero-knowledge proofs for $\mathcal{NP}$ without relying on intractability assumptions. Yet, the complexity theoretic prospects of the new class, denoted $\mathcal{MIP}$, have not been ignored.

The amazing power of interactive proof systems has been demonstrated by using algebraic methods. The basic technique has been introduced by Lund, Fortnow, Karloff and Nisan [151], who applied it to show that the polynomial-time hierarchy (and actually $\mathcal{P}^{\#\mathcal{P}}$) is in $\mathcal{IP}$. Subsequently, Shamir [192] used the technique to show that $\mathcal{IP} = \mathcal{PSPACE}$, and Babai, Fortnow and Lund [17] used it to show that $\mathcal{MIP} = \mathcal{NEXP}$. (Our entire proof of Theorem 9.4 follows [192].)

The aforementioned multi-prover proof system of Babai, Fortnow and Lund [17] (hereafter referred to as the BFL proof system) has been the starting point for fundamental developments regarding $\mathcal{NP}$. The first development was the discovery that the BFL proof system can be "scaled-down" from $\mathcal{NEXP}$ to $\mathcal{NP}$. This important discovery was made independently by two sets of authors: Babai, Fortnow, Levin, and Szegedy [18] and Feige, Goldwasser, Lovász, and Safra [69]. However, the manner in which the BFL proof is scaled-down is different in the two papers, and so are the consequences of the scaling-down.

Babai *et. al.* [18] start by considering (only) inputs encoded using a special error-correcting code. The encoding of strings, relative to this error-correcting code, can be computed in polynomial time. They presented an almost-linear time algorithm that transforms NP-witnesses (to inputs in a set $S \in \mathcal{NP}$) into *transparent proofs* that can be verified (as vouching for the correctness of the encoded assertion) in (probabilistic) *poly-logarithmic time* (by a Random Access Machine). Babai *et. al.* [18] stress the practical aspects of transparent proofs; specifically, for rapidly checking transcripts of long computations.

In contrast, in the proof system of Feige *et. al.* [69, 70] the verifier stays polynomial-time and only two more refined complexity measures (i.e., the randomness and query complexities) are reduced to poly-logarithmic. This eliminates the need to assume that the input is in a special error-correcting form, and yields a refined (quantitative) version of the notion of probabilistically checkable proof systems (introduced in [76]), where the refinement is obtained by specifying the randomness and query complexities (see Definition 9.14). Hence, whereas the BFL proof system [17] can be reinterpreted as establishing $\mathcal{NEXP} = \mathcal{PCP}(\texttt{poly}, \texttt{poly})$, the work of Feige *et. al.* [70] establishes $\mathcal{NP} \subseteq \mathcal{PCP}(f, f)$, where $f(n) = O(\log n \cdot \log\log n)$. (In retrospect, we note that the work of Babai *et. al.* [18] implies that $\mathcal{NP} \subseteq \mathcal{PCP}(\texttt{log}, \texttt{polylog})$, but the latter terminology was not available at the time.)

Interest in the new complexity class became immense since Feige *et. al.* [69, 70] demonstrated its relevance to proving the intractability of approximating some combinatorial problems (specifically, for MaxClique). When using the PCP–to–MaxClique connection established by Feige *et. al.*, the randomness and query complexities of the verifier (in a PCP system for an NP-complete set) relate to the strength of the negative results obtained for approximation problems. This fact provided a very strong motivation for trying to reduce these complexities and obtain a tight characterization of $\mathcal{NP}$ in terms of $\mathcal{PCP}(\cdot,\cdot)$. The obvious challenge was showing that $\mathcal{NP}$ equals $\mathcal{PCP}(\texttt{log},\texttt{log})$. This challenge was met by Arora and Safra [14]. Actually, they showed that $\mathcal{NP} = \mathcal{PCP}(\texttt{log}, q)$, where $q(n) = o(\log n)$.

Hence, a new challenge arose; namely, further reducing the query complexity – in particular, to a constant – while maintaining the logarithmic randomness complexity. Again, additional motivation for this challenge came from the relevance of such a result to the study of approximation problems. The new challenge was met by Arora, Lund, Motwani, Sudan and Szegedy [13], and is captured by the PCP Characterization Theorem, which asserts that $\mathcal{NP} = \mathcal{PCP}(\texttt{log}, O(1))$.

Indeed the PCP Characterization Theorem is a culmination of a sequence of impressive works [151, 17, 18, 70, 14, 13]. These works are rich in innovative ideas (e.g., various arithmetizations of SAT as well as various forms of proof composition) and employ numerous techniques (e.g., low-degree tests, self-correction, and pseudorandomness).

Our overview of the original proof of the PCP Theorem (in §9.3.2.1–9.3.2.2) is based on [13, 14].[34] The alternative proof outlined in §9.3.2.3 is due to Dinur [63]. We also mention some of the ideas and techniques involved in deriving even stronger variants of the PCP Theorem (which are surveyed in §9.3.4.1). These include the Parallel Repetition Theorem [173], the use of the Long-Code [27], and the application of Fourier analysis in this setting [111, 112].

Computationally-Sound Proof Systems. Argument systems were defined by Brassard, Chaum and Crépeau [46], with the motivation of providing *perfect* zero-knowledge arguments (rather than zero-knowledge *proofs*) for $\mathcal{NP}$. A few years later, Kilian [136] demonstrated their significance beyond the domain of zero-knowledge by showing that, under some reasonable intractability assumptions, every set in $\mathcal{NP}$ has a computationally-sound proof in which the randomness and communication complexities are poly-logarithmic.[35] Interestingly, these argument systems rely on the fact that $\mathcal{NP} \subseteq \mathcal{PCP}(f, f)$, for $f(n) = \text{poly}(\log n)$. We mention that Micali [154] suggested a different type of computationally-sound proof systems (which he called CS-proofs).

Final comment: The current chapter is a revision of [86, Chap. 2]. In particular, more details are provided here for the main topics, whereas numerous secondary topics discussed in [86, Chap. 2] are not mentioned here (or are only briefly mentioned here). In addition, a couple of the research directions that were mentioned in [86, Sec. 2.4.4] received considerable attention in the period that elapsed, and improved results are currently known. In particular, the interested reader is referred to [33, 34, 63] (for a study of the length of PCPs) and to [114] (for a study

[34] Our presentation also benefits from the notions of PCPs of proximity and robustness, put forward in [33, 64].

[35] We comment that interactive proofs are unlikely to have such low complexities; see [102].

of their amortized query complexity).

Exercises

Exercise 9.1 (parallel error-reduction for interactive proof systems) Prove that the error probability (in the soundness condition) can be reduced by parallel repetitions of the proof system.

Guideline: As a warm-up consider first the case of public-coin interactive proof systems. Next, note that the analysis generalizes to arbitrary interactive proof systems. (Extra hint: As a mental experiment, consider a "powerful verifier" that emulates the original verifier while behaving as in the public-coin model.) A proof appears in [86, Apdx. C.1].

Exercise 9.2 Complete the details of the proof that co$\mathcal{NP} \subseteq \mathcal{IP}$ (i.e., the first part of the proof of Theorem 9.4). In particular, regarding the proof of non-satisfiability of a CNF with n variables and m clauses, what is the length of the messages sent by the two parties? What is the soundness error?

Exercise 9.3 Present a $n/O(\log n)$-round interactive proof for the non-satisfiability of a CNF having n variables.

Guideline: Modify the (first part of the) proof of Theorem 9.4, by stripping $O(\log n)$ summations in each round.

Exercise 9.4 (an interactive proof system for $\#\mathcal{P}$) Using the main part of the proof of Theorem 9.4, present a proof system for the counting set of Eq. (9.5).

Guideline: Use a slightly different arithmetization of CNF formulae. Specifically, instead of replacing the clause $x \vee \neg y \vee z$ by the term $(x + (1-y) + z)$, replace it by the term $(1-((1-x)\cdot y \cdot (1-z)))$.

Exercise 9.5 Show that `QBF` can be reduced to a special form of `QBF` in which no variable appears both to the left and the right of more than one universal quantifier.

Guideline: Consider a process (which proceeds from left to right) of "refreshing" variables after each universal quantifier. Let $\phi(x_1,...,x_s,y,x_{s+1},...,x_{s+t})$ be a quantifier-free boolean formula and let $Q_{s+1},...,Q_{s+t}$ be an arbitrary sequence of quantifiers. Then, we replace the quantified (sub-)formula

$$\forall y Q_{s+1}x_{s+1} \cdots Q_{s+t}x_{s+t}\, \phi(x_1,...,x_s,y,x_{s+1},...,x_{s+t})$$

by the (sub-)formula

$$\forall y \exists x'_1 \cdots \exists x'_s [(\wedge_{i=1}^{s}(x'_i = x_i)) \ \wedge \ Q_{s+1}x_{s+1} \cdots Q_{s+t}x_{s+t}\, \phi(x'_1,...,x'_s,y,x_{s+1},...,x_{s+t})]\,.$$

Note that the variables $x_1,...,x_s$ do not appear to the right of the quantifier Q_{s+1} in the replaced formula, and that the length of the replaced formula grows by an additive term of $O(s)$. This process of refreshing variables is applied from left to right on the entire sequence of universal quantifiers (except the inner one, for which this refreshing is useless).[36]

[36] For example,

$$\exists z_1 \forall z_2 \exists z_3 \forall z_4 \exists z_5 \forall z_6\, \phi(z_1,z_2,z_3,z_4,z_5,z_6)$$

Exercise 9.6 Prove that if two integers in $[0, M]$ are different then they must be different modulo most of the primes in the interval $[3, L]$, where $L = \text{poly}(\log M)$]. Prove the same for the interval $[L, 2L]$.

Guideline: Let $a \neq b \in [0, M]$ and let $P_1, ..., P_t$ be an enumeration of the primes in the interval $[3, \text{poly}(\log M)]$ such that for every $i = 1, ..., t$ it holds that $a \equiv b \pmod{P_i}$. Using the Chinese Reminder Theorem, prove that $Q \stackrel{\text{def}}{=} \prod_{i=1}^{t} P_i \leq M$ (because otherwise $a = b$ follows by combining $a \equiv b \pmod{Q}$ with the hypothesis $a, b \in [0, M]$). It follows that $t < \log_2 M$. Using a lower-bound on the density of prime numbers, the claim follows.

Exercise 9.7 (on interactive proofs with two-sided error (following [78])) Let $\mathcal{IP}'(f)$ denote the class of sets having a two-sided error interactive proof system in which a total of $f(|x|)$ messages are exchanged on common input x. Similarly, let $\mathcal{AM}'$ denote the public-coin version of $\mathcal{IP}'$.

1. Establish $\mathcal{IP}'(f) \subseteq \mathcal{AM}'(f + 3)$ by noting that the proof of Theorem F.2, which establishes $\mathcal{IP}(f) \subseteq \mathcal{AM}(f+3)$, extends to the two-sided error setting.
2. Prove that $\mathcal{AM}'(f) \subseteq \mathcal{AM}'(f + 1)$ by extending the ideas underlying the proof of Theorem 6.7, which actually establishes that $\mathcal{BPP} \subseteq \mathcal{AM}(1)$ (where $\mathcal{BPP} = \mathcal{AM}'(0)$).

Using the Round Speed-up Theorem (i.e., Theorem F.3), conclude that, for every function $f : \mathbb{N} \to \mathbb{N} \setminus \{1\}$, it holds that $\mathcal{IP}'(f) = \mathcal{AM}(f) = \mathcal{IP}(f)$.

Guideline: We focus on establishing $\mathcal{AM}'(f) \subseteq \mathcal{AM}(f+1)$ for arbitrary f (rather than for $f \equiv 0$). Consider an optimal prover strategy and the set of verifier coins that make the verifier accept any fixed yes-instance. Applying the ideas underlying the transformation of $\mathcal{BPP}$ to $\mathcal{MA} = \mathcal{AM}(1)$, we obtain the desired result. For further details, see [78].

Exercise 9.8 In continuation to Exercise 9.7, show that $\mathcal{IP}'(f) = \mathcal{IP}(f)$ for every function $f : \mathbb{N} \to \mathbb{N}$ (including $f \equiv 1$).

Guideline: Focus on establishing $\mathcal{IP}'(1) = \mathcal{IP}(1)$, which is identical to Part 2 of Exercise 6.12. Note that the relevant classes defined in Exercise 6.12 coincide with $\mathcal{IP}(1)$ and $\mathcal{IP}'(1)$; that is, $\mathcal{MA} = \mathcal{IP}(1)$ and $\mathcal{MA}^{(2)} = \mathcal{IP}'(1)$.

Exercise 9.9 (on the role of soundness error in zero-knowledge proofs) Prove that if S has a zero-knowledge interactive proof system with perfect soundness (i.e., soundness error equals zero) then $S \in \mathcal{BPP}$.

---

is first replaced by

$$\exists z_1 \forall z_2 \exists z_1' \, [(z_1' = z_1) \wedge \exists z_3 \forall z_4 \exists z_5 \forall z_6 \, \phi(z_1', z_2, z_3, z_4, z_5, z_6)]$$

and next (written as $\exists z_1 \forall z_2' \exists z_1' \, [(z_1' = z_1) \wedge \exists z_3' \forall z_4' \exists z_5' \forall z_6' \, \phi(z_1', z_2', z_3', z_4', z_5', z_6')]$) is replaced by

$$\begin{array}{l} \exists z_1 \forall z_2' \exists z_1' \, [(z_1' = z_1) \wedge \exists z_3' \forall z_4' \exists z_1'' \exists z_2'' \exists z_3'' \\ \qquad [(\wedge_{i=1}^{3} (z_i'' = z_i')) \wedge \exists z_5' \forall z_6' \phi(z_1'', z_2'', z_3'', z_4', z_5', z_6')]]. \end{array}$$

Thus, in the resulting formula, no variable appears both to the left and to the right of more than a single universal quantifier.

Guideline: Let M be an arbitrary algorithm that simulates the view of the (honest) verifier. Consider the algorithm that on input x, accepts x if and only if $M(x)$ represents a valid view of the verifier in an accepting interaction (i.e., an interaction that leads the verifier to accept the common input x). Use the simulation condition to analyze the case $x \in S$, and the perfect soundness hypothesis to analyze the case $x \notin S$.

Exercise 9.10 (on the role of interaction in zero-knowledge proofs) Prove that if S has a zero-knowledge interactive proof system with a uni-directional communication then $S \in \mathcal{BPP}$.

Guideline: Let M be an arbitrary algorithm that simulates the view of the (honest) verifier, and let $M'(x)$ denote the part of this view that consists of the prover message. Consider the algorithm that on input x, obtains $m \leftarrow M'(x)$, and emulates the verifier's decision on input x and message m. Note that this algorithm ignores the part of $M(x)$ that represents the verifier's internal coin tosses, and uses fresh verifier's coins when deciding on (x, m).

Exercise 9.11 (on the effective length of PCP oracles) Suppose that V is a PCP verifier of query complexity q and randomness complexity r. Show that for every fixed x, the number of possible locations in the proof oracle that are examined by V on input x (when considering all possible internal coin tosses of V and all possible answers it may receive) is upper-bounded by $2^{q(|x|)+r(|x|)}$. Show that if V is non-adaptive then the upper-bound can be improved to $2^{r(|x|)} \cdot q(|x|)$. (Hint: In the adaptive case, the i^{th} query is determined by V's internal coin tosses and the previous $i-1$ answers. In the non-adaptive case, all q queries are determined by V's internal coin tosses.)

Exercise 9.12 (upper-bounds on the complexity of PCPs) Suppose that a set S has a PCP of query complexity q and randomness complexity r. Show that S can be decided by a non-deterministic machine that, on input of length n, makes at most $2^{r(n)} \cdot q(n)$ non-deterministic[37] steps and halts within a total number of $2^{r(n)} \cdot \text{poly}(n)$ steps. Thus, $S \in \text{NTIME}(2^r \cdot \text{poly}) \cap \text{DTIME}(2^{2^r q} \cdot \text{poly})$.

Guideline: For each input $x \in S$ and each possible value $\omega \in \{0,1\}^{r(|x|)}$ of the random-tape, we consider a sequence of $q(|x|)$ bit values that represent a sequence of oracle answers that make the verifier accept. Indeed, for fixed x and $\omega \in \{0,1\}^{r(|x|)}$, each setting of the $q(|x|)$ oracle answers determine the computation of the corresponding verifier (including the queries it makes).

Exercise 9.13 (on the effective randomness of PCPs) Suppose that a set S has a PCP of query complexity q that utilizes proof oracles of length ℓ. Show that, for every constant $\varepsilon > 0$, the set S has a "non-uniform" PCP of query complexity q, soundness error $0.5 + \varepsilon$ and randomness complexity r such that $r(n) = O(1) + \log_2(\ell(n) + n)$. By a "non-uniform PCP" we mean one in which the verifier is a probabilistic polynomial-time oracle machine that is given direct access to a non-uniform $\text{poly}(\ell)$-bit long advice.

[37] See §4.2.1.3 for definition of non-deterministic machines.

Guideline: Consider a PCP verifier V as in the hypothesis, and denote its randomness complexity by r_V. We construct a non-uniform verifier V' that, on input of length n, obtains as advice a set $R_n \subseteq \{0,1\}^{r_V(n)}$ of cardinality $O((\ell(n)+n)/\varepsilon^2)$, and emulates V on a uniformly selected element of R_n. Show that for a random R_n of the said size, the verifier V' satisfies the claims of the exercise.
(Extra hint: Fixing any input $x \notin S$ and any oracle $\pi \in \{0,1\}^{\ell(|x|)}$, upper-bound the probability that a random set R_n causes V' to accept x with probability $0.5+\varepsilon$ when using the oracle π.)

Exercise 9.14 (The FGLSS-reduction [70]) For any $S \in \mathcal{PCP}(r,q)$, consider the following mapping of instances for S to instances of the `Independent Set` problem. The instance x is mapped to a graph $G_x = (V_x, E_x)$, where $V_x \subseteq \{0,1\}^{r(|x|)+q(|x|)}$ consists of pairs (ω, α) such that the PCP verifier *accepts* the input x, when using coins $\omega \in \{0,1\}^{r(|x|)}$ and receiving the answers $\alpha = \alpha_1 \cdots \alpha_{q(|x|)}$ (to the oracle queries determined by x, r and the previous answers). Note that V_x contains only accepting "views" of the verifier. The set E_x consists of edges that connect vertices that represents inconsistent view of the said verifier; that is, the vertex $v = (\omega, \alpha_1 \cdots \alpha_{q(|x|)})$ is connected to the vertex $v' = (\omega', \alpha'_1 \cdots \alpha'_{q(|x|)})$ if there exists i and i' such that $\alpha_i \neq \alpha'_{i'}$ and $\mathsf{q}_i^x(v) = \mathsf{q}_{i'}^x(v')$, where $\mathsf{q}_i^x(v)$ (resp., $\mathsf{q}_{i'}^x(v')$) denotes the i-th (resp., i'-th) query of the verifier on input x, when using coins ω (resp., ω') and receiving the answers $\alpha_1 \cdots \alpha_{i-1}$ (resp., $\alpha'_1 \cdots \alpha'_{i'-1}$). In particular, for every $\omega \in \{0,1\}^{r(|x|)}$ and $\alpha \neq \alpha'$, if $(\omega,\alpha),(\omega,\alpha') \in V_x$, then $((\omega,\alpha),(\omega,\alpha')) \in E_x$.

1. Prove that the mapping $x \mapsto G_x$ can be computed in time that is polynomial in $2^{r(|x|)+q(|x|)} \cdot |x|$.

   (Note that the number of vertices in G_x equals $2^{r(|x|)+f(|x|)}$, where $f \leq q$ is the free-bit complexity of the PCP verifier.)

2. Prove that, for every x, the size of the maximum independent set in G_x is at most $2^{r(|x|)}$.

3. Prove that if $x \in S$ then G_x has an independent set of size $2^{r(|x|)}$.

4. Prove that if $x \notin S$ then the size of the maximum independent set in G_x is at most $2^{r(|x|)-1}$.

In general, denoting the PCP verifier by V, prove that the size of the maximum independent set in G_x is exactly $2^{r(|x|)} \cdot \max_\pi\{\mathsf{Pr}[V^\pi(x)=1]\}$. (Note the similarity to the proof of Proposition 2.25.)
Show that the PCP Theorem implies that *the size of the maximum independent set* (resp., clique) in a graph *is NP-hard to approximate to within any constant factor.*

Guideline: Note that an independent set in G_x corresponds to a set of coins R and a partial oracle π' such that V accepts x when using coins in R and accessing any oracle that is consistent with π'. The FGLSS reduction creates a gap of a factor of 2 between yes and no-instances of S (having a standard PCP). Larger factors can be obtained by considering a PCP that results from repeating the original PCP for a constant number of times. The result for `Clique` follows by considering the complement graph.

Exercise 9.15 Using the ideas of Exercise 9.14, prove that, for any $t(n) = o(\log n)$, it holds that $\mathcal{NP} \subseteq \mathcal{PCP}(t,t)$ implies that $\mathcal{P} = \mathcal{NP}$.

Guideline: We only use the fact that the said reduction reduces PCP to instances of the `Clique` problem (and ignore the fact that we actually get a stronger reduction to a "gapClique" problem). Furthermore, when applies to problems in $\mathcal{NP} \subseteq \mathcal{PCP}(t,t)$, this reduction runs in polynomial-time. The key observation is that this reduction maps instances of the `Clique` problem (which is in $\mathcal{NP} \subseteq \mathcal{PCP}(o(\log), o(\log))$) to shorter instances of the same problem (because $2^{o(\log n)} \ll n$). Thus, iteratively applying the reduction, we can reduce instances of `Clique` to instances of constant size. This yields a reduction of `Clique` to a finite set, and $\mathcal{NP} = \mathcal{P}$ follows (by the $\mathcal{NP}$-completeness of `Clique`).

Exercise 9.16 (a simple but partial analysis of the BLR Linearity Test) For Abelian groups G and H, consider functions from G to H. For such a (generic) function f, consider the linearity (or rather homomorphism) test that selects uniformly $r, s \in G$ and checks that $f(r)+f(s) = f(r+s)$. Let $\delta(f)$ denote the distance of f from the set of homomorphisms (of G to H); that is, $\delta(f)$ is the minimum taken over all homomorphisms $h : G \to H$ of $\mathsf{Pr}_{x\in G}[f(x) \neq h(x)]$. Using the following guidelines, prove that the probability that the test rejects f, denoted $\varepsilon(f)$, is at least $3\delta(f) - 6\delta(f)^2$.

1. Suppose that h is the homomorphism closest to f (i.e., $\delta(f) = \mathsf{Pr}[f(x) \neq h(x)]$). Prove that $\varepsilon(f) = \mathsf{Pr}_{x,y\in G}[f(x) + f(y) \neq f(x+y)]$ is lower-bounded by $3 \cdot \mathsf{Pr}_{x,y}[f(x) \neq h(x) \wedge f(y) = h(y) \wedge f(x+y) = h(x+y)]$.

   (Hint: consider three out of four disjoint cases that are possible when $f(x)+f(y) \neq f(x+y)$, where the three cases refer to the disagreement of h and f on exactly one out of the three relevant points.)

2. Prove that $\mathsf{Pr}_{x,y}[f(x) \neq h(x) \wedge f(y) = h(y) \wedge f(x+y) = h(x+y)] \geq \delta(f) - 2\delta(f)^2$.

   (Hint: lower-bound the said probability by $\mathsf{Pr}_{x,y}[f(x) \neq h(x)] - (\mathsf{Pr}_{x,y}[f(x) \neq h(x) \wedge f(y) \neq h(y)] + \mathsf{Pr}_{x,y}[f(x) \neq h(x) \wedge f(x+y) \neq h(x+y)])$.)

Note that the lower-bound $\varepsilon(f) \geq 3\delta(f) - 6\delta(f)^2$ increases with $\delta(f)$ only in the case that $\delta(f) \leq 1/4$. Furthermore, the lower-bound is useless in the case that $\delta(f) \geq 1/2$. Thus an alternative lower-bound is needed in case $\delta(f)$ approaches $1/2$ (or is larger than it); see Exercise 9.17.

Exercise 9.17 (a better analysis of the BLR Linearity Test (cf. [38])) In continuation to Exercise 9.16, use the following guidelines in order to prove that $\varepsilon(f) \geq \min(1/7, \delta(f)/2)$. Specifically, focusing on the case that $\varepsilon(f) < 1/7$, show that f is $2\varepsilon(f)$-close to some homomorphism (and thus $\varepsilon(f) \geq \delta(f)/2$).

1. Define the vote of y regarding the value of f at x as $\phi_y(x) \stackrel{\text{def}}{=} f(x+y) - f(y)$, and define $\phi(x)$ as the corresponding plurality vote (i.e., $\phi(x) \stackrel{\text{def}}{=} \mathrm{argmax}_{v\in H}\{|\{y \in G : \phi_y(x) = v\}|\}$).

   Prove that, for every $x \in G$, it holds that $\mathsf{Pr}_y[\phi_y(x) = \phi(x)] \geq 1 - 2\varepsilon(f)$.

Extra guideline: Fixing x, call a pair (y_1, y_2) good if $f(y_1) + f(y_2 - y_1) = f(y_2)$ and $f(x+y_1)+f(y_2-y_1) = f(x+y_2)$. Prove that, for any x, a random pair (y_1, y_2) is good with probability at least $1 - 2\varepsilon(f)$. On the other hand, for a good (y_1, y_2), it holds that $\phi_{y_1}(x) = \phi_{y_2}(x)$. Show that the graph in which *edges* correspond to good pairs must have a connected component of size at least $(1 - 2\varepsilon(f)) \cdot |G|$. Note that $\phi_y(x)$ is identical for all vertices y in this connected component, which in turn contains a majority of all y's in G.

2. Prove that ϕ is a homomorphism; that is, prove that, for every $x, y \in G$, it holds that $\phi(x) + \phi(y) = \phi(x + y)$.

   Extra guideline: Prove that $\phi(x) + \phi(y) = \phi(x + y)$ holds by considering the somewhat fictitious expression $\mathsf{Pr}_{r\in G}[\phi(x) + \phi(y) \neq \phi(x + y)]$, and showing that it is strictly smaller than 1 (and hence $\phi(x) + \phi(y) \neq \phi(x + y)$ is false). Upper-bound the probabilistic expression by

   $$\mathsf{Pr}_r[\phi(x)\neq f(x+r) - f(r) \vee \phi(y)\neq f(r) - f(r-y) \vee \phi(x+y)\neq f(x+r) - f(r-y)].$$

   Use the union bound (and Item 1), and note that $\mathsf{Pr}_r[\phi(x) \neq f(x + r) - f(r)] < 2\varepsilon(f) < 1/3$, whereas $\mathsf{Pr}_r[\phi(y) \neq f(r) - f(r - y)] = \mathsf{Pr}_{r'}[\phi(y) \neq f(y + r') - f(r')]$ and $\mathsf{Pr}_r[\phi(x + y) \neq f(x + r) - f(r - y)] = \mathsf{Pr}_{r'}[\phi(x + y) \neq f(x + y + r') - f(r')]$ (by substituting $r' = r - y$).

3. Prove that f is $2\varepsilon(f)$-close to ϕ.

   Extra guideline: Denoting $B = \{x\in G : \mathsf{Pr}_{y\in G}[f(x) \neq \phi_y(x)] \geq 1/2\}$, prove that $\varepsilon(f) \geq (1/2) \cdot (|B|/|G|)$. Note that if $x \in G \setminus B$ then $f(x) = \phi(x)$.

We comment that better bounds on the behavior of $\varepsilon(f)$ as a function of $\delta(f)$ are known.

Exercise 9.18 (checking matrix identity) Let M be a non-zero m-by-n matrix over GF(p). Prove that $\mathsf{Pr}_{r,s}[r^\top Ms \neq 0] \geq (1 - p^{-1})^2$, where r (resp., s) is a random m-ary (resp., n-ary) vector.

Guideline: Prove that if $v \neq 0^m$ then $\mathsf{Pr}_s[v^\top s = 0] = p^{-1}$, and that if M has rank ρ then $\mathsf{Pr}_r[r^\top M = 0^n] = p^{-\rho}$.

Exercise 9.19 (3SAT and CSP with two variables) Show that `3SAT` is reducible to $\mathtt{gapCSP}^{\{1,\ldots,7\}}_\tau$ for $\tau(m) = 1/m$, where `gapCSP` is as in Definition 9.18. Furthermore, show that the size of the resulting `gapCSP` instance is linear in the length of the input formula.

Guideline: Given an instance ψ of `3SAT`, consider the graph in which vertices correspond to clauses of ψ, edges correspond to pairs of clauses that share a variable, and the constraints represent the natural consistency condition regarding partial assignments that satisfy the clauses. See a similar construction in Exercise 9.14.

Exercise 9.20 (CSP with two Boolean variables) In contrast to Exercise 9.19, prove that for every positive function $\tau : \mathbb{N} \to (0, 1]$ the problem $\mathtt{gapCSP}^{\{0,1\}}_\tau$ is solvable in polynomial-time.

Guideline: Reduce $\mathtt{gapCSP}^{\{0,1\}}_\tau$ to 2SAT.

Exercise 9.21 Show that, for any fixed finite Σ and constant $c > 0$, the problem $\mathtt{gapCSP}_c^{\Sigma}$ is in $\mathcal{PCP}(\mathtt{log}, O(1))$.

Guideline: Consider an oracle that, for some satisfying assignment for the CSP-instance (G, Φ), provides a trivial encoding of the assignment; that is, for a satisfying assignment $\alpha : V \to \Sigma$, the oracle responds to the query (v, i) with the i^{th} bit in the binary representation of $\alpha(v)$. Consider a verifier that uniformly selects an edge (u, v) of G and checks the constraint $\phi_{(u,v)}$ when applied to the values $\alpha(u)$ and $\alpha(v)$ obtained from the oracle. This verifier makes $\log_2 |\Sigma|$ queries and reject each no-instance with probability at least c.

Exercise 9.22 For any constant Σ and $d \geq 14$, show that $\mathtt{gapCSP}^{\Sigma}$ can be reduced to itself such that the instance at the target of the reduction is a d-regular expander, and the fraction of violated constraints is preserved up to a constant factor. That is, the instance (G, Φ) is reduced to (G_1, Φ_1) such that G_1 is a d-regular expander graph and $\mathtt{vlt}(G_1, \Phi_1) = \Theta(\mathtt{vlt}(G, \Phi))$. Furthermore, make sure that $|G_1| = O(|G|)$ and that each vertex in G_1 has at least $d/2$ self-loops.

Guideline: First, replace each vertex of degree $d' > 3$ by a 3-regular expander of size d', and connect each of the original d' edges to a different vertex of this expander, thus obtaining a graph of maximum degree 4. Maintain the constraints associated with the original edges, and associate the equality constraint (i.e., $\phi(i, j) = 1$ if and only if $i = j$) to each new edge (residing in any of the added expanders). Next, denoting the number of vertices in the resulting graph by N_1, add to this graph a 3-regular expander of size N_1 (while associating with these edges the trivially satisfied constraint; i.e., $\phi(i, j) = 1$ for all $i, j \in \Sigma$). Finally, add at least $d/2$ self-loops to each vertex (using again trivially satisfied constraints), so to obtain a d-regular graph. Prove that this sequence of modifications may only decrease the fraction of violated constraints, and that the decrease is only by a constant factor. The latter assertion relies on the equality constraints associated with the small expanders used in the first step.

Exercise 9.23 (free bit complexity zero) Note that only sets in $\mathcal{BPP}$ have PCPs of *query* complexity zero. Furthermore, Exercise 9.12 implies that only sets in $\mathcal{P}$ have PCP systems of logarithmic randomness and *query* complexity zero.

1. Show that only sets in $\mathcal{P}$ have PCP systems of logarithmic randomness and *free-bit* complexity zero.

   (Hint: Consider an application of the FGLSS-reduction to a set having a PCP of free-bit complexity zero.)

2. In contrast, show that Graph Non-Isomorphism has a PCP system of *free-bit* complexity zero (and linear randomness complexity).

Exercise 9.24 (free bit complexity one) In continuation to Exercise 9.23, prove that only sets in $\mathcal{P}$ have PCP systems of logarithmic randomness and free-bit complexity one.

Guideline: Consider an application of the FGLSS-reduction to a set having a PCP of free-bit complexity one and randomness complexity r. Note that the question of whether the resulting graph has an independent set of size 2^r can be expressed as a 2CNF formula of size $\mathrm{poly}(2^r)$, and see Exercise 2.21.

Appendix C

On the Foundations of Modern Cryptography

It is possible to build a cabin with no foundations, but not a lasting building.

Eng. Isidor Goldreich (1906–1995)

Summary: Cryptography is concerned with the construction of computing systems that withstand any abuse: Such a system is constructed so to maintain a desired functionality, even under malicious attempts aimed at making it deviate from this functionality.

This appendix is aimed at presenting the foundations of cryptography, which are the paradigms, approaches and techniques used to conceptualize, define and provide solutions to natural security concerns. It presents some of these conceptual tools as well as some of the fundamental results obtained using them. The emphasis is on the clarification of fundamental concepts, and on demonstrating the feasibility of solving several central cryptographic problems. The presentation assumes basic knowledge of algorithms, probability theory and complexity theory, but nothing beyond this.

The appendix augments the treatment of one-way functions, pseudorandom generators and zero-knowledge proofs, which is given in Sections 7.1, 8.3 and 9.2, respectively. (These augmentations are important for cryptography, but are less central to the main context of this book and thus were omitted from the main text.) Using these basic tools, the appendix provides a treatment of basic cryptographic applications such as Encryption, Signatures, and General Cryptographic Protocols.

C.1 Introduction and Preliminaries

The vast expansion and rigorous treatment of cryptography is one of the major achievements of theoretical computer science. In particular, concepts such as computational indistinguishability, pseudorandomness and zero-knowledge interactive proofs were introduced, classical notions such as secure encryption and unforgeable signatures were placed on sound grounds, and new (unexpected) directions and connections were uncovered. Indeed, modern cryptography is strongly linked to complexity theory (in contrast to "classical" cryptography which is strongly related to information theory).

C.1.1 Modern cryptography

Modern cryptography is concerned with the construction of information systems that are robust against malicious attempts to make these systems deviate from their prescribed functionality. The prescribed functionality may be the private and authenticated communication of information through the Internet, the holding of incoercible and secret electronic voting, or conducting any "fault-resilient" multi-party computation. Indeed, the scope of modern cryptography is very broad, and it stands in contrast to "classical" cryptography (which has focused on the single problem of enabling secret communication over insecure communication media).

The design of cryptographic systems is a very difficult task. One cannot rely on intuitions regarding the "typical" state of the environment in which the system operates. For sure, the adversary attacking the system will try to manipulate the environment into "untypical" states. Nor can one be content with counter-measures designed to withstand specific attacks, since the adversary (which acts after the design of the system is completed) will try to attack the schemes in ways that are different from the ones the designer had envisioned. Although the validity of the foregoing assertions seems self-evident, still some people hope that in practice ignoring these tautologies will not result in actual damage. Experience shows that these hopes rarely come true; cryptographic schemes based on make-believe are broken, typically sooner than later.

In view of the foregoing, we believe that it makes little sense to make assumptions regarding the specific *strategy* that the adversary may use. The only assumptions that can be justified refer to the computational *abilities* of the adversary. Furthermore, the design of cryptographic systems has to be based on *firm foundations*; whereas ad-hoc approaches and heuristics are a very dangerous way to go. A heuristic may make sense when the designer has a very good idea regarding the environment in which a scheme is to operate, yet a cryptographic scheme has to operate in a maliciously selected environment that typically transcends the designer's view.

This appendix is aimed at presenting the foundations for cryptography. The foundations of cryptography are the paradigms, approaches and techniques used to conceptualize, define and provide solutions to natural "security concerns". Solving a cryptographic problem (or addressing a security concern) is a two-stage process consisting of a *definitional stage* and a *constructive stage*. First, in the definitional stage, the functionality underlying the natural concern is to be identified, and an adequate cryptographic problem has to be defined. Trying to list all undesired situations is infeasible and prone to error. Instead, one should define the functionality in terms of operation in an imaginary ideal model, and require a candidate solution to emulate this operation in the real, clearly defined, model (which specifies the adversary's abilities). Once the definitional stage is completed, one proceeds to construct a system that satisfies the definition. Such a construction may use some simpler tools, and its security is proved relying on the features of these tools. In practice, of course, such a scheme may need to satisfy also some *specific* efficiency requirements.

This appendix focuses on several archetypical cryptographic problems (e.g., encryption and signature schemes) and on several central tools (e.g., computational difficulty, pseudorandomness, and zero-knowledge proofs). For each of these problems (resp., tools), we start by presenting the natural concern underlying it (resp., its intuitive objective), then define the problem (resp., tool), and finally demonstrate that the problem may be solved (resp., the tool can be constructed). In the latter step, our focus is on demonstrating the feasibility of solving the problem, not on providing a practical solution.

Computational Difficulty

The aforementioned tools and applications (e.g., secure encryption) exist only if some sort of computational hardness exists. Specifically, all these problems and tools require (either explicitly or implicitly) the ability to generate instances of hard problems. Such ability is captured in the definition of one-way functions. Thus, one-way functions are the very minimum needed for doing most natural tasks of cryptography. (It turns out, as we shall see, that this necessary condition is "essentially" sufficient; that is, the existence of one-way functions (or augmentations and extensions of this assumption) suffices for doing most of cryptography.)

Our current state of understanding of efficient computation does not allow us to prove that one-way functions exist. In particular, if $\mathcal{P} = \mathcal{NP}$ then no one-way functions exist. Furthermore, the existence of one-way functions implies that $\mathcal{NP}$ is not contained in $\mathcal{BPP} \supseteq \mathcal{P}$ (not even "on the average"). Thus, proving that one-way functions exist is not easier than proving that $\mathcal{P} \neq \mathcal{NP}$; in fact, the former task seems significantly harder than the latter. Hence, we have no choice (at this stage of history) but to assume that one-way functions exist. As justification to this assumption we can only offer the combined beliefs of hundreds (or thousands) of researchers. Furthermore, these beliefs concern a simply stated assumption, and their validity follows from several widely believed conjectures which are central to various fields (e.g., the conjectured intractability of integer factorization is central to computational number theory).

Since we need assumptions anyhow, why not just assume what we want (i.e., the existence of a solution to some natural cryptographic problem)? Well, first we need to know what we want: as stated above, we must first clarify what exactly we want; that is, go through the typically complex definitional stage. But once this stage is completed, can we just assume that the definition derived can be met?

Not really: once a definition is derived, how can we know that it can at all be met? The way to demonstrate that a definition is viable (and that the corresponding intuitive security concern can be satisfied at all) is to construct a solution based on a *better understood* assumption (i.e., one that is more common and widely believed). For example, looking at the definition of zero-knowledge proofs, it is not a-priori clear that such proofs exist at all (in a non-trivial sense). The non-triviality of the notion was first demonstrated by presenting a zero-knowledge proof system for statements, regarding Quadratic Residuosity, which are believed to be hard to verify (without extra information). Furthermore, contrary to prior beliefs, it was later shown that the existence of one-way functions implies that any NP-statement can be proved in zero-knowledge. Thus, facts that were not known at all to hold (and even believed to be false), were shown to hold by reduction to widely believed assumptions (without which most of modern cryptography collapses anyhow). To summarize, not all assumptions are equal, and so reducing a complex, new and doubtful assumption to a widely-believed and simple (or even merely simpler) assumption is of great value. Furthermore, reducing the solution of a new task to the assumed security of a well-known primitive typically means providing a construction that, using the known primitive, solves the new task. This means that we do not only know (or assume) that the new task is solvable but we also have a solution based on a primitive that, being well-known, typically has several candidate implementations.

C.1.2 Preliminaries

Modern Cryptography, as surveyed here, is concerned with the construction of *efficient* schemes for which it is *infeasible* to violate the security feature. Thus, we need a notion of efficient computations as well as a notion of infeasible ones. The computations of the legitimate users of the scheme ought be efficient, whereas violating the security features (by an adversary) ought to be infeasible. We stress that we do not identify feasible computations with efficient ones, but rather view the former notion as potentially more liberal. Let us elaborate.

C.1.2.1 Efficient Computations and Infeasible ones

Efficient computations are commonly modeled by computations that are polynomial-time in the security parameter. The polynomial bounding the running-time of the legitimate user's strategy is *fixed and typically explicit* (and *small*). Indeed, our aim is to have a notion of efficiency that is as strict as possible (or, equivalently, develop strategies that are as efficient as possible). Here (i.e., when referring to the complexity of the legitimate users) we are in the same situation as in any algorithmic setting. Things are different when referring to our assumptions regarding the computational resources of the adversary, where we refer to the notion of feasible, which we wish to be as wide as possible. A common approach is to postulate that feasible computations are polynomial-time too, but here the polynomial is *not a-priori specified* (and is to be thought of as arbitrarily large). In other words, the adversary is restricted to the class of polynomial-time computations and anything beyond this is considered to be infeasible.

Although many definitions explicitly refer to the convention of associating feasible computations with polynomial-time ones, this convention is *inessential* to any of the results known in the area. In all cases, a more general statement can be made by referring to a general notion of feasibility, which should be preserved under standard algorithmic composition, yielding theories that refer to adversaries of running-time bounded by any specific super-polynomial function (or class of functions). Still, for sake of concreteness and clarity, we shall use the former convention in our formal definitions (but our motivational discussions will refer to an unspecified notion of feasibility that covers at least efficient computations).

C.1.2.2 Randomized (or probabilistic) Computations

Randomized computations play a central role in cryptography. One fundamental reason for this fact is that randomness is essential for the existence (or rather the generation) of secrets. Thus, we must allow the legitimate users to employ randomized computations, and certainly (since we consider randomization as feasible) we must consider also adversaries that employ randomized computations. This brings up the issue of success probability: typically, we require that legitimate users succeed (in fulfilling their legitimate goals) with probability 1 (or negligibly close to this), whereas adversaries succeed (in violating the security features) with negligible probability. Thus, the notion of a negligible probability plays an important role in our exposition.

One requirement of the definition of negligible probability is to provide a robust notion of rareness: A rare event should occur rarely even if we repeat the experiment for a feasible number of times. That is, in case we consider any polynomial-time computation to be feasible, a function $\mu : \mathbb{N} \to \mathbb{N}$ is called negligible if $1 - (1 - \mu(n))^{p(n)} < 0.01$ for every polynomial p and sufficiently big n (i.e., μ is negligible if for every positive polynomial p' the function $\mu(\cdot)$ is upper-bounded by $1/p'(\cdot)$).

We will also refer to the notion of noticeable probability. Here the requirement is that events that occur with noticeable probability, will occur almost surely (i.e., except with negligible probability) if we repeat the experiment for a polynomial number of times. Thus, a function $\nu : \mathbb{N} \to \mathbb{N}$ is called noticeable if for some positive polynomial p' the function $\nu(\cdot)$ is lower-bounded by $1/p'(\cdot)$.

C.1.3 Prerequisites, Organization, and Beyond

Our aim is to present the basic concepts, techniques and results in cryptography, and our emphasis is on the clarification of fundamental concepts and the relationship among them. This is done in a way independent of the particularities of some popular number theoretic examples. These particular examples played a central role in the development of the field and still offer the most practical implementations of all cryptographic primitives, but this does not mean that the presentation has to be linked to them. On the contrary, we believe that concepts are best clarified when presented at an abstract level, decoupled from specific implementations.

The appendix is organized in two main parts, corresponding to the Basic Tools of Cryptography and the Basic Applications of Cryptography.

The basic tools: The most basic tool is computational difficulty, which in turn is captured by the notion of one-way functions. Another notion of key importance is that of computational indistinguishability, underlying the theory of pseudorandomness as well as much of the rest of cryptography. Pseudorandom generators and functions are important tools that are frequently used. So are zero-knowledge proofs, playing a key role in the design of secure cryptographic protocols and in their study.

The basic applications: Encryption and signature schemes are the most basic applications of Cryptography. Their main utility is in providing secret and reliable communication over insecure communication media. Loosely speaking, encryption schemes are used for ensuring the secrecy (or privacy) of the actual information being communicated, whereas signature schemes are used to ensure its reliability (or authenticity). Another basic topic is the construction of secure cryptographic protocols for the implementation of arbitrary functionalities.

The presentation of the basic tools in Sections C.2–C.4 augments (and sometimes repeats parts of) Sections 7.1, 8.3, and 9.2 (which provide a basic treatment of one-way functions, pseudorandom generators, and zero-knowledge proofs, respectively). Sections C.5–C.7, provide a overview of the basic applications; that is, Encryption Schemes, Signature Schemes, and General Cryptographic Protocols.

Suggestions for further reading. This appendix is a brief summary of the author's two-volume work on the subject [87, 88]. Furthermore, the first part (i.e., Basic Tools) corresponds to [87], whereas the second part (i.e., Basic Applications) corresponds to [88]. Needless to say, the interested reader is referred to these textbooks for further detail (and, in particular, for missing references).

Practice. The aim of this appendix is to introduce the reader to the *theoretical foundations* of cryptography. As argued, such foundations are necessary for *sound* practice of cryptography. Indeed, practice requires more than theoretical foundations, whereas the current text makes no attempt to provide anything beyond the latter. However, given a sound foundation, one can learn and evaluate various practical suggestions that appear elsewhere. On the other hand, lack of sound foundations results in inability to critically evaluate practical suggestions, which in turn leads to unsound decisions. *Nothing could be more harmful to the design of schemes that need to withstand adversarial attacks than misconceptions about such attacks.*

C.2 Computational Difficulty

Modern Cryptography is concerned with the construction of systems that are easy to operate (properly) but hard to foil. Thus, a complexity gap (between the ease of proper usage and the difficulty of deviating from the prescribed functionality) lies at the heart of Modern Cryptography. However, gaps as required for Modern Cryptography are not known to exist; they are only widely believed to exist. Indeed, almost all of Modern Cryptography rises or falls with the question of whether one-way functions exist. We mention that the existence of one-way functions implies that $\mathcal{NP}$ contains search problems that are hard to solve *on the average*, which in turn implies that $\mathcal{NP}$ is not contained in $\mathcal{BPP}$ (i.e., a worst-case complexity conjecture).

Loosely speaking, one-way functions are functions that are easy to evaluate but hard (on the average) to invert. Such functions can be thought of as an efficient way of generating "puzzles" that are infeasible to solve (i.e., the puzzle is a random image of the function and a solution is a corresponding preimage). Furthermore, the person generating the puzzle knows a solution to it and can efficiently verify the validity of (possibly other) solutions to the puzzle. Thus, one-way functions have, by definition, a clear cryptographic flavor (i.e., they manifest a gap between the ease of one task and the difficulty of a related one).

C.2.1 One-Way Functions

We start by reproducing the basic definition of one-way functions as appearing in Section 7.1.1, where this definition is further discussed.

Definition C.1 (one-way functions, Definition 7.1 restated): *A function* $f:\{0,1\}^* \to \{0,1\}^*$ *is called* one-way *if the following two conditions hold:*

1. easy to evaluate: *There exist a polynomial-time algorithm A such that $A(x) = f(x)$ for every $x \in \{0,1\}^*$.*
2. hard to invert: *For every probabilistic polynomial-time algorithm A', every polynomial p, and all sufficiently large n,*

$$\mathsf{Pr}[A'(f(x), 1^n) \in f^{-1}(f(x))] \; < \; \frac{1}{p(n)}$$

   where the probability is taken uniformly over $x \in \{0,1\}^n$ and all the internal coin tosses of algorithm A'.

Some of the most popular candidates for one-way functions are based on the conjectured intractability of computational problems in number theory. One such conjecture is that it is infeasible to factor large integers. Consequently, the function that takes as input two (equal length) primes and outputs their product is widely believed to be a one-way function. Furthermore, factoring such a composite is infeasible if and only if squaring modulo such a composite is a one-way function (see [171]). For certain composites (i.e., products of two primes that are both congruent to 3 mod 4), the latter function induces a permutation over the set of quadratic residues modulo this composite. A related permutation, which is widely believed to be one-way, is the RSA function [181]: $x \mapsto x^e \bmod N$, where $N = P \cdot Q$ is a composite as above, e is relatively prime to $(P-1) \cdot (Q-1)$, and

$x \in \{0, ..., N-1\}$. The latter examples (as well as other popular suggestions) are better captured by the following formulation of a collection of one-way functions (which is indeed related to Definition C.1):

Definition C.2 (collections of one-way functions): *A collection of functions, $\{f_i : D_i \to \{0,1\}^*\}_{i \in \overline{I}}$, is called* one-way *if there exists three probabilistic polynomial-time algorithms, I, D and F, such that the following two conditions hold:*

1. easy to sample and compute: *On input 1^n, the output of* (the index selection) *algorithm I is distributed over the set $\overline{I} \cap \{0,1\}^n$ (i.e., is an n-bit long index of some function). On input (an index of a function) $i \in \overline{I}$, the output of* (the domain sampling) *algorithm D is distributed over the set D_i (i.e., over the domain of the function). On input $i \in \overline{I}$ and $x \in D_i$,* (the evaluation) *algorithm F* always *outputs $f_i(x)$.*

2. hard to invert:[1] *For every probabilistic polynomial-time algorithm, A', every positive polynomial $p(\cdot)$, and all sufficiently large n's*

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

where $i \leftarrow I(1^n)$ and $x \leftarrow D(i)$.

The collection is said to be a collection of permutations *if each of the f_i's is a permutation over the corresponding D_i, and $D(i)$ is almost uniformly distributed in D_i.*

For example, in case of the RSA, one considers $f_{N,e} : D_{N,e} \to D_{N,e}$ that satisfies $f_{N,e}(x) = x^e \bmod N$, where $D_{N,e} = \{0, ..., N-1\}$. Definition C.2 is also a good starting point for the definition of a trapdoor permutation.[2] Loosely speaking, the latter is a collection of one-way permutations augmented with an efficient algorithm that allows for inverting the permutation when given adequate auxiliary information (called a trapdoor).

Definition C.3 (trapdoor permutations): *A collection of permutations as in Definition C.2 is called a* trapdoor permutation *if there are two auxiliary probabilistic polynomial-time algorithms I' and F^{-1} such that (1) the distribution $I'(1^n)$ ranges over pairs of strings so that the first string is distributed as in $I(1^n)$, and (2) for every (i,t) in the range of $I'(1^n)$ and every $x \in D_i$ it holds that $F^{-1}(t, f_i(x)) = x$.* (That is, t is a trapdoor that allows to invert f_i.)

For example, in case of the RSA, $f_{N,e}$ can be inverted by raising to the power d (modulo $N = P \cdot Q$), where d is the multiplicative inverse of e modulo $(P-1) \cdot (Q-1)$. Indeed, in this case, the trapdoor information is (N, d).

[1] Note that this condition refers to the distributions $I(1^n)$ and $D(i)$, which are merely required to range over $\overline{I} \cap \{0,1\}^n$ and D_i, respectively. (Typically, the distributions $I(1^n)$ and $D(i)$ are (almost) uniform over $\overline{I} \cap \{0,1\}^n$ and D_i, respectively.)

[2] Indeed, a more adequate term would be a collection of trapdoor permutations, but the shorter (and less precise) term is the commonly used one.

Strong versus weak one-way functions (summary of Section 7.1.2). Recall that the foregoing definitions require that any feasible algorithm *succeeds in inverting* the function *with negligible probability.* A weaker notion only requires that any feasible algorithm *fails to invert* the function *with noticeable probability.* It turns out that the existence of such weak one-way functions implies the existence of strong one-way functions (as in Definition C.1). The construction itself is straightforward, but analyzing it transcends the analogous information theoretic setting. Instead, the security (i.e., hardness of inverting) the resulting construction is proved via a so called "reducibility argument" that transforms the violation of the conclusion (i.e., the security of the resulting construction) into a violation of the hypothesis (i.e., the security of the given primitive). This strategy (i.e., a "reducibility argument") is used to prove all conditional results in the area.

C.2.2 Hard-Core Predicates

Recall that saying that a function f is one-way implies that given y (in the range of f) it is infeasible to find a preimage of y under f. This does not mean that it is infeasible to find out partial information about the preimage(s) of y under f. Specifically it may be easy to retrieve half of the bits of the preimage (e.g., given a one-way function f consider the function g defined by $g(x,r) \stackrel{\text{def}}{=} (f(x),r)$, for every $|x|=|r|$). As will become clear in subsequent sections, hiding partial information (about the function's preimage) plays an important role in more advanced constructs (e.g., secure encryption). This partial information can be considered as a "hard core" of the difficulty of inverting f. Loosely speaking, a *polynomial-time computable* (Boolean) predicate b, is called a hard-core of a function f if no feasible algorithm, given $f(x)$, can guess $b(x)$ with success probability that is non-negligibly better than one half. The actual definition is presented in Section 7.1.3 (i.e., Definition 7.6).

Note that if b is a hard-core of a 1-1 function f that is polynomial-time computable then f is a one-way function. On the other hand, recall that Theorem 7.7 asserts that *for any one-way function f, the inner-product mod 2 of x and r is a hard-core of $f'(x,r) = (f(x),r)$.*

C.3 Pseudorandomness

In practice "pseudorandom" sequences are often used instead of truly random sequences. The underlying belief is that if an (efficient) application performs well when using a truly random sequence then it will perform essentially as well when using a "pseudorandom" sequence. However, this belief is not supported by ad-hoc notions of "pseudorandomness" such as passing the statistical tests in [137] or having large "linear-complexity" (as defined in [108]). Needless to say, using such "pseudorandom" sequences (instead of truly random sequences) in a cryptographic application is very dangerous.

In contrast, truly random sequences can be safely replaced by pseudorandom sequences provided that pseudorandom distributions are defined as being computationally indistinguishable from the uniform distribution. Such a definition makes the soundness of this replacement an easy corollary. Loosely speaking, pseudorandom generators are then defined as efficient procedures for creating long pseudorandom sequences based on few truly random bits (i.e., a short random seed). The relevance of such constructs to cryptography is in providing legitimate users that share short random seeds a method for creating long sequences that look random to any feasible adversary (which does not know the said seed).

C.3.1 Computational Indistinguishability

A central notion in Modern Cryptography is that of "effective similarity" (a.k.a computational indistinguishability; cf. [104, 223]). The underlying thesis is that we do not care whether or not objects are equal, all we care about is whether or not a difference between the objects can be observed by a feasible computation. In case the answer is negative, the two objects are equivalent as far as any practical application is concerned. Indeed, in the sequel we will often interchange such (computationally indistinguishable) objects. In this section we recall the definition of computational indistinguishability (presented in Section 8.3.3), and consider two variants.

Definition C.4 (computational indistinguishability, Definition 8.4 revised[3]): *We say that* $X = \{X_n\}_{n\in\mathbb{N}}$ *and* $Y = \{Y_n\}_{n\in\mathbb{N}}$ *are* computationally indistinguishable *if for every probabilistic polynomial-time algorithm* D *every polynomial* p, *and all sufficiently large* n,

$$|\mathsf{Pr}[D(1^n, X_n)=1] - \mathsf{Pr}[D(1^n, Y_n)=1]| \;<\; \frac{1}{p(n)}$$

where the probabilities are taken over the relevant distribution (i.e., either X_n or Y_n) *and over the internal coin tosses of algorithm* D.

See further discussion in Section 8.3.3. In particular, recall that for "efficiently constructible" distributions, indistinguishability by a single sample (as defined above) implies indistinguishability by multiple samples (as in Definition 8.5).

Extension to ensembles indexed by strings. Here we refer to a natural extension of Definition C.4: Rather than referring to ensembles indexed by $\mathbb{N}$, we refer to ensembles indexed by an arbitrary set $S \subseteq \{0,1\}^*$. Typically, for an ensemble $\{Z_\alpha\}_{\alpha\in S}$, it holds that Z_α ranges over strings of length that is polynomially-related to the length of α.

---

[3]For sake of streamlining Definition C.4 with Definition C.5 (and unlike in Definition 8.4), here the distinguisher is explicitly given the index n of the distribution that it inspects. (In typical applications, the difference between Definitions 8.4 and C.4 is immaterial because the index n is easily determined from any sample of the corresponding distributions.)

Definition C.5 *We say that* $\{X_\alpha\}_{\alpha\in S}$ *and* $\{Y_\alpha\}_{\alpha\in S}$ *are* computationally indistinguishable *if for every probabilistic polynomial-time algorithm* D *every polynomial* p, *and all sufficiently long* $\alpha \in S$,

$$|\mathsf{Pr}[D(\alpha, X_\alpha)=1] - \mathsf{Pr}[D(\alpha, Y_\alpha)=1]| \;<\; \frac{1}{p(|\alpha|)}$$

where the probabilities are taken over the relevant distribution (i.e., either X_α *or* Y_α*) and over the internal coin tosses of algorithm* D.

Note that Definition C.4 is obtained as a special case by setting $S = \{1^n : n \in \mathbb{N}\}$.

A non-uniform version. A non-uniform definition of computational indistinguishability can be derived from Definition C.5 by artificially augmenting the indices of the distributions. That is, $\{X_\alpha\}_{\alpha\in S}$ and $\{Y_\alpha\}_{\alpha\in S}$ are computationally indistinguishable in a non-uniform sense if for every polynomial p the ensembles $\{X'_{\alpha'}\}_{\alpha'\in S'}$ and $\{Y'_{\alpha'}\}_{\alpha'\in S'}$ are computationally indistinguishable (as in Definition C.5), where $S' = \{\alpha\beta : \alpha \in S \wedge \beta \in \{0,1\}^{p(|\alpha|)}\}$ and $X'_{\alpha\beta} = X_\alpha$ (resp., $Y'_{\alpha\beta} = Y_\alpha$) for every $\beta \in \{0,1\}^{p(|\alpha|)}$. An equivalent (alternative) definition can be obtained by following the formulation that underlies Definition 8.12.

C.3.2 Pseudorandom Generators

Loosely speaking, a pseudorandom generator is an efficient (deterministic) algorithm that on input a short random *seed* outputs a (typically much) longer sequence that is computationally indistinguishable from a uniformly chosen sequence.

Definition C.6 (pseudorandom generator, Definition 8.1 restated): *Let* $\ell:\mathbb{N}\to\mathbb{N}$ *satisfy* $\ell(n) > n$, *for all* $n \in \mathbb{N}$. *A* pseudorandom generator, *with* stretch function ℓ, *is a* (deterministic) *polynomial-time algorithm* G *satisfying the following:*

1. *For every* $s \in \{0,1\}^*$, *it holds that* $|G(s)| = \ell(|s|)$.
2. $\{G(U_n)\}_{n\in\mathbb{N}}$ *and* $\{U_{\ell(n)}\}_{n\in\mathbb{N}}$ *are computationally indistinguishable, where* U_m *denotes the uniform distribution over* $\{0,1\}^m$.

Indeed, the probability ensemble $\{G(U_n)\}_{n\in\mathbb{N}}$ *is called* pseudorandom.

We stress that pseudorandom sequences can replace truly random sequences not only in "standard" algorithmic applications but also in cryptographic ones. That is, *any* cryptographic application that is secure when the legitimate parties use truly random sequences, is also secure when the legitimate parties use pseudorandom sequences. The benefit in such a substitution (of random sequences by pseudorandom ones) is that the latter sequences can be efficiently generated using much less true randomness. Furthermore, *in an interactive setting*, it is possible to eliminate all random steps from the on-line execution of a program, by replacing them with the generation of pseudorandom bits based on a random seed selected and fixed off-line (or at set-up time). This allows interactive parties to generate a long sequence of common secret bits based on a shared random seed which may have been selected at a much earlier time.

Various cryptographic applications of pseudorandom generators will be presented in the sequel, but let us first recall that *pseudorandom generators exist if and only if one-way functions exist* (see Theorem 8.11). For further treatment of pseudorandom generators, the reader is referred to Section 8.3.

C.3.3 Pseudorandom Functions

Pseudorandom generators provide a way to efficiently generate long pseudorandom sequences from short random seeds. Pseudorandom functions, introduced and constructed by Goldreich, Goldwasser, and Micali [91], are even more powerful: they provide efficient direct access to the bits of a huge pseudorandom sequence (which is not feasible to scan bit-by-bit). More precisely, a pseudorandom function is an efficient (deterministic) algorithm that given an n-bit *seed*, s, and an n-bit *argument*, x, returns an n-bit string, denoted $f_s(x)$, such that it is infeasible to distinguish the values of f_s, for a uniformly chosen $s \in \{0,1\}^n$, from the values of a truly random function $F : \{0,1\}^n \to \{0,1\}^n$. That is, the (feasible) testing procedure is given oracle access to the function (but not its explicit description), and cannot distinguish the case it is given oracle access to a pseudorandom function from the case it is given oracle access to a truly random function.

Definition C.7 (pseudorandom functions): *A* pseudorandom function (ensemble), *is a collection of functions* $\{f_s:\{0,1\}^{|s|}\to\{0,1\}^{|s|}\}_{s\in\{0,1\}^*}$ *that satisfies the following two conditions:*

1. (efficient evaluation) *There exists an efficient* (deterministic) *algorithm that given a* seed, s, *and an* argument, $x \in \{0,1\}^{|s|}$, *returns* $f_s(x)$.
2. (pseudorandomness) *For every probabilistic polynomial-time oracle machine,* M, *every positive polynomial* p *and all sufficiently large* n*'s*

$$\left|\,\mathsf{Pr}[M^{f_{U_n}}(1^n)=1] - \mathsf{Pr}[M^{F_n}(1^n)=1]\,\right| \;<\; \frac{1}{p(n)}$$

where F_n *denotes a uniformly selected function mapping* $\{0,1\}^n$ *to* $\{0,1\}^n$.

One key feature of the foregoing definition is that pseudorandom functions can be generated and shared by merely generating and sharing their seed; that is, a "random looking" function $f_s : \{0,1\}^n \to \{0,1\}^n$, is determined by its n-bit seed s. Parties wishing to share a "random looking" function f_s (determining 2^n-many values), merely need to generate and share among themselves the n-bit seed s. (For example, one party may randomly select the seed s, and communicate it, via a secure channel, to all other parties.) Sharing a pseudorandom function allows parties to determine (by themselves and without any further communication) random-looking values depending on their current views of the environment (which need not be known a priori). To appreciate the potential of this tool, one should realize that sharing a pseudorandom function is essentially as good as being able to agree, on the fly, on the association of random values to (on-line) given values, where the latter are taken from a huge set of possible values. We stress that this agreement is achieved without communication and synchronization: Whenever some party needs to associate a random value to a given value, $v \in \{0,1\}^n$, it will associate to v the (same) random value $r_v \in \{0,1\}^n$ (by setting $r_v = f_s(v)$, where f_s is a pseudorandom function agreed upon beforehand). Concrete applications of (this power of) pseudorandom functions appear in Sections C.5.2 and C.6.2.

Theorem C.8 (How to construct pseudorandom functions): *Pseudorandom functions can be constructed using any pseudorandom generator.*

Proof Sketch:[4] Let G be a pseudorandom generator that stretches its seed by a factor of two (i.e., $\ell(n) = 2n$), and let $G_0(s)$ (resp., $G_1(s)$) denote the first (resp., last) $|s|$ bits in $G(s)$. Define

$$G_{\sigma_{|s|}\cdots\sigma_2\sigma_1}(s) \stackrel{\rm def}{=} G_{\sigma_{|s|}}(\cdots G_{\sigma_2}(G_{\sigma_1}(s))\cdots).$$

We consider the function ensemble $\{f_s : \{0,1\}^{|s|} \to \{0,1\}^{|s|}\}_{s\in\{0,1\}^*}$, where $f_s(x) \stackrel{\rm def}{=} G_x(s)$. Pictorially, the function f_s is defined by n-step walks down a full binary tree of depth n having labels at the vertices. The root of the tree, hereafter referred to as the level 0 vertex of the tree, is labeled by the string s. If an internal vertex is labeled r then its left child is labeled $G_0(r)$ whereas its right child is labeled $G_1(r)$. The value of $f_s(x)$ is the string residing in the leaf reachable from the root by a path corresponding to the string x.

We claim that this function ensemble $\{f_s\}_{s\in\{0,1\}^*}$ is pseudorandom. The proof uses the hybrid technique (cf. Section 8.3.3): The $i^{\rm th}$ hybrid, H_n^i, is a function ensemble consisting of $2^{2^i \cdot n}$ functions $\{0,1\}^n \to \{0,1\}^n$, each determined by 2^i random n-bit strings, denoted $\overline{s} = \langle s_\beta \rangle_{\beta\in\{0,1\}^i}$. The value of such function $h_{\overline{s}}$ at $x = \alpha\beta$, where $|\beta| = i$, is defined to equal $G_\alpha(s_\beta)$. (Pictorially, the function $h_{\overline{s}}$ is defined by placing the strings in $\overline{s}$ in the corresponding vertices of level i, and labeling vertices of lower levels using the very rule used in the definition of f_s.) The extreme hybrids correspond to our indistinguishability claim (i.e., $H_n^0 \equiv f_{U_n}$ and H_n^n is a truly random function), and neighboring hybrids can be related to our indistinguishability hypothesis (specifically, to the indistinguishability of $G(U_n)$ and U_{2n} under multiple samples). $\square$

Variants. Useful variants (and generalizations) of the notion of pseudorandom functions include Boolean pseudorandom functions that are defined over all strings (i.e., $f_s : \{0,1\}^* \to \{0,1\}$) and pseudorandom functions that are defined for other domains and ranges (i.e., $f_s : \{0,1\}^{d(|s|)} \to \{0,1\}^{r(|s|)}$, for arbitrary polynomially bounded functions $d, r : \mathbb{N} \to \mathbb{N}$). Various transformations between these variants are known (cf. [87, Sec. 3.6.4] and [88, Apdx. C.2]).

[4] See details in [87, Sec. 3.6.2].

Applications and a generic methodology. Pseudorandom functions are a very useful cryptographic tool: One may first design a cryptographic scheme assuming that the legitimate users have black-box access to a random function, and next implement the random function using a pseudorandom function. The usefulness of this tool stems from the fact that having (black-box) access to a random function gives the legitimate parties a potential advantage over the adversary (which does not have free access to this function).[5] The security of the resulting implementation (which uses a pseudorandom function) is established in two steps: First one proves the security of an idealized scheme that uses a truly random function, and next one argues that the actual implementation (which uses a pseudorandom function) is secure (as otherwise one obtains an efficient oracle machine that distinguishes a pseudorandom function from a truly random one).

C.4 Zero-Knowledge

Zero-knowledge proofs provide a powerful tool for the design of cryptographic protocols as well as a good bench-mark for the study of various issues regarding such protocols. Loosely speaking, zero-knowledge proofs are proofs that yield nothing beyond the validity of the assertion. That is, a verifier obtaining such a proof only gains conviction in the validity of the assertion (as if it was told by a trusted party that the assertion holds). This is formulated by saying that anything that is feasibly computable from a zero-knowledge proof is also feasibly computable from the (valid) assertion itself. The latter formulation follows the simulation paradigm, which is discussed next, while reproducing part of the discussion in §9.2.1.1 and making additional comments regarding the use of this paradigm in cryptography.

C.4.1 The Simulation Paradigm

A key question regarding the modeling of security concerns is how to express the intuitive requirement that an adversary "gains nothing substantial" by deviating from the prescribed behavior of an honest user. Our approach is that the adversary *gains nothing* if whatever it can obtain by unrestricted adversarial behavior can also be obtained within essentially the same computational effort by a benign behavior. The definition of the "benign behavior" captures what we want to achieve in terms of security, and is specific to the security concern to be addressed. For example, in the context of zero-knowledge the unrestricted adversarial behavior is captured by an arbitrary probabilistic polynomial-time verifier strategy, whereas the benign behavior is any computation that is based (only) on the assertion itself (while assuming that the latter is valid). Other examples are discussed in Sections C.5.1 and C.7.1.

A notable property of the simulation paradigm, as well as of the entire definitional approach surveyed here, is that this approach is overly liberal with respect to

[5]The aforementioned methodology is sound provided that the adversary does not get the description of the pseudorandom function (i.e., the seed) in use, but has only (possibly limited) oracle access to it. This is different from the so-called Random Oracle Methodology.

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

C.4.2 The Actual Definition

In §9.2.1.2 zero-knowledge was defined as a property of some prover strategies (within the context of interactive proof systems, as defined in Section 9.1.1). More generally, the term may apply to any interactive machine, regardless of its goal. A strategy A is zero-knowledge on (inputs from) the set S if, for every feasible strategy B^*, there exists a feasible computation C^* such that the following two probability ensembles are computationally indistinguishable (according to Definition C.5):

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

2. $\{C^*(x)\}_{x\in S} \stackrel{\rm def}{=}$ the output of C^* on input $x \in S$.

Recall that the first ensemble represents an actual execution of an interactive protocol, whereas the second ensemble represents the computation of a stand-alone procedure (called the "simulator"), which does not interact with anybody.

The foregoing definition does *not* account for auxiliary information that an adversary B^* may have prior to entering the interaction. Accounting for such auxiliary information is essential for using zero-knowledge proofs as subprotocols inside larger protocols. This is taken care of by a stricter notion called auxiliary-input zero-knowledge, which was not presented in Section 9.2.

Definition C.9 (zero-knowledge, revisited): *A strategy A is* auxiliary-input zero-knowledge *on inputs from S if, for every probabilistic polynomial-time strategy B^* and every polynomial p, there exists a probabilistic polynomial-time algorithm C^* such that the following two probability ensembles are computationally indistinguishable:*

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

2. $\{C^*(x,z)\}_{x\in S\,,\,z\in\{0,1\}^{p(|x|)}} \stackrel{\rm def}{=}$ *the output of C^* on inputs $x \in S$ and $z \in \{0,1\}^{p(|x|)}$.*

Almost all known zero-knowledge proofs are in fact auxiliary-input zero-knowledge. As hinted, *auxiliary-input zero-knowledge is preserved under sequential composition.* A simulator for the multiple-session protocol can be constructed by iteratively invoking the single-session simulator that refers to the residual strategy of the adversarial verifier in the given session (while feeding this simulator with the transcript of previous sessions). Indeed, the residual single-session verifier gets the transcript of the previous sessions as part of its auxiliary input (i.e., z in Definition C.9). For details, see [87, Sec. 4.3.4].

C.4.3 A construction and a generic application

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

Assuming the existence of "commitment schemes" (cf. §C.4.3.1), which in turn exist if one-way functions exist [158, 113], *there exist* (auxiliary-input) *zero-knowledge proofs of membership in any NP-set.* These zero-knowledge proofs, abstractly depicted in Construction 9.10, have the following important property: the prescribed prover strategy is efficient, provided it is given as auxiliary-input an NP-witness to the assertion (to be proved).[6] Indeed, by using the standard Karp-reductions to 3-Colorability, the protocol of Construction 9.10 can be used for obtaining zero-knowledge proofs for any set in $\mathcal{NP}$. Implementing the abstract boxes (referred to in Construction 9.10) by commitment schemes, we get:

Theorem C.10 (On the applicability of zero-knowledge proofs): *If* (non-uniformly hard) *one-way functions exist then every set $S \in \mathcal{NP}$ has an auxiliary-input zero-knowledge interactive proof. Furthermore, the prescribed prover strategy can be implemented in probabilistic polynomial-time, provided it is given as auxiliary-input an NP-witness for membership of the common input in S.*

Theorem C.10 makes zero-knowledge a very powerful tool in the design of cryptographic schemes and protocols (see §C.4.3.3). We comment that the intractability assumption used in Theorem C.10 seems essential.

C.4.3.1 Commitment schemes

Loosely speaking, commitment schemes are two-stage (two-party) protocols allowing for one party to commit itself (at the first stage) to a value while keeping the value secret. In a (second) latter stage, the commitment is "opened" and it is guaranteed that the "opening" can yield only a single value, which is determined

[6] The auxiliary-input given to the prescribed prover (in order to allow for an efficient implementation of its strategy) is not to be confused with the auxiliary-input that is given to malicious verifiers (in the definition of auxiliary-input zero-knowledge). The former is typically an NP-witness for the common input, which is available to the user that invokes the prover strategy (cf. the generic application discussed in §C.4.3.3). In contrast, the auxiliary-input that is given to malicious verifiers models arbitrary partial information that may be available to the adversary.

during the committing phase. Thus, the (first stage of the) commitment scheme is both *binding* and *hiding*.

A simple (uni-directional communication) commitment scheme can be constructed based on any one-way 1-1 function f (with a corresponding hard-core b). To commit to a bit σ, the sender uniformly selects $s \in \{0,1\}^n$, and sends the pair $(f(s), b(s) \oplus \sigma)$. Note that this is both binding and hiding. An alternative construction, which can be based on any one-way function, uses a pseudorandom generator G that stretches its seed by a factor of three (cf. Theorem 8.11). A commitment is established, via two-way communication, as follows (cf. [158]): The receiver selects uniformly $r \in \{0,1\}^{3n}$ and sends it to the sender, which selects uniformly $s \in \{0,1\}^n$ and sends $r \oplus G(s)$ if it wishes to commit to the value one and $G(s)$ if it wishes to commit to zero. To see that this is binding, observe that there are at most 2^{2n} "bad" values r that satisfy $G(s_0) = r \oplus G(s_1)$ for some pair (s_0, s_1), and with overwhelmingly high probability the receiver will not pick one of these bad values. The hiding property follows by the pseudorandomness of G.

C.4.3.2 Efficiency considerations

The number of rounds in a protocol is commonly considered the most important efficiency criterion (or complexity measure), and typically one desires to have it be a constant. However, in order to obtain negligible soundness error, the protocol of Construction 9.10 has to be invoked for a non-constant number of times (and the analysis of the resulting protocol relies on the preservation of zero-knowledge under sequential composition). At first glance, it seems that one can derive a constant-round zero-knowledge proof system (of negligible soundness error) by performing these invocations in parallel (rather than sequentially). Unfortunately, it is not clear that the resulting interactive proof is zero-knowledge. Still, under standard intractability assumptions (e.g., the intractability of factoring), constant-round zero-knowledge proofs (of negligible soundness error) do exist for every set in $\mathcal{NP}$.

C.4.3.3 A generic application

As mentioned, Theorem C.10 makes zero-knowledge a very powerful tool in the design of cryptographic schemes and protocols. This wide applicability is due to two important aspects regarding Theorem C.10: Firstly, Theorem C.10 provides a zero-knowledge proof for every NP-set, and secondly the prescribed prover can be implemented in probabilistic polynomial-time when given an adequate NP-witness. We now turn to a typical application of zero-knowledge proofs.

In a typical cryptographic setting, a user U has a secret and is supposed to take some action based on its secret. The question is how can other users verify that U indeed took the correct action (as determined by U's secret and publicly known information). Indeed, if U discloses its secret then anybody can verify that U took the correct action. However, U does not want to reveal its secret. Using zero-knowledge proofs we can satisfy both conflicting requirements (i.e., having other users verify that U took the correct action without violating U's interest in not revealing its secret). That is, U can prove in zero-knowledge that it took the correct action. Note that U's claim to having taken the correct action is an NP-assertion (since U's legal action is determined as a polynomial-time function of its secret and the public information), and that U has an NP-witness to its validity (i.e., the secret is an NP-witness to the claim that the action fits the public information). Thus, by Theorem C.10, it is possible for U to efficiently prove the correctness of its action without yielding anything about its secret. Consequently, it is fair to ask U to prove (in zero-knowledge) that it behaves properly, and so to force U to behave properly. Indeed, "forcing proper behavior" is the canonical application of zero-knowledge proofs (see §C.7.3.2).

This paradigm (i.e., "forcing proper behavior" via zero-knowledge proofs), which in turn is based on Theorem C.10, has been utilized in numerous different settings. Indeed, this paradigm is the basis for the wide applicability of zero-knowledge protocols in Cryptography.

C.4.4 Variants and Issues

In this section we consider numerous variants on the notion of zero-knowledge and the underlying model of interactive proofs. These include black-box simulation and other variants of zero-knowledge (cf. Section C.4.4.1), as well as notions such as proofs of knowledge, non-interactive zero-knowledge, and witness indistinguishable proofs (cf. Section C.4.4.2).

Before starting, we call the reader's attention to the notion of computational soundness and to the related notion of argument systems, discussed in §9.1.4.2. We mention that argument systems may be more efficient than interactive proofs as well as provide stronger zero-knowledge guarantees. Specifically, perfect zero-knowledge arguments for $\mathcal{NP}$ can be constructed based on some reasonable assumptions [46], where perfect zero-knowledge means that the simulator's output is distributed *identically* to the verifier's view in the real interaction (see Definition 9.7 or a discussion in §C.4.4.1). Note that stronger security guarantee for the prover (as provided by perfect zero-knowledge) comes at the cost of weaker security guarantee for the verifier (as provided by computational soundness). The answer to the question of whether or not this trade-off is worthwhile seems to be application dependent, and one should also take into account the availability and complexity of the corresponding protocols.

C.4.4.1 Definitional variations

We consider several definitional issues regarding the notion of zero-knowledge (as defined in Definition C.9).

Universal and black-box simulation. A strengthening of Definition C.9 is obtained by requiring the existence of a universal simulator, denoted $\mathcal{C}$, that can simulate (the interactive gain of) any verifier strategy B^* when given the verifier's program an auxiliary-input; that is, in terms of Definition C.9, one should replace $C^*(x,z)$ by $\mathcal{C}(x,z,\langle B^*\rangle)$, where $\langle B^*\rangle$ denotes the description of the program of B^*

(which may depend on x and on z). That is, we effectively restrict the simulation by requiring that it be a uniform (feasible) function of the verifier's program (rather than arbitrarily depend on it). This restriction is very natural, because it seems hard to envision an alternative way of establishing the zero-knowledge property of a given protocol. Taking another step, one may argue that since it seems infeasible to reverse-engineer programs, the simulator may as well just use the verifier strategy as an oracle (or as a "black-box"). This reasoning gave rise to the notion of black-box simulation, which was introduced and advocated in [94] and further studied in numerous works. The belief was that inherent limitations regarding black-box simulation represent inherent limitations of zero-knowledge itself. For example, it was believed that the *fact* that the parallel version of the interactive proof of Construction 9.10 cannot be simulated in a black-box manner (unless $\mathcal{NP}$ is contained in $\mathcal{BPP}$) *implies* that this version is not zero-knowledge (as per Definition C.9 itself). However, the (underlying) belief that *any* zero-knowledge protocol can be simulated in a black-box manner was refuted recently by Barak [22].

Honest verifier versus general cheating verifier. Definition C.9 refers to all feasible verifier strategies, which is most natural in the cryptographic setting because zero-knowledge is supposed to capture the robustness of the prover under *any feasible* (i.e., adversarial) attempt to gain something by interacting with it. A weaker and still interesting notion of zero-knowledge refers to what can be gained by an "honest verifier" (or rather a semi-honest verifier)[7] that interacts with the prover as directed, with the exception that it may maintain (and output) a record of the entire interaction (i.e., even if directed to erase all records of the interaction). Although such a weaker notion is not satisfactory for standard cryptographic applications, it yields a fascinating notion from a conceptual as well as a complexity-theoretic point of view. Furthermore, every proof system that is *zero-knowledge with respect to the honest-verifier* can be transformed into a *standard zero-knowledge* proof (without using intractability assumptions and in case of "public-coin" proofs this is done without significantly increasing the prover's computational effort; see [214]).

Statistical versus Computational Zero-Knowledge. Recall that Definition C.9 postulates that for every probability ensemble of one type (i.e., representing the verifier's output after interaction with the prover) there exists a "similar" ensemble of a second type (i.e., representing the simulator's output). One key parameter is the interpretation of "similarity". Three interpretations, yielding different notions of zero-knowledge, have been commonly considered in the literature:

[7] The term "honest verifier" is more appealing when considering an alternative (equivalent) formulation of Definition C.9. In the alternative definition (see [87, Sec. 4.3.1.3]), the simulator is "only" required to generate the verifier's view of the real interaction, where the verifier's view includes its (common and auxiliary) inputs, the outcome of its coin tosses, and all messages it has received.

1. Perfect Zero-Knowledge requires that the two probability ensembles be identically distributed.[8]

2. Statistical Zero-Knowledge requires that these probability ensembles be statistically close (i.e., the variation distance between them is negligible).

3. Computational (or rather general) Zero-Knowledge requires that these probability ensembles be computationally indistinguishable.

Indeed, Computational Zero-Knowledge is the most liberal notion, and is the notion considered in Definition C.9. We note that the class of problems having statistical zero-knowledge proofs contains several problems that are considered intractable. The interested reader is referred to [213].

Strict versus expected probabilistic polynomial-time. The notion of probabilistic polynomial-time (which is mentioned both with respect to the verifier and the simulator), has been given two interpretations:

1. Strict probabilistic polynomial-time. That is, there exist a (polynomial in the length of the input) bound on the *number of steps in each possible run* of the machine, regardless of the outcome of its coin tosses.

2. Expected probabilistic polynomial-time. The standard approach is to look at the running-time as a random variable and *bound its expectation* (by a polynomial in the length of the input). However, as observed by Levin (see §10.2.1.1), this definitional approach is quite problematic and an alternative treatment of the aforementioned random variable is preferable.

Consequently, the notion of expected polynomial-time raises a variety of conceptual and technical problems. For that reason, whenever possible, one should prefer the more robust (and restricted) notion of strict (probabilistic) polynomial-time. Thus, with the *exception of constant-round* zero-knowledge protocols, whenever we talked of a probabilistic polynomial-time verifier (resp., simulator) we mean one in the strict sense. In contrast, with a couple of exceptions (e.g., [22]), all results regarding *constant-round* zero-knowledge protocols refer to a strict polynomial-time verifier and an expected polynomial-time simulator, which is indeed a small cheat.

C.4.4.2 Related notions: POK, NIZK, and WI

We briefly discuss the notions of proofs of knowledge (POK), non-interactive zero-knowledge (NIZK), and witness indistinguishable proofs (WI).

[8] The actual definition of Perfect Zero-Knowledge allows the simulator to fail (while outputting a special symbol) with negligible probability, and the output distribution of the simulator is conditioned on its not failing.

Proofs of Knowledge. Loosely speaking, proofs of knowledge (cf. [105]) are interactive proofs in which the prover asserts "knowledge" of some object (e.g., a 3-coloring of a graph), and not merely its existence (e.g., the existence of a 3-coloring of the graph, which in turn is equivalent to the assertion that the graph is 3-colorable). See further discussion in Section 9.2.3. We mention that "proofs of knowledge", and in particular zero-knowledge "proofs of knowledge", have many applications to the design of cryptographic schemes and cryptographic protocols. One famous application of zero-knowledge proofs of knowledge is to the construction of identification schemes (e.g., the Fiat-Shamir scheme).

Non-Interactive Zero-Knowledge. The model of non-interactive zero-knowledge proof systems consists of three entities: a prover, a verifier and a uniformly selected reference string (which can be thought of as being selected by a trusted third party). Both the verifier and prover can read the reference string (as well as the common input), and each can toss additional coins. The interaction consists of a single message sent from the prover to the verifier, who is then left with the final decision (whether or not to accept the common input). The (basic) zero-knowledge requirement refers to a simulator that outputs pairs that should be computationally indistinguishable from the distribution (of pairs consisting of a uniformly selected reference string and a random prover message) seen in the real model.[9] Non-interactive zero-knowledge proof systems have numerous applications (e.g., to the construction of public-key encryption and signature schemes, where the reference string may be incorporated in the public-key). Several different definitions of non-interactive zero-knowledge proofs were considered in the literature (see [87, Sec. 4.10] and [88, Sec. 5.4.4.4]). Constructing non-interactive zero-knowledge proofs seems more difficult than constructing interactive zero-knowledge proofs. Still, based on standard intractability assumptions (e.g., intractability of factoring), it is known how to construct a non-interactive zero-knowledge proof for any NP-set.

Witness Indistinguishability. The notion of witness indistinguishability was suggested in [72] as a meaningful relaxation of zero-knowledge. Loosely speaking, for any NP-relation R, a proof (or argument) system for the corresponding NP-set is called witness indistinguishable if no feasible verifier may distinguish the case in which the prover uses one NP-witness to x (i.e., w_1 such that $(x, w_1) \in R$) from the case in which the prover is using a different NP-witness to the same input x (i.e., w_2 such that $(x, w_2) \in R$). Clearly, any zero-knowledge protocol is witness indistinguishable, but the converse does not necessarily hold. Furthermore, it seems that witness indistinguishable protocols are easier to construct than zero-knowledge ones. Another advantage of witness indistinguishable protocols is that they are closed under arbitrary concurrent composition, whereas (in general) zero-knowledge protocols are not closed even under parallel composition. Witness indistinguishable protocols turned out to be an *important tool in the construction of more complex*

[9]Note that the verifier does not effect the distribution seen in the real model, and so the basic definition of zero-knowledge does not refer to it. The verifier (or rather a process of adaptively selecting assertions to be proved) is referred to in the adaptive variants of the definition.

protocols. We refer, in particular, to the technique of [71] for constructing zero-knowledge proofs (and arguments) based on witness indistinguishable proofs (resp., arguments).

C.5 Encryption Schemes

The problem of providing *secret communication over insecure media* is the traditional and most basic problem of cryptography. The setting of this problem consists of two parties communicating through a channel that is possibly tapped by an adversary. The parties wish to exchange information with each other, but keep the "wire-tapper" as ignorant as possible regarding the contents of this information. The canonical solution to this problem is obtained by the use of encryption schemes. Loosely speaking, an encryption scheme is a protocol allowing these parties to communicate *secretly* with each other. Typically, the encryption scheme consists of a pair of algorithms. One algorithm, called encryption, is applied by the sender (i.e., the party sending a message), while the other algorithm, called decryption, is applied by the receiver. Hence, in order to send a message, the sender first applies the encryption algorithm to the message, and sends the result, called the ciphertext, over the channel. Upon receiving a ciphertext, the other party (i.e., the receiver) applies the decryption algorithm to it, and retrieves the original message (called the plaintext).

In order for the foregoing scheme to provide secret communication, the receiver must know something that is not known to the wire-tapper. (Otherwise, the wire-tapper can decrypt the ciphertext exactly as done by the receiver.) This extra knowledge may take the form of the decryption algorithm itself, or some parameters and/or auxiliary inputs used by the decryption algorithm. We call this extra knowledge the decryption-key. Note that, without loss of generality, we may assume that the decryption algorithm is known to the wire-tapper, and that the decryption algorithm operates on two inputs: a ciphertext and a decryption-key. (This description implicitly presupposes the existence of an efficient algorithm for generating (random) keys.) We stress that the existence of a decryption-key, not known to the wire-tapper, is merely a necessary condition for secret communication.

Evaluating the "security" of an encryption scheme is a very tricky business. A preliminary task is to understand what is "security" (i.e., to properly define what is meant by this intuitive term). Two approaches to defining security are known. The first ("classical") approach, introduced by Shannon [191], is *information theoretic*. It is concerned with the "information" about the plaintext that is "present" in the ciphertext. Loosely speaking, if the ciphertext contains information about the plaintext then the encryption scheme is considered insecure. It has been shown that such high (i.e., "perfect") level of security can be achieved only if the key in use is at least as long as the *total* amount of information sent via the encryption scheme [191]. This fact (i.e., that the key has to be longer than the information exchanged using it) is indeed a drastic limitation on the applicability of such (perfectly-secure) encryption schemes.

The second ("modern") approach, followed in the current text, is based on *computational complexity*. This approach is based on the thesis that it *does not matter* whether the ciphertext contains information about the plaintext, but rather whether this information can be *efficiently extracted*. In other words, instead of asking whether it is *possible* for the wire-tapper to extract specific information, we ask whether it is *feasible* for the wire-tapper to extract this information. It turns out that the new (i.e., "computational complexity") approach can offer security even when the key is much shorter than the total length of the messages sent via the encryption scheme.

The computational complexity approach enables the introduction of concepts and primitives that cannot exist under the information theoretic approach. A typical example is the concept of *public-key encryption schemes*, introduced by Diffie and Hellman [62] (with the most popular candidate suggested by Rivest, Shamir, and Adleman [181]). Recall that in the foregoing discussion we concentrated on the decryption algorithm and its key. It can be shown that the encryption algorithm must also get, in addition to the message, an auxiliary input that depends on the decryption-key. This auxiliary input is called the encryption-key. Traditional encryption schemes, and in particular all the encryption schemes used in the millennia until the 1980's, operate with an encryption-key that equals the decryption-key. Hence, the wire-tapper in these schemes must be ignorant of the encryption-key, and consequently the *key distribution* problem arises; that is, how can two parties wishing to communicate over an insecure channel agree on a secret encryption/decryption key. (The traditional solution is to exchange the key through an alternative channel that is secure, though much more expensive to use.) The computational complexity approach allows the introduction of encryption schemes in which the encryption-key may be given to the wire-tapper without compromising the security of the scheme. Clearly, the decryption-key in such schemes is different from the encryption-key, and furthermore it is infeasible to obtain the decryption-key from the encryption-key. Such encryption schemes, called public-key schemes, have the advantage of trivially resolving the key distribution problem (because the encryption-key can be publicized). That is, once some Party X generates a pair of keys and publicizes the encryption-key, any party can send encrypted messages to Party X such that Party X can retrieve the actual information (i.e., the plaintext), whereas nobody else can learn anything about the plaintext.

In contrast to public-key schemes, traditional encryption schemes in which the encryption-key equals the description-key are called private-key schemes, because in these schemes the encryption-key must be kept secret (rather than be public as in public-key encryption schemes). We note that a full specification of either schemes requires the specification of the way in which keys are generated; that is, a (randomized) key-generation algorithm that, given a security parameter, produces a (random) pair of corresponding encryption/decryption keys (which are identical in case of private-key schemes).

Thus, both private-key and public-key encryption schemes consist of three efficient algorithms: a key generation algorithm denoted G, an encryption algorithm denoted E, and a decryption algorithm denoted D. For every pair of encryption and decryption keys (e,d) generated by G, and for every plaintext x, it holds that

$D_d(E_e(x)) = x$, where $E_e(x) \stackrel{\text{def}}{=} E(e,x)$ and $D_d(y) \stackrel{\text{def}}{=} D(d,y)$. The difference between the two types of encryption schemes is reflected in the definition of security: the security of a public-key encryption scheme should hold also when the adversary is given the encryption-key, whereas this is not required for a private-key encryption scheme. In the following definitional treatment we focus on the public-key case (and the private-key case can be obtained by omitting the encryption-key from the sequence of inputs given to the adversary).

C.5.1 Definitions

A good disguise should not reveal the person's height.

Shafi Goldwasser and Silvio Micali, 1982

For simplicity, we first consider the encryption of a single message (which, for further simplicity, is assumed to be of length that equals the security parameter, n).[10] As implied by the foregoing discussion, a public-key encryption scheme is said to be secure if it is infeasible to gain any information about the plaintext by looking at the ciphertext (and the encryption-key). That is, whatever information about the plaintext one may compute from the ciphertext and some a-priori information, can be essentially computed as efficiently from the a-priori information alone. This fundamental definition of security, called semantic security, was introduced by Goldwasser and Micali [104].

Definition C.11 (semantic security): *A* public-key *encryption scheme* (G,E,D) *is* semantically secure *if for every probabilistic polynomial-time algorithm, A, there exists a probabilistic polynomial-time algorithm B such that for every two functions $f,h:\{0,1\}^* \to \{0,1\}^*$ and all probability ensembles $\{X_n\}_{n\in\mathbb{N}}$ that satisfy $|h(x)| = \mathrm{poly}(|x|)$ and $X_n \in \{0,1\}^n$, it holds that*

$$\Pr[A(e,E_e(x),h(x))=f(x)] \ < \ \Pr[B(1^n,h(x))=f(x)] + \mu(n)$$

where the plaintext x is distributed according to X_n, the encryption-key e is distributed according to $G(1^n)$, and μ is a negligible function.

That is, it is feasible to predict $f(x)$ from $h(x)$ as successfully as it is to predict $f(x)$ from $h(x)$ and $(e,E_e(x))$, which means that nothing is gained by obtaining $(e,E_e(x))$. Note that no computational restrictions are made regarding the functions h and f. We stress that the foregoing definition (as well as the next one) refers to public-key encryption schemes, and in the case of private-key schemes algorithm A is not given the encryption-key e.

The following technical interpretation of security states that it is infeasible to distinguish the encryptions of any two plaintexts (of the same length). As we shall see, this definition (also originating in [104]) is equivalent to Definition C.11 (and meeting it requires a probabilistic encryption algorithm).

[10] In the case of public-key schemes no generality is lost by these simplifying assumptions, but in the case of private-key schemes one should consider the encryption of polynomially-many messages (as we do at the end of this section).

Definition C.12 (indistinguishability of encryptions): *A* public-key *encryption scheme* (G, E, D) *has* indistinguishable encryptions *if for every probabilistic polynomial-time algorithm, A, and all sequences of triples,* $(x_n, y_n, z_n)_{n\in\mathbb{N}}$*, where* $|x_n| = |y_n| = n$ *and* $|z_n| = \mathrm{poly}(n)$*,*

$$|\mathsf{Pr}[A(e, E_e(x_n), z_n)=1] - \mathsf{Pr}[A(e, E_e(y_n), z_n)=1]| = \mu(n)$$

Again, e is distributed according to $G(1^n)$*, and* 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-zero message and the all-ones message). Thus, the following motto is adequate too.

A good disguise should not allow a mother to distinguish her own children.

Shafi Goldwasser and Silvio Micali, 1982

Definition C.11 is more appealing in most settings where encryption is considered the end goal. Definition C.12 is used to establish the security of candidate encryption schemes as well as to analyze their application as modules inside larger cryptographic protocols. Thus, the equivalence of these definitions is of major importance.

Equivalence of Definitions C.11 and C.12 – proof ideas. Intuitively, indistinguishability of encryptions (i.e., of the encryptions of x_n and y_n) is a special case of semantic security; specifically, it corresponds to the case that X_n is uniform over $\{x_n, y_n\}$, the function f indicates one of the plaintexts and h does not distinguish them (i.e., $f(w) = 1$ iff $w = x_n$ and $h(x_n) = h(y_n) = z_n$, where z_n is as in Definition C.12). The other direction is proved by considering the algorithm B that, on input $(1^n, v)$ where $v = h(x)$, generates $(e, d) \leftarrow G(1^n)$ and outputs $A(e, E_e(1^n), v)$, where A is as in Definition C.11. Indistinguishability of encryptions is used to prove that B performs as well as A (i.e., for every h, f and $\{X_n\}_{n\in\mathbb{N}}$, it holds that $\mathsf{Pr}[B(1^n, h(X_n))=f(X_n)] = \mathsf{Pr}[A(e, E_e(1^n), h(X_n))=f(X_n)]$ approximately equals $\mathsf{Pr}[A(e, E_e(X_n), h(X_n))=f(X_n)]$).

Probabilistic Encryption: 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-zero message from the encryption of the all-ones message.[11] This explains the association of the robust security definitions and the method of *probabilistic encryption*, an association that goes back to the title of the pioneering work of Goldwasser and Micali [104].

[11] The same holds for (stateless) *private-key* encryption schemes, when considering the security of encrypting several messages (rather than a single message as done above). For example, if one uses a deterministic encryption algorithm then the adversary can distinguish two encryptions of the same message from the encryptions of a pair of different messages.

Further discussion: We stress that (the equivalent) Definitions C.11 and C.12 go way beyond saying that it is infeasible to recover the plaintext from the ciphertext. The latter statement is indeed a minimal requirement from a secure encryption scheme, but is far from being a sufficient requirement. Typically, encryption schemes are used in applications where even obtaining partial information on the plaintext may endanger 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 a specific application, it is rare (to say the least) that one has a precise characterization of all possible partial information that endanger this application. Thus, we need to require that it is 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 may be available to the adversary. We require that the secrecy of all partial information is preserved also in such a case. That is, even in presence of a-priori information on the plaintext, it is infeasible to obtain any (new) information about the plaintext from the ciphertext (beyond what 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 effect as part of larger protocols.

Security of multiple messages: Definitions C.11 and C.12 refer to the security of an encryption scheme that is used to encrypt a single plaintext (per a generated key). Since the plaintext may be longer than the key[12], these definitions are already non-trivial, and an encryption scheme satisfying them (even in the private-key model) implies the existence of one-way functions. Still, in many cases, it is desirable to encrypt many plaintexts using the same encryption-key. Loosely speaking, an encryption scheme is secure in the multiple-messages setting if conditions as in Definition C.11 (resp., Definition C.12) hold when polynomially-many plaintexts are encrypted using the same encryption-key (cf. [88, Sec. 5.2.4]). *In the public-key model*, security in the single-message setting implies security in the multiple-messages setting. We stress that this is not necessarily true *for the private-key model*.

C.5.2 Constructions

It is common practice to use "pseudorandom generators" as a basis for private-key encryption schemes. We stress that this is a very dangerous practice when the "pseudorandom generator" is easy to predict (such as the "linear congruential generator"). However, this common practice becomes sound provided one uses

[12]Recall that for sake of simplicity we have considered only messages of length n, but the general definitions refer to messages of arbitrary (polynomial in n) length. We comment that, in the general form of Definition C.11, one should provide the length of the message as an auxiliary input to both algorithms (A and B).

pseudorandom generators (as defined in Section C.3.2). An alternative and more flexible construction follows.

Private-Key Encryption Scheme based on Pseudorandom Functions: We present a simple construction that uses pseudorandom functions as defined in Section C.3.3. The key generation algorithm consists of selecting a seed, denoted s, for a (pseudorandom) 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 $(r, x \oplus f_s(r))$, where $\oplus$ denotes the exclusive-or of bit strings. 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 C.3.3):

1. Proving 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. Concluding that the real scheme is secure (because, otherwise one could distinguish a pseudorandom function from a truly random one).

Note that we could have gotten rid of the randomization (in the encryption process) if we had allowed the encryption algorithm to be history dependent (e.g., use a counter in the role of r). This can be done if all parties that use the same key (for encryption) coordinate their encryption actions (by maintaining a joint state (e.g., counter)). Indeed, when using a private-key encryption scheme, a common situation is that the same key is only used for communication between two specific parties, which update a joint counter during their communication. Furthermore, if the encryption scheme is used for FIFO communication between the parties and both parties can reliably maintain the counter value, then there is no need (for the sender) to send the counter value. (The resulting scheme is related to "stream ciphers" which are commonly used in practice.)

We comment that the use of a counter (or any other state) in the encryption process is not reasonable in the case of public-key encryption schemes, because it is incompatible with the canonical usage of such schemes (i.e., allowing all parties to send encrypted messages to the "owner of the encryption-key" without engaging in any type of further coordination or communication). Furthermore (unlike in the case of private-key schemes), probabilistic encryption is essential for a secure public-key encryption scheme *even in the case of encrypting a single message*. Following Goldwasser and Micali [104], we now demonstrate the use of *probabilistic encryption* in the construction of public-key encryption schemes.

Public-Key Encryption Scheme based on Trapdoor Permutations: We present two constructions that employ a collection of trapdoor permutations, as defined in Definition C.3. Let $\{f_i : D_i \to D_i\}_i$ be such a collection, and let b be a corresponding hard-core predicate. The key generation algorithm consists of selecting a permutation f_i along with a corresponding trapdoor t, and outputting

(i,t) as the key-pair. To encrypt a (*single*) bit σ (using the encryption-key i), the encryption algorithm uniformly selects $r \in D_i$, and produces the ciphertext $(f_i(r), \sigma \oplus b(r))$. To decrypt the ciphertext (y,τ) (using the decryption-key t), the decryption algorithm computes $\tau \oplus b(f_i^{-1}(y))$ (using the trapdoor t of f_i). Clearly, $(\sigma \oplus b(r)) \oplus b(f_i^{-1}(f_i(r))) = \sigma$. Indistinguishability of encryptions is implied by the hypothesis that b is a hard-core of f_i. We comment that this scheme is quite wasteful in bandwidth; nevertheless, the paradigm underlying its construction (i.e., applying the trapdoor permutation to a randomized version of the plaintext rather than to the actual plaintext) is valuable in practice.

A more efficient construction of a public-key encryption scheme, which uses the same key-generation algorithm, follows. To encrypt an ℓ-bit long string x (using the encryption-key i), the encryption algorithm uniformly selects $r \in D_i$, computes $y \leftarrow b(r) \cdot b(f_i(r)) \cdots b(f_i^{\ell-1}(r))$ and produces the ciphertext $(f_i^{\ell}(r), x \oplus y)$. To decrypt the ciphertext (u,v) (using the decryption-key t), the decryption algorithm first recovers $r = f_i^{-\ell}(u)$ (using the trapdoor t of f_i), and then obtains $v \oplus b(r) \cdot b(f_i(r)) \cdots b(f_i^{\ell-1}(r))$. Note the similarity to the Blum-Micali Construction (depicted in Eq. (8.8)), and the fact that the proof of the pseudorandomness of Eq. (8.8) can be extended to establish the computational indistinguishability of $(b(r) \cdots b(f_i^{\ell-1}(r)), f_i^{\ell}(r))$ and $(r', f_i^{\ell}(r))$, for random and independent $r \in D_i$ and $r' \in \{0,1\}^{\ell}$. Indistinguishability of encryptions follows, and thus the second scheme is secure. We mention that, assuming the intractability of factoring integers, this scheme has a concrete implementation with efficiency comparable to that of RSA.

C.5.3 Beyond Eavesdropping Security

Our treatment so far has referred only to a "passive" attack in which the adversary merely eavesdrops the line over which ciphertexts are sent. Stronger types of attacks (i.e., "active" ones), culminating in the so-called Chosen Ciphertext Attack, may be possible in various applications. Specifically, in some settings it is feasible for the adversary to make the sender encrypt a message of the adversary's choice, and in some settings the adversary may even make the receiver decrypt a ciphertext of the adversary's choice. This gives rise to *chosen plaintext attacks* and to *chosen ciphertext attacks*, respectively, which are not covered by the security definitions considered in Sections C.5.1 and C.5.2. Here we briefly discuss such "active" attacks, focusing on chosen ciphertext attacks (of the strongest type known as "a posteriori" or "CCA2").

Loosely speaking, in a chosen ciphertext attack, the adversary may obtain the decryptions of ciphertexts of its choice, and is deemed successful if it learns something regarding the plaintext that corresponds to some different ciphertext (see [88, Sec. 5.4.4]). That is, the adversary is given oracle access to the decryption function corresponding to the decryption-key in use (and, in the case of private-key schemes, it is also given oracle access to the corresponding encryption function). The adversary is allowed to query the decryption oracle on any ciphertext except for the "test ciphertext" (i.e., the very ciphertext for which it tries to learn something about the corresponding plaintext). It may also make queries that do not correspond to legitimate ciphertexts, and the answer will be accordingly (i.e., a special 'failure' symbol). Furthermore, the adversary may effect the selection of the test ciphertext (by specifying a distribution from which the corresponding plaintext is to be drawn).

Private-key and public-key encryption schemes secure against chosen ciphertext attacks can be constructed under (almost) the same assumptions that suffice for the construction of the corresponding passive schemes. Specifically:

Theorem C.13 *Assuming the existence of* one-way functions*, there exist* private-key *encryption schemes that are secure against chosen ciphertext attack.*

Theorem C.14 *Assuming the existence of* enhanced[13] trapdoor permutations*, there exist* public-key *encryption schemes that are secure against chosen ciphertext attack.*

Both theorems are proved by constructing encryption schemes in which the adversary's gain from a chosen ciphertext attack is eliminated by making it infeasible (for the adversary) to obtain any useful knowledge via such an attack. In the case of private-key schemes (i.e., Theorem C.13), this is achieved by making it infeasible (for the adversary) to produce legitimate ciphertexts (other than those explicitly given to it, in response to its request to encrypt plaintexts of its choice). This, in turn, is achieved by augmenting the ciphertext with an "authentication tag" that is hard to generate without knowledge of the encryption-key; that is, we use a message-authentication scheme (as defined in Section C.6). In the case of public-key schemes (i.e., Theorem C.14), the adversary can certainly generate ciphertexts by itself, and the aim is to make it infeasible (for the adversary) to produce legitimate ciphertexts without "knowing" the corresponding plaintext. This, in turn, will be achieved by augmenting the plaintext with a non-interactive zero-knowledge "proof of knowledge" of the corresponding plaintext.

Security against chosen ciphertext attack is related to the notion of *non-malleability* of the encryption scheme. Loosely speaking, in a non-malleable encryption scheme it is infeasible for an adversary, given a ciphertext, to produce a valid ciphertext for a related plaintext (e.g., given a ciphertext of a plaintext $1x$, for an unknown x, it is infeasible to produce a ciphertext to the plaintext $0x$). For further discussion see [88, Sec. 5.4.5].

C.6 Signatures and Message Authentication

Both signature schemes and message authentication schemes are methods for "validating" data; that is, verifying that the data was approved by a certain party (or set of parties). The difference between signature schemes and message authentication schemes is that signatures should be "universally verifiable", whereas authentication tags are only required to be verifiable by parties that are also able to generate them.

[13]Loosely speaking, the enhancement refers to the hardness condition of Definition C.2, and requires that it be hard to recover $f_i^{-1}(y)$ also when given the coins used to sample y (rather than merely y itself). See [88, Apdx. C.1].

Signature Schemes: The need to discuss "digital signatures" (cf. [62, 170]) has arisen with the introduction of computer communication to the business environment (in which parties need to commit themselves to proposals and/or declarations that they make). Discussions of "unforgeable signatures" did take place also prior to the computer age, but the objects of discussion were handwritten signatures (and not digital ones), and the discussion was not perceived as related to "cryptography". Loosely speaking, a scheme for unforgeable signatures should satisfy the following:

- each user can *efficiently produce its own signature* on documents of its choice;
- every user can *efficiently verify* whether a given string is a signature of another (specific) user on a specific document; but
- *it is infeasible to produce signatures of other users* to documents they did not sign.

We note that the formulation of unforgeable digital signatures provides also a clear statement of the essential ingredients of handwritten signatures. The ingredients are each person's ability to sign for itself, a universally agreed verification procedure, and the belief (or assertion) that it is infeasible (or at least hard) to forge signatures (i.e., produce some other person's signatures to documents that were not signed by it such that these "unauthentic" signatures are accepted by the verification procedure).

Message authentication schemes: Message authentication is a task related to the setting considered for encryption schemes; that is, communication over an insecure channel. This time, we consider an active adversary that is monitoring the channel and may alter the messages sent over it. The parties communicating through this insecure channel wish to authenticate the messages they send such that their counterpart can tell an original message (sent by the sender) from a modified one (i.e., modified by the adversary). Loosely speaking, a scheme for message authentication should satisfy the following:

- each of the communicating parties can *efficiently produce an authentication tag* to any message of its choice;
- each of the communicating parties can *efficiently verify* whether a given string is an authentication tag of a given message; but
- *it is infeasible for an external adversary* (i.e., a party other than the communicating parties) *to produce authentication tags* to messages not sent by the communicating parties.

Note that, in contrast to the specification of signature schemes, we do not require universal verification: only the designated receiver is required to be able to verify the authentication tags. Furthermore, we do not require that the receiver can not produce authentication tags by itself (i.e., we only require that *external parties* can not do so). Thus, 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, signatures can be used to convince third parties: in fact, a signature to a document is typically sent to a second party so that in the future this party may (by merely presenting the signed document) convince third parties that the document was indeed generated (or sent or approved) by the signer.

C.6.1 Definitions

Formally speaking, both signature schemes and message authentication schemes consist of three efficient algorithms: key generation, signing and verification. As in the case of encryption schemes, the key-generation algorithm, denoted G, is used to generate a pair of corresponding keys, one is used for signing (via algorithm S) and the other is used for verification (via algorithm V). That is, $S_s(\alpha)$ denotes a signature produced by algorithm S on input a signing-key s and a document α, whereas $V_v(\alpha, \beta)$ denotes the verdict of the verification algorithm V regarding the document α and the alleged signature β relative to the verification-key v. Needless to say, for any pair of keys (s, v) generated by G and for every α, it holds that $V_v(\alpha, S_s(\alpha)) = 1$.

The difference between the two types of schemes is reflected in the definition of security. In the case of *signature schemes*, the adversary is given the verification-key, whereas in the case of *message authentication schemes* the verification-key (which may equal the signing-key) is not given to the adversary. Thus, schemes for message authentication can be viewed as a private-key version of signature schemes. This difference yields different functionalities (even more than in the case of encryption): In typical use of a signature scheme, each user generates a pair of signing and verification keys, publicizes the verification-key and keeps the signing-key secret. Subsequently, each user may sign documents using its own signing-key, and these signatures are *universally verifiable* with respect to its public verification-key. In contrast, message authentication schemes are typically used to authenticate information sent among a set of *mutually trusting* parties that agree on a secret key, which is being used both to produce and verify authentication-tags. (Indeed, it is assumed that the mutually trusting parties have generated the key together or have exchanged the key in a secure way, prior to the communication of information that needs to be authenticated.)

We focus on the definition of secure signature schemes, and 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 (because 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 a general/robust definition of an attack seems to have to 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 means that the ability to form signatures to "nonsensical" messages is also viewed as a breaking of the scheme. Yet, again, we see no way to have a general (i.e., application-independent) notion of "meaningful" messages (such that only forging signatures to them will be considered a breaking of the scheme).

Definition C.15 (secure signature schemes – a sketch): *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, where the probability is taken over the initial choice of the key-pair as well as over the adversary's actions.*

We stress that *plain* RSA (alike plain versions of Rabin's scheme [171] and the DSS) is not secure under the above definition. However, it may be secure if the message is "randomized" before RSA (or the other schemes) is applied.

C.6.2 Constructions

Secure *message authentication schemes* can be constructed using pseudorandom functions. Specifically, the key-generation algorithm consists of selecting a seed $s \in \{0,1\}^n$ for such a function, denoted $f_s : \{0,1\}^* \to \{0,1\}^n$, and the (only valid) tag of message x with respect to the key s is $f_s(x)$. As in the case of our private-key encryption scheme, the proof of security of the current message authentication scheme consists of two steps:

1. Proving that an idealized version of the scheme, in which one uses a uniformly selected function $F : \{0,1\}^* \to \{0,1\}^n$, rather than the pseudorandom function f_s, is secure (i.e., unforgeable).
2. Concluding that the real scheme is secure (because, otherwise one could distinguish a pseudorandom function from a truly random one).

Note that this message authentication scheme makes an "extensive use of pseudorandom functions" (i.e., the pseudorandom function is applied directly to the message, which requires a generalized notion of pseudorandom functions (cf. end of Section C.3.3)). More efficient schemes can be constructed either based on a more restricted use of a pseudorandom function or based on other cryptographic primitives.

Constructing secure *signature schemes* seems more difficult than constructing message authentication schemes. Nevertheless, secure signature schemes can be constructed based on the same assumptions.

Theorem C.16 *The following three conditions are equivalent:*

1. *One-way functions exist.*
2. *Secure signature schemes exist.*

3. *Secure message authentication schemes exist.*

We stress that, unlike in the case of public-key encryption schemes, the construction of signature schemes (which may be viewed as a public-key analogue of message authentication) does not require a trapdoor property.

How to construct secure signature schemes

Three central paradigms used in the construction of secure *signature schemes* are the "refreshing" of the "effective" signing-key, the usage of an "authentication tree", and the "hashing paradigm" (to be all discussed in the sequel). In addition to being used in the proof of Theorem C.16, these three paradigms are of independent interest.

The refreshing paradigm. Introduced in [106], the *refreshing paradigm* is aimed at limiting the potential dangers of chosen message attacks. This is achieved by signing the actual document using a newly (and randomly) generated instance of the signature scheme, and authenticating (the verification-key of) this random instance with respect to the fixed public-key. That is, consider a basic signature scheme (G, S, V) used as follows. Suppose that the user U has generated a key-pair, $(s, v) \leftarrow G(1^n)$, and has placed the verification-key v on a public-file. When a party asks U to sign some document α, the user U generates a new ("fresh") key-pair, $(s', v') \leftarrow G(1^n)$, signs v' using the original signing-key s, signs α using the new signing-key s', and presents $(S_s(v'), v', S_{s'}(\alpha))$ as a signature to α. An alleged signature, (β_1, v', β_2), is verified by checking whether both $V_v(v', \beta_1) = 1$ and $V_{v'}(\alpha, \beta_2) = 1$ hold. Intuitively, the gain in terms of security is that a full-fledged chosen message attack cannot be launched on a fixed instance of (G, S, V) (i.e., on the fixed verification-key that resides in the public-file and is known to the attacker). All that an attacker may obtain (via a chosen message attack on the new scheme) is signatures, relative to the original signing-key s of (G, S, V), to random strings (distributed according to $G(1^n)$) as well as additional signatures that are each relative to a random and independently distributed signing-key.

Authentication trees. The security benefits of the refreshing paradigm are increased when combining it with the use of *authentication trees*. The idea is to use the public verification-key for authenticating several (e.g., two) fresh instances of the signature scheme, use each of these instances for authenticating several additional fresh instances, and so on. Thus, we obtain a tree of fresh instances of the basic signature scheme, where each internal node authenticates its children. We can now use the leaves of this tree for signing actual documents, where each leaf is used at most once. Thus, a signature to an actual document consists of

1. a signature to this document authenticated with respect to the verification-key associated with some leaf, and

2. a sequence of verification-keys associated with the nodes along the path from the root to this leaf, where each such verification-key is authenticated with respect to the verification-key of its parent.

We stress that the same signature, relative to the key of the parent node, is used for authenticating the verification-keys of all its children. Thus[14], each instance of the signature scheme is used for signing at most one string (i.e., a single sequence of verification-keys if the instance resides in an internal node, and an actual document if the instance resides in a leaf). Hence, it suffices to use a signature scheme that is secure as long as it is used for legitimately signing a *single* string. Such signature schemes, called one-time signature schemes, are easier to construct than standard signature schemes, especially if one only wishes to sign strings that are significantly shorter than the signing-key (resp., than the verification-key). For example, using a one-way function f, we may let the signing-key consist of a sequence of n pairs of strings, let the corresponding verification-key consist of the corresponding sequence of images of f, and sign an n-bit long message by revealing the adequate pre-images. (That is, the signing-key consist of a sequence $((s_1^0, s_1^1), ..., (s_n^0, s_n^1)) \in \{0,1\}^{2n^2}$, the corresponding verification-key is $(f(s_1^0), f(s_1^1)), ..., (f(s_n^0), f(s_n^1)))$, and the signature of the message $\sigma_1 \cdots \sigma_n$ is $(s_1^{\sigma_1}, ..., s_n^{\sigma_n})$.)

The hashing paradigm. Note, however, that in the foregoing authentication-tree, the instances of the signature scheme (associated with internal nodes) are used for signing a pair of verification-keys. Thus, we need a one-time signature scheme that can be used for signing messages that are longer than the verification-key. Here is where the *hashing paradigm* comes into play. This paradigm refers to the common practice of signing documents via a two stage process: First the actual document is hashed to a (relatively) short string, and next the basic signature scheme is applied to the resulting string. This practice (as well as other usages of the hashing paradigm) is sound provided that the hashing function belongs to a family of *collision-free hashing* (a.k.a *collision-resistant hashing*) functions. Loosely speaking, given a hash function that is randomly selected in such a family, it is infeasible to find two different strings that are hashed by this function to the same value. We also refer the interested reader to a variant of the *hashing paradigm* that uses the seemingly weaker notion of a family of *Universal One-Way Hash Functions* (see [160] or [88, Sec. 6.4.3]).

C.7 General Cryptographic Protocols

The design of secure protocols that implement arbitrary desired functionalities is a major part of modern cryptography. Taking the opposite perspective, the design of any cryptographic scheme may be viewed as the design of a secure protocol for implementing a corresponding functionality. Still, we believe that it makes sense to

[14]A naive implementation of the foregoing (full-fledged) signature scheme calls for storing in (secure) memory all the instances of the basic (one-time) signature scheme that are generated throughout the entire signing process (which refers to numerous documents). However, we note that it suffices to be able to reconstruct the random-coins used for generating each of these instances, and the former can be determined by a pseudorandom function (applied to the name of the corresponding vertex in the tree). Indeed, the seed of this pseudorandom function will be part of the signing-key of the resulting (full-fledged) signature scheme.

differentiate between basic cryptographic primitives (which involve little interaction) like encryption and signature schemes on one hand, and general cryptographic protocols on the other hand.

In this section, we survey *general* results concerning secure *multi*-party computations, where the *two*-party case is an important special case. In a nutshell, these results assert that one can construct protocols for securely computing *any* desirable multi-party functionality. Indeed, what is striking about these results is their generality, and we believe that the wonder is not diminished by the (various alternative) conditions under which these results hold.

A general framework for casting (m-party) cryptographic (protocol) problems consists of specifying a random process[15] that maps m inputs to m outputs. The inputs to the process are to be thought of as the local inputs of m parties, and the m outputs are their corresponding local outputs. The random process describes the desired functionality. That is, if the m parties were to trust each other (or trust some external party), then they could each send their local input to the trusted party, who would compute the outcome of the process and send to each party the corresponding output. A pivotal question in the area of cryptographic protocols is to what extent can this (imaginary) trusted party be "emulated" by the mutually distrustful parties themselves.

The results surveyed in this section describe a variety of models in which such an "emulation" is possible. The models vary by the underlying assumptions regarding the communication channels, numerous parameters governing the extent of adversarial behavior, and the desired level of emulation of the trusted party (i.e., level of "security"). Our treatment refers to the security of stand-alone executions. The preservation of security in an environment in which many executions of many protocols are attacked is beyond the scope of this section, and the interested reader is referred to [88, Sec. 7.7.2].

C.7.1 The Definitional Approach and Some Models

Before describing the aforementioned results, we further discuss the notion of "emulating a trusted party", which underlies the definitional approach to secure multi-party computation. This approach follows the simulation paradigm (cf. Section C.4.1) which deems a scheme to be secure if whatever a feasible adversary can obtain after attacking it, is also feasibly attainable by a benign behavior. In the general setting of multi-party computation we compare the effect of adversaries that participate in the execution of the actual protocol to the effect of adversaries that participate in an imaginary execution of a trivial (ideal) protocol for computing the desired functionality with the help of a trusted party. If whatever the adversaries can feasibly obtain in the real setting can also be feasibly obtained in

[15]That is, we consider the secure evaluation of randomized functionalities, rather than "only" the secure evaluation of functions. Specifically, we consider an arbitrary (randomized) process F that on input $(x_1,...,x_m)$, first selects at random (depending only on $\ell \stackrel{\text{def}}{=} \sum_{i=1}^m |x_i|$) an m-ary function f, and then outputs the m-tuple $f(x_1,...,x_m) = (f_1(x_1,...,x_m),...,f_m(x_1,...,x_m))$. In other words, $F(x_1,...,x_m) = F'(r,x_1,...,x_m)$, where r is uniformly selected in $\{0,1\}^{\ell'}$ (with $\ell' = \mathrm{poly}(\ell)$), and F' is a function mapping $(m+1)$-long sequences to m-long sequences.

the ideal setting then the actual protocol "emulates the ideal setting" (i.e., "emulates a trusted party"), and thus is deemed secure. This basic approach can be applied in a variety of models, and is used to define the goals of security in these models.[16] We first discuss some of the parameters used in defining various models, and next demonstrate the application of this approach in two important cases. For further details, see [88, Sec. 7.2 and 7.5.1].

C.7.1.1 Some parameters used in defining security models

The following parameters are described in terms of the actual (or real) computation. In *some cases*, the corresponding definition of security is obtained by imposing some restrictions or provisions on the ideal model. For example, in the case of two-party computation (see §C.7.1.3), secure computation is possible only if premature termination is *not* considered a breach of security. In that case, the suitable security definition is obtained (via the simulation paradigm) by allowing (an analogue of) premature termination in the ideal model. In *all cases*, the desired notion of security is defined by requiring that for any adequate adversary in the real model, there exist a corresponding adversary in the corresponding ideal model that obtains essentially the same impact (as the real-model adversary).

The communication channels: The standard assumption in cryptography is that the adversary may tap all communication channels (between honest parties), but cannot modify (or omit or insert) messages sent over them. In contrast, one may *postulate* that the adversary cannot obtain messages sent between a pair of honest parties, yielding the so-called private-channel model. Most works in the area assume that communication is *synchronous* and that point-to-point channels exist between every pair of processors (i.e., a *complete network*).

Set-up assumptions: Unless stated differently, no set-up assumptions are made (except for the obvious assumption that all parties have identical copies of the protocol's program).

Computational limitations: Typically, the focus is on computationally-bounded adversaries (e.g., probabilistic polynomial-time adversaries). However, the private-channel model allows for the (meaningful) consideration of computationally-unbounded adversaries.[17]

[16] A few technical comments are in place. Firstly, we assume that the inputs of all parties are of the same length. We comment that as long as the lengths of the inputs are polynomially related, the foregoing convention can be enforced by padding. On the other hand, some length restriction is essential for the security results, because in general it is impossible to hide all information regarding the length of the inputs to a protocol. Secondly, we assume that the desired functionality is computable in probabilistic polynomial-time, because we wish the secure protocol to run in probabilistic polynomial-time (and a protocol cannot be more efficient than the corresponding centralized algorithm). Clearly, the results can be extended to functionalities that are computable within any given (time-constructible) time bound, using adequate padding.

[17] We stress that, also in the case of computationally-unbounded adversaries, security should be defined by requiring that for every real adversary, whatever the adversary can compute after

Restricted adversarial behavior: The parameters of the model include questions like whether the adversary is "active" or "passive" (i.e., whether a dishonest party takes active steps to disrupt the execution of the protocol or merely gathers information) and whether or not the adversary is "adaptive" (i.e., whether the set of dishonest parties is fixed before the execution starts or is adaptively chosen by an adversary during the execution).

Restricted notions of security: One important example is the willingness to tolerate "unfair" protocols in which the execution can be suspended (at any time) by a dishonest party, provided that it is detected doing so. We stress that in case the execution is suspended, the dishonest party does not obtain more information than it could have obtained when not suspending the execution. (What may happen is that the honest parties will not obtain their desired outputs, but will detect that the execution was suspended.) We stress that the motivation to this restricted model is the impossibility of obtaining general secure two-party computation in the unrestricted model.

Upper bounds on the number of dishonest parties: These are assumed in some models, when required. For example, in some models, secure multi-party computation is possible only if a majority of the parties is honest.

C.7.1.2 Example: Multi-party protocols with honest majority

Here we consider an active, non-adaptive, computationally-bounded adversary, and do not assume the existence of private channels. Our aim is to define multi-party protocols that remain secure provided that the honest parties are in majority. (The reason for requiring an honest majority will be discussed at the end of this subsection.)

We first observe that in any multi-party protocol, each party may change its local input before even entering the execution of the protocol. However, this is unavoidable also when the parties utilize a trusted party. Consequently, such an effect of the adversary on the real execution (i.e., modification of its own input prior to entering the actual execution) is not considered a breach of security. In general, whatever cannot be avoided when the parties utilize a trusted party, is not considered a breach of security. We wish secure protocols (in the real model) to suffer only from whatever is unavoidable also when the parties utilize a trusted party. Thus, the basic paradigm underlying the definitions of *secure multi-party computations* amounts to requiring that the only situations that may occur in the real execution of a secure protocol are those that can also occur in a corresponding ideal model (where the parties may employ a trusted party). In other words, the

participating in the execution of the actual protocol is computable *within comparable time* by an imaginary adversary participating in an imaginary execution of the trivial ideal protocol (for computing the desired functionality with the help of a trusted party). That is, although no computational restrictions are made on the real-model adversary, it is required that the ideal-model adversary that obtains the same impact does so within comparable time (i.e., within time that is polynomially related to the running time of the real-model adversary being simulated).

"effective malfunctioning" of parties in secure protocols is restricted to what is postulated in the corresponding ideal model.

When defining secure multi-party protocols (with honest majority), we need to pin-point what cannot be avoided in the ideal model (i.e., when the parties utilize a trusted party). Since we are interested in executions in which the majority of parties are honest, we consider an ideal model in which any minority group (of the parties) may collude as follows:

1. Firstly this dishonest minority shares its original inputs and decides together on replaced inputs to be sent to the trusted party. (The other parties send their respective original inputs to the trusted party.)

2. Upon receiving inputs from all parties, the trusted party determines the corresponding outputs and sends them to the corresponding parties. (We stress that the information sent between the honest parties and the trusted party is not seen by the dishonest colluding minority.)

3. Upon receiving the output-message from the trusted party, each honest party outputs it locally, whereas the dishonest colluding minority may determine their outputs based on all they know (i.e., their initial inputs and their received outputs).

A *secure multi-party computation with honest majority* is required to emulate this ideal model. That is, the effect of any feasible adversary that controls a minority of the parties in a real execution of such a (real) protocol, can be essentially simulated by a (different) feasible adversary that controls the corresponding parties in the ideal model.

Definition C.17 (secure protocols – a sketch): *Let f be an m-ary functionality and Π be an m-party protocol operating in the real model.*

- *For a real-model adversary A, controlling some minority of the parties* (and tapping all communication channels)*, and an m-sequence $\overline{x}$, we denote by* $\text{REAL}_{\Pi,A}(\overline{x})$ *the sequence of m outputs resulting from the execution of Π on input $\overline{x}$ under the attack of the adversary A.*

- *For an ideal-model adversary A', controlling some minority of the parties, and an m-sequence $\overline{x}$, we denote by* $\text{IDEAL}_{f,A'}(\overline{x})$ *the sequence of m outputs resulting from the foregoing three-step ideal process, when applied to input $\overline{x}$ under the attack of the adversary A'.*

We say that Π securely implements f with honest majority *if for every feasible real-model adversary A, controlling some minority of the parties, there exists a feasible ideal-model adversary A', controlling the same parties, such that the probability ensembles* $\{\text{REAL}_{\Pi,A}(\overline{x})\}_{\overline{x}}$ *and* $\{\text{IDEAL}_{f,A'}(\overline{x})\}_{\overline{x}}$ *are computationally indistinguishable (as in Definition C.5).*

Thus, security means that the effect of each minority group in a real execution of a secure protocol is "essentially restricted" to replacing its own local inputs (independently of the local inputs of the majority parties) 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 parties do obtain additional pieces of information; yet in a secure protocol they gain nothing from these additional pieces of information, because they can actually reproduce those by themselves.)

The fact that Definition C.17 refers to a model without private channels is reflected in the fact that our (sketchy) definition of the real-model adversary allowed it to tap the channels, which in turn effects the set of possible ensembles $\{\text{REAL}_{\Pi,A}(\overline{x})\}_{\overline{x}}$. When defining security in the private-channel model, the real-model adversary is not allowed to tap channels between honest parties, and this again effects the possible ensembles $\{\text{REAL}_{\Pi,A}(\overline{x})\}_{\overline{x}}$. On the other hand, when we wish to define security with respect to passive adversaries, both the scope of the real-model adversaries and the scope of the ideal-model adversaries changes. In the real-model execution, all parties follow the protocol but the adversary may alter the output of the dishonest parties arbitrarily depending on their intermediate internal states (during the entire execution). In the corresponding ideal-model, the adversary is not allowed to modify the *inputs* of dishonest parties (in Step 1), but is allowed to modify their outputs (in Step 3).

We comment that a definition analogous to Definition C.17 can be presented also in the case that the dishonest parties are not in minority. In fact, such a definition seems more natural, but the problem is that such a definition cannot be satisfied. That is, most (natural) functionalities do not have a protocol for computing them securely in the case that at least half of the parties are dishonest and employ an adequate adversarial strategy. This follows from an impossibility result regarding two-party computation, which essentially asserts that there is no way to prevent a party from prematurely suspending the execution. On the other hand, secure multi-party computation with dishonest majority is possible if premature suspension of the execution is not considered a breach of security (see §C.7.1.3).

C.7.1.3 Another example: Two-party protocols allowing abort

In light of the last paragraph, we now consider multi-party computations in which premature suspension of the execution is not considered a breach of security. For simplicity, we focus on the special case of two-party computations (As in §C.7.1.2, we consider a non-adaptive, active, computationally-bounded adversary.)

Intuitively, in any two-party protocol, each party may suspend the execution at any point in time, and furthermore it may do so as soon as it learns the desired output. Thus, in case the output of each parties depends on both inputs, it is always possible for one of the parties to obtain the desired output while preventing the other party from fully determining its own output.[18] The same phenomenon occurs even in case the two parties just wish to generate a common random value. Thus, when considering active adversaries in the two-party setting, we do not consider

[18]In contrast, in the case of an honest majority (cf., §C.7.1.2), the honest party that fails to obtain its output is not alone. It may seek help from the other honest parties, which together and being in majority can do things that dishonest minorities cannot do: See §C.7.3.2.

such premature suspension of the execution a breach of security. Consequently, we consider an ideal model where each of the two parties may "shut-down" the trusted (third) party at any point in time. In particular, this may happen after the trusted party has supplied the outcome of the computation to one party but before it has supplied the outcome to the other. Thus, an execution in the corresponding ideal model proceeds as follows:

1. Each party sends its input to the trusted party, where the dishonest party may replace its input or send no input at all (which can be treated as sending a default value).

2. Upon receiving inputs from both parties, the trusted party determines the corresponding pair of outputs, and sends the first output to the first party.

3. In case the first party is dishonest, it may instruct the trusted party to halt, otherwise it always instructs the trusted party to proceed. If instructed to proceed, the trusted party sends the second output to the second party.

4. Upon receiving the output-message from the trusted party, an honest party outputs it locally, whereas a dishonest party may determine its output based on all it knows (i.e., its initial input and its received output).

A secure two-party computation allowing abort is required to emulate this ideal model. That is, as in Definition C.17, security is defined by requiring that for every feasible real-model adversary A, there exists a feasible ideal-model adversary A', controlling the same party, such that the probability ensembles representing the corresponding (real and ideal) executions are computationally indistinguishable. This means that each party's "effective malfunctioning" in a secure protocol is restricted to supplying an initial input of its choice and aborting the computation at any point in time. (Needless to say, the choice of the initial input of each party may *not* depend on the input of the other party.)

We mention that an alternative way of dealing with the problem of premature suspension of execution (i.e., abort) is to restrict our attention to single-output functionalities; that is, functionalities in which only one party is supposed to obtain an output. The definition of secure computation of such functionalities can be made identical to Definition C.17, with the exception that no restriction is made on the set of dishonest parties (and in particular one may consider a single dishonest party in the case of two-party protocols). For further details, see [88, Sec. 7.2.3].

C.7.2 Some Known Results

We next list some of the models for which general secure multi-party computation is known to be attainable (i.e., models in which one can construct secure multi-party protocols for computing any desired functionality). We mention that the first results of this type were obtained by Goldreich, Micali, Wigderson and Yao [96, 225, 97].

In the standard channel model. *Assuming the existence of enhanced*[19] *trapdoor permutations*, secure multi-party computation is possible in the following three models (cf. [96, 225, 97] and details in [88, Chap. 7]):

1. Passive adversary, for any number of dishonest parties (see [88, Sec. 7.3]).
2. Active adversary that may control only a minority of the parties (see [88, Sec. 7.5.4]).
3. Active adversary, for any number of dishonest parties, provided that suspension of execution (as discussed in §C.7.1.3) is not considered a violation of security (see [88, Sec. 7.4 and 7.5.5]).

In all these cases, the adversary is computationally-bounded and non-adaptive. On the other hand, the adversary may tap the communication lines between honest parties (i.e., we do not assume "private channels" here). The results for active adversaries assume a broadcast channel. Indeed, the latter can be implemented (while tolerating any number of dishonest parties) using a signature scheme and assuming that each party knows (or can reliably obtain) the verification-key corresponding to each of the other parties.

In the private channels model. Making no computational assumptions and allowing computationally-unbounded adversaries, but *assuming private channels*, secure multi-party computation is possible in the following two models (cf. [32, 50]):

1. Passive adversary that may control only a minority of the parties.
2. Active adversary that may control only less than one third of the parties.

In both cases the adversary may be adaptive.

C.7.3 Construction Paradigms and Two Simple Protocols

We briefly sketch a couple of paradigms used in the construction of secure multi-party protocols. We focus on the construction of secure protocols for the model of computationally-bounded and non-adaptive adversaries [96, 225, 97]. These constructions proceed in two steps (see details in [88, Chap. 7]): First a secure protocol is presented for the model of passive adversaries (for any number of dishonest parties), and next such a protocol is "compiled" into a protocol that is secure in one of the two models of active adversaries (i.e., either in a model allowing the adversary to control only a minority of the parties or in a model in which premature suspension of the execution is not considered a violation of security). These two steps are presented in the following two corresponding subsections, in which we also present two relatively simple protocols for two specific tasks, which in turn are used extensively in the general protocols.

Recall that in the model of passive adversaries, all parties follow the prescribed protocol, but at termination the adversary may alter the outputs of the dishonest

[19] See Footnote 13.

parties depending on their intermediate internal states (during the entire execution). Below, we refer to protocols that are secure in the model of passive (resp., active) adversaries by the term passively-secure (resp., actively-secure).

C.7.3.1 Passively-secure computation with shares

For any $m \geq 2$, suppose that m parties, each having a private input, wish to obtain the value of a predetermined m-argument function evaluated at their sequence of inputs. Below, we outline a passively-secure protocol for achieving this goal. We mention that the design of passively-secure multi-party protocol for any functionality (allowing different outputs to different parties as well as handling also randomized computations) reduces easily to the aforementioned task.

We assume that the parties hold a circuit for computing the value of the function on inputs of the adequate length, and that the circuit contains only `and` and `not` gates. The key idea is having each party "secretly share" its input with everybody else, and having the parties "secretly transform" shares of the input wires of the circuit into shares of the output wires of the circuit, thus obtaining shares of the outputs (which allows for the reconstruction of the actual outputs). The value of each wire in the circuit is shared such that all shares yield the value, whereas lacking even one of the shares keeps the value totally undetermined. That is, we use a simple secret sharing scheme such that a bit b is shared by a random sequence of m bits that sum-up to $b \bmod 2$. First, each party shares each of its input bits with all parties (by secretly sending each party a random value and setting its own share accordingly). Next, all parties jointly scan the circuit from its input wires to its output wires, processing each gate as follows:

- When encountering a gate, the parties already hold shares of the values of the wires entering the gate, and their aim is to obtain shares of the value of the wires exiting the gate.
- For a `not`-gate this is easy: the first party just flips the value of its share, and all other parties maintain their shares.
- Since an `and`-gate corresponds to multiplication modulo 2, the parties need to securely compute the following randomized functionality (in which the x_i's denote shares of one entry-wire, the y_i's denote shares of the second entry-wire, the z_i's denote shares of the exit-wire, and the shares indexed by i are held by Party i):

$$((x_1, y_1), ..., (x_m, y_m)) \mapsto (z_1, ..., z_m), \text{ where} \tag{C.1}$$

$$\sum_{i=1}^{m} z_i = \left(\sum_{i=1}^{m} x_i\right) \cdot \left(\sum_{i=1}^{m} y_i\right) \tag{C.2}$$

  That is, the z_i's are random subject to Eq. (C.2).

Finally, the parties send their shares of each circuit-output wire to the designated party, which reconstructs the value of the corresponding bit. Thus, the parties have propagated shares of the circuit-input wires into shares of the circuit-output wires, by repeatedly conducting passively-secure computation of the m-ary functionality of Eq. (C.1) & (C.2). That is, securely evaluating the entire (arbitrary) circuit "reduces" to securely conducting a specific (very simple) multi-party computation. But things get even simpler: the key observation is that

$$\left(\sum_{i=1}^{m} x_i\right) \cdot \left(\sum_{i=1}^{m} y_i\right) = \sum_{i=1}^{m} x_i y_i + \sum_{1 \le i < j \le m} (x_i y_j + x_j y_i) . \tag{C.3}$$

Thus, the m-ary functionality of Eq. (C.1) & (C.2) can be computed as follows (where all arithmetic operations are mod 2):

1. Each Party i locally computes $z_{i,i} \stackrel{\text{def}}{=} x_i y_i$.
2. Next, each pair of parties (i.e., Parties i and j) securely compute random shares of $x_i y_j + y_i x_j$. That is, Parties i and j (holding (x_i, y_i) and (x_j, y_j), respectively), need to securely compute the randomized two-party functionality $((x_i, y_i), (x_j, y_j)) \mapsto (z_{i,j}, z_{j,i})$, where the z's are random subject to $z_{i,j} + z_{j,i} = x_i y_j + y_i x_j$. Equivalently, Party j uniformly selects $z_{j,i} \in \{0,1\}$, and Parties i and j securely compute the following deterministic functionality
$$((x_i, y_i), (x_j, y_j, z_{j,i})) \mapsto (z_{j,i} + x_i y_j + y_i x_j, \lambda), \tag{C.4}$$
where λ denotes the empty string.
3. Finally, for every $i = 1, ..., m$, the sum $\sum_{j=1}^{m} z_{i,j}$ yields the desired share of Party i.

The foregoing construction is analogous to a construction that was outlined in [97]. A detailed description and full proofs appear in [88, Sec. 7.3.4 and 7.5.2].

The foregoing construction reduces the passively-secure computation of any m-ary functionality to the implementation of the simple 2-ary functionality of Eq. (C.4). The latter can be implemented in a passively-secure manner by using a 1-out-of-4 Oblivious Transfer. Loosely speaking, a 1-out-of-k Oblivious Transfer is a protocol enabling one party to obtain one of k secrets held by another party, without the second party learning which secret was obtained by the first party. That is, it allows a passively-secure computation of the two-party functionality

$$(i, (s_1, ..., s_k)) \mapsto (s_i, \lambda). \tag{C.5}$$

Note that any function $f : [k] \times \{0,1\}^* \to \{0,1\}^* \times \{\lambda\}$ can be computed in a passively-secure manner by invoking a 1-out-of-k Oblivious Transfer on inputs i and $(f(1, y), ..., f(k, y))$, where i (resp., y) is the initial input of the first (resp., second) party.

A passively-secure 1-out-of-k Oblivious Transfer. Using a collection of enhanced trapdoor permutations, $\{f_\alpha : D_\alpha \to D_\alpha\}_{\alpha \in \overline{I}}$ and a corresponding hard-core predicate b, we outline a passively-secure implementation of the functionality of Eq. (C.5), when restricted to single-bit secrets.

Inputs: The first party, hereafter called the receiver, has input $i \in \{1, 2, ..., k\}$. The second party, called the sender, has input $(\sigma_1, \sigma_2, ..., \sigma_k) \in \{0, 1\}^k$.

Step S1: The sender selects at random a permutation f_α along with a corresponding trapdoor, denoted t, and sends the permutation f_α (i.e., its index α) to the receiver.

Step R1: The receiver uniformly and independently selects $x_1, ..., x_k \in D_\alpha$, sets $y_i = f_\alpha(x_i)$ and $y_j = x_j$ for every $j \neq i$, and sends $(y_1, y_2, ..., y_k)$ to the sender.

Thus, the receiver knows $f_\alpha^{-1}(y_i) = x_i$, but cannot predict $b(f_\alpha^{-1}(y_j))$ for any $j \neq i$. Needless to say, the last assertion presumes that the receiver follows the protocol (i.e., we only consider passive-security).

Step S2: Upon receiving $(y_1, y_2, ..., y_k)$, using the inverting-with-trapdoor algorithm and the trapdoor t, the sender computes $z_j = f_\alpha^{-1}(y_j)$, for every $j \in \{1, ..., k\}$. It sends the k-tuple $(\sigma_1 \oplus b(z_1), \sigma_2 \oplus b(z_2), ..., \sigma_k \oplus b(z_k))$ to the receiver.

Step R2: Upon receiving $(c_1, c_2, ..., c_k)$, the receiver locally outputs $c_i \oplus b(x_i)$.

We first observe that this protocol correctly computes 1-out-of-k Oblivious Transfer; that is, the receiver's local output (i.e., $c_i \oplus b(x_i)$) indeed equals $(\sigma_i \oplus b(f_\alpha^{-1}(f_\alpha(x_i)))) \oplus b(x_i) = \sigma_i$. Next, we offer some intuition as to why this protocol constitutes a privately-secure implementation of 1-out-of-k Oblivious Transfer. Intuitively, the sender gets no information from the execution because, for any possible value of i, the senders sees the same distribution; specifically, a sequence of k uniformly and independently distributed elements of D_α. (Indeed, the key observation is that applying f_α to a uniformly distributed element of D_α yields a uniformly distributed element of D_α.) As for the receiver, intuitively, it gains no computational knowledge from the execution because, for $j \neq i$, the only information that the receiver has regarding σ_j is the triplet $(\alpha, x_j, \sigma_j \oplus b(f_\alpha^{-1}(x_j)))$, where x_j is uniformly distributed in D_α, and from this information it is infeasible to predict σ_j better than by a random guess.[20] (See [88, Sec. 7.3.2] for a detailed proof of security.)

C.7.3.2 From passively-secure protocols to actively-secure ones

We show how to transform any passively-secure protocol into a corresponding actively-secure protocol. The communication model in both protocols consists of a single broadcast channel. Note that the messages of the original protocol may be assumed to be sent over a broadcast channel, because the adversary may see them anyhow (by tapping the point-to-point channels), and because a broadcast

[20]The latter intuition presumes that sampling D_α is trivial (i.e., that there is an easily computable correspondence between the coins used for sampling and the resulting sample), whereas in general the coins used for sampling may be hard to compute from the corresponding outcome. This is the reason that an enhanced hardness assumption is used in the general analysis of the foregoing protocol.

channel is trivially implementable in the case of passive adversaries. As for the resulting actively-secure protocol, the broadcast channel it uses can be implemented via an (authenticated) Byzantine Agreement protocol, thus providing an emulation of this model on the standard point-to-point model (in which a broadcast channel does not exist). We mention that authenticated Byzantine Agreement is typically implemented using a signature scheme (and assuming that each party knows the verification-key corresponding to each of the other parties).

Turning to the transformation itself, the main idea is using zero-knowledge proofs (as described in §C.4.3.3) in order to force parties to behave in a way that is consistent with the (passively-secure) protocol. Actually, we need to confine each party to a unique consistent behavior (i.e., according to some fixed local input and a sequence of coin tosses), and to guarantee that a party cannot fix its input (and/or its coin tosses) in a way that depends on the inputs (and/or coin tosses) of honest parties. Thus, some preliminary steps have to be taken before the step-by-step emulation of the original protocol may start. Specifically, the compiled protocol (which, like the original protocol, is executed over a broadcast channel) proceeds as follows:

1. *Committing to the local input*: Prior to the emulation of the original protocol, each party commits to its input (using a commitment scheme as defined in §C.4.3.1). In addition, using a zero-knowledge proof-of-knowledge (see Section 9.2.3), each party also proves that it knows its own input; that is, it proves that it can decommit to the commitment it sent. (These zero-knowledge proof-of-knowledge prevent dishonest parties from setting their inputs in a way that depends on inputs of honest parties.)

2. *Generation of local random tapes*: Next, all parties jointly generate a sequence of random bits for each party such that only this party knows the outcome of the random sequence generated for it, and everybody else gets a commitment to this outcome. These sequences will be used as the random-inputs (i.e., sequence of coin tosses) for the original protocol. Each bit in the random-sequence generated for Party X is determined as the exclusive-or of the outcomes of instances of an (augmented) coin-tossing protocol (cf. [88, Sec. 7.4.3.5]) that Party X plays with each of the other parties. The latter protocol provides the other parties with a commitment to the outcome obtained by Party X.

3. *Effective prevention of premature termination*: In addition, when compiling (the passively-secure protocol to an actively-secure protocol) *for the model that allows the adversary to control only a minority of the parties*, each party shares its input and random-input with all other parties using a "Verifiable Secret Sharing" (VSS) protocol (cf. [88, Sec. 7.5.5.1]). Loosely speaking, a VSS protocol allows sharing a secret in a way that enables each participant to verify that the share it got fits the publicly posted information, which includes commitments to all shares, where a sufficient number of the latter allow for the efficient recovery of the secret. The use of VSS guarantees that if Party X prematurely suspends the execution, then the honest parties can

together reconstruct all Party X's secrets and carry on the execution while playing its role. This step effectively prevents premature termination, and is not needed in a model that does not consider premature termination a breach of security.

4. *Step-by-step emulation of the original protocol*: Once all the foregoing steps are completed, the new protocol emulates the steps of the original protocol. In each step, each party augments the message determined by the original protocol with a zero-knowledge proof that asserts that the message was indeed computed correctly. Recall that the next message (as determined by the original protocol) is a function of the sender's own input, its random-input, and the messages it has received so far (where the latter are known to everybody because they were sent over a broadcast channel). Furthermore, the sender's input is determined by its commitment (as sent in Step 1), and its random-input is similarly determined (in Step 2). Thus, the next message (as determined by the original protocol) is a function of publicly known strings (i.e., the said commitments as well as the other messages sent over the broadcast channel). Moreover, the assertion that the next message was indeed computed correctly is an NP-assertion, and the sender knows a corresponding NP-witness (i.e., its own input and random-input as well as the corresponding decommitment information). Thus, the sender can prove in zero-knowledge (to each of the other parties) that the message it is sending was indeed computed according to the original protocol.

The above compilation was first outlined in [96, 97]. A detailed description and full proofs appear in [88, Sec. 7.4 and 7.5].

A secure coin-tossing protocol. Using a commitment scheme, we outline a secure (ordinary as opposed to augmented) coin-tossing protocol.

Step C1: Party 1 uniformly selects $\sigma \in \{0,1\}$ and sends Party 2 a commitment, denoted c, to σ.

Step C2: Party 2 uniformly selects $\sigma' \in \{0,1\}$, and sends σ' to Party 1.

Step C3: Party 1 outputs the value $\sigma \oplus \sigma'$, and sends σ along with the decommitment information, denoted d, to Party 2.

Step C4: Party 2 checks whether or not (σ, d) fit the commitment c it has obtained in Step 1. It outputs $\sigma \oplus \sigma'$ if the check is satisfied and halts with output $\perp$ otherwise, where $\perp$ indicates that Party 1 has effectively aborted the protocol prematurely.

Outputs: Party 1 always outputs $b \stackrel{\text{def}}{=} \sigma \oplus \sigma'$, whereas Party 2 either outputs b or $\perp$.

Intuitively, Steps C1–C2 may be viewed as "tossing a coin into the well". At this point (i.e., after Step C2), the value of the coin is determined (essentially as a random value), but only one party (i.e., Party 1) "can see" (i.e., knows) this value. Clearly, if both parties are honest then they both output the same uniformly chosen bit, recovered in Steps C3 and C4, respectively. Intuitively, each party can guarantee that the outcome is uniformly distributed, and Party 1 can cause premature termination by improper execution of Step 3. Formally, we have to show how the effect of any real-model adversary can be simulated by an adequate ideal-model adversary (which is allowed premature termination). This is done in [88, Sec. 7.4.3.1].

C.7.4 Concluding Remarks

In Sections C.7.1-C.7.2 we have mentioned numerous definitions and results regarding secure multi-party protocols, where some of these definitions are incomparable to others (i.e., they neither imply the others nor are implies by them). For example, in §C.7.1.2 and §C.7.1.3, we have presented two alternative definitions of "secure multi-party protocols", one requiring an honest majority and the other allowing abort. These definitions are incomparable and there is no generic reason to prefer one over the other. Actually, as mentioned in §C.7.1.2, one could formulate a natural definition that implies both definitions (i.e., waiving the bound on the number of dishonest parties in Definition C.17). Indeed, the resulting definition is free of the annoying restrictions that were introduced in each of the two aforementioned definitions; the "only" problem with the resulting definition is that it cannot be satisfied (in general). Thus, for the first time in this appendix, we have reached a situation in which a natural (and general) definition cannot be satisfied, and we are forced to choose between two weaker alternatives, where each of these alternatives carries fundamental disadvantages.

In general, Section C.7 carries a stronger flavor of compromise (i.e., recognizing inherent limitations and settling for a restricted meaningful goal) than previous sections. In contrast to the impression given in other parts of this appendix, it is now obvious that we cannot get all that we may want (and this is without mentioning the problems involved in preserving security under concurrent composition; cf. [88, Sec. 7.7.2]). Instead, we should study the alternatives, and go for the one that best suits our real needs.

Indeed, as stated in Section C.1, the fact that we can define a cryptographic goal does not mean that we can satisfy it as defined. In case we cannot satisfy the initial definition, we should search for relaxations that can be satisfied. These relaxations should be defined in a clear manner such that it would be obvious what they achieve (and what they fail to achieve). Doing so will allow a sound choice of the relaxation to be used in a specific application. This seems to be a good point to end the current appendix.

A good compromise is one in which the most important interests of all parties are satisfied.

Adv. Klara Goldreich-Ingwer (1912–2004)

Bibliography

[1] S. Aaronson. Complexity Zoo. A continueously updated web-site at `http://qwiki.caltech.edu/wiki/Complexity_Zoo/`.

[2] 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 STOC*, 1987.

[3] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *Annals of Mathematics*, Vol. 160 (2), pages 781–793, 2004.

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

[5] R. Aleliunas, R.M. Karp, R.J. Lipton, L. Lovász and C. Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *20th IEEE Symposium on Foundations of Computer Science*, pages 218–223, 1979.

[6] N. Alon, L. Babai and A. Itai. A fast and Simple Randomized Algorithm for the Maximal Independent Set Problem. *J. of Algorithms*, Vol. 7, pages 567–583, 1986.

[7] N. Alon and R. Boppana. The monotone circuit complexity of Boolean functions. *Combinatorica*, Vol. 7 (1), pages 1–22, 1987.

[8] N. Alon, E. Fischer, I. Newman, and A. Shapira. A Combinatorial Characterization of the Testable Graph Properties: It's All About Regularity. In *38th ACM Symposium on the Theory of Computing*, to appear, 2006.

[9] N. Alon, O. Goldreich, J. Håstad, R. Peralta. Simple Constructions of Almost k-wise Independent Random Variables. *Journal of Random structures and Algorithms*, Vol. 3, No. 3, (1992), pages 289–304.

[10] N. Alon and J.H. Spencer. *The Probabilistic Method*. John Wiley & Sons, Inc., 1992.

[11] R. Armoni. On the derandomization of space-bounded computations. In the proceedings of *Random98*, Springer-Verlag, Lecture Notes in Computer Science (Vol. 1518), pages 49–57, 1998.

[12] S. Arora. Approximation schemes for NP-hard geometric optimization problems: A survey. *Math. Programming*, Vol. 97, pages 43–69, July 2003.

[13] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy. Proof Verification and Intractability of Approximation Problems. *Journal of the ACM*, Vol. 45, pages 501–555, 1998. Preliminary version in *33rd FOCS*, 1992.

[14] S. Arora and S. Safra. Probabilistic Checkable Proofs: A New Characterization of NP. *Journal of the ACM*, Vol. 45, pages 70–122, 1998. Preliminary version in *33rd FOCS*, 1992.

[15] H. Attiya and J. Welch: *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. McGraw-Hill, 1998.

[16] L. Babai. Trading Group Theory for Randomness. In *17th ACM Symposium on the Theory of Computing*, pages 421–429, 1985.

[17] L. Babai, L. Fortnow, and C. Lund. Non-Deterministic Exponential Time has Two-Prover Interactive Protocols. *Computational Complexity*, Vol. 1, No. 1, pages 3–40, 1991. Preliminary version in *31st FOCS*, 1990.

[18] L. Babai, L. Fortnow, L. Levin, and M. Szegedy. Checking Computations in Polylogarithmic Time. In *23rd ACM Symposium on the Theory of Computing*, pages 21–31, 1991.

[19] L. Babai, L. Fortnow, N. Nisan and A. Wigderson. BPP has Subexponential Time Simulations unless EXPTIME has Publishable Proofs. *Complexity Theory*, Vol. 3, pages 307–318, 1993.

[20] L. Babai and S. Moran. Arthur-Merlin Games: A Randomized Proof System and a Hierarchy of Complexity Classes. *Journal of Computer and System Science*, Vol. 36, pp. 254–276, 1988.

[21] E. Bach and J. Shallit. *Algorithmic Number Theory* (Volume I: Efficient Algorithms). MIT Press, 1996.

[22] B. Barak. Non-Black-Box Techniques in Crypptography. PhD Thesis, Weizmann Institute of Science, 2004.

[23] W. Baur and V. Strassen. The Complexity of Partial Derivatives. *Theor. Comput. Sci.* 22, pages 317–330, 1983.

[24] P. Beame and T. Pitassi. Propositional Proof Complexity: Past, Present, and Future. In *Bulletin of the European Association for Theoretical Computer Science*, Vol. 65, June 1998, pp. 66–89.

[25] A. Beimel, Y. Ishai, E. Kushilevitz, and J.F. Raymond. Breaking the $O(n^{1/(2k-1)})$ barrier for information-theoretic private information retrieval. In *43rd IEEE Symposium on Foundations of Computer Science*, pages 261–270, 2002.

[26] M. Bellare, O. Goldreich, and E. Petrank. Uniform Generation of NP-witnesses using an NP-oracle. *Information and Computation*, Vol. 163, pages 510–526, 2000.

[27] M. Bellare, O. Goldreich and M. Sudan. Free Bits, PCPs and Non-Approximability – Towards Tight Results. *SIAM Journal on Computing*, Vol. 27, No. 3, pages 804–915, 1998. Extended abstract in *36th FOCS*, 1995.

[28] 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 (2), pages 193–219, 1992.

[29] A. Ben-Dor and S. Halevi. In *2nd Israel Symp. on Theory of Computing and Systems*, IEEE Computer Society Press, pages 108-117, 1993.

[30] 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

[31] 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.

[32] 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.

[33] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan. Robust PCPs of proximity, Shorter PCPs and Applications to Coding. In *36th ACM Symposium on the Theory of Computing*, pages 1–10, 2004. Full version in *ECCC*, TR04-021, 2004.

[34] E. Ben-Sasson and M. Sudan. Simple PCPs with Poly-log Rate and Query Complexity. *ECCC*, TR04-060, 2004.

[35] L. Berman and J. Hartmanis. On isomorphisms and density of NP and other complete sets. *SIAM Journal on Computing*, Vol. 6 (2), 1977, pages 305–322. Extended abstract in *8th STOC*, 1976.

[36] M. Blum. A Machine-Independent Theory of the Complexity of Recursive Functions. *Journal of the ACM*, Vol. 14 (2), pages 290–305, 1967.

[37] 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 FOCS*, 1982.

[38] M. Blum, M. Luby and R. Rubinfeld. Self-Testing/Correcting with Applications to Numerical Problems. *Journal of Computer and System Science*, Vol. 47, No. 3, pages 549–595, 1993.

[39] A. Bogdanov, K. Obata, and L. Trevisan. A lower bound for testing 3-colorability in bounded-degree graphs. In *43rd IEEE Symposium on Foundations of Computer Science*, pages 93–102, 2002.

[40] A. Bogdanov and L. Trevisan. On worst-case to average-case reductions for NP problems. In *Proc. 44th IEEE Symposium on Foundations of Computer Science*, pages 308–317, 2003.

[41] A. Bogdanov and L. Trevisan. Average-case complexity: a survey. In preparation, 2005.

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

[43] R. Boppana and M. Sipser. The complexity of finite functions. In *Handbook of Theoretical Computer Science: Volume A – Algorithms and Complexity*, J. van Leeuwen editor, MIT Press/Elsevier, 1990, pages 757–804.

[44] A. Borodin. Computational Complexity and the Existence of Complexity Gaps. *Journal of the ACM*, Vol. 19 (1), pages 158–174, 1972.

[45] A. Borodin. On Relating Time and Space to Size and Depth. *SIAM Journal on Computing*, Vol. 6 (4), pages 733–744, 1977.

[46] 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 FOCS*, 1986.

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

[48] G.J. Chaitin. On the Length of Programs for Computing Finite Binary Sequences. *Journal of the ACM*, Vol. 13, pages 547–570, 1966.

[49] A.K. Chandra, D.C. Kozen and L.J. Stockmeyer. Alternation. *Journal of the ACM*, Vol. 28, pages 114–133, 1981.

[50] 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.

[51] B. Chor and O. Goldreich. On the Power of Two–Point Based Sampling. *Jour. of Complexity*, Vol 5, 1989, pages 96–106. Preliminary version dates 1985.

[52] B. Chor and O. Goldreich. Unbiased Bits from Sources of Weak Randomness and Probabilistic Communication Complexity. *SIAM Journal on Computing*, Vol. 17, No. 2, pages 230–261, 1988.

[53] A. Church. An Unsolvable Problem of Elementary Number Theory. *Amer. J. of Math.*, Vol. 58, pages 345–363, 1936.

[54] A. Cobham. The Intristic Computational Difficulty of Functions. In *Proc. 1964 Iternational Congress for Logic Methodology and Philosophy of Science*, pages 24–30, 1964.

[55] S.A. Cook. The Complexity of Theorem Proving Procedures. In *3rd ACM Symposium on the Theory of Computing*, pages 151–158, 1971.

[56] S.A. Cook. A overview of Computational Complexity. Turing Award Lecture. *CACM*, Vol. 26 (6), pages 401–408, 1983.

[57] S.A. Cook. A Taxonomy of Problems with Fast Parallel Algorithms. *Information and Control*, Vol. 64, pages 2–22, 1985.

[58] S.A. Cook and R.A. Reckhow. Stephen A. Cook, Robert A. Reckhow: The Relative Efficiency of Propositional Proof Systems. *J. of Symbolic Logic*, Vol. 44 (1), pages 36–50, 1979.

[59] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9, pages 251–280, 1990.

[60] T.M. Cover and G.A. Thomas. *Elements of Information Theory.* John Wiley & Sons, Inc., New-York, 1991.

[61] P. Crescenzi and V. Kann. A compendium of NP Optimization problems. Available at `http://www.nada.kth.se/~viggo/wwwcompendium/`

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

[63] I. Dinur. The PCP Theorem by Gap Amplification. *ECCC*, TR05-046, 2005.

[64] I. Dinur and O. Reingold. Assignment-testers: Towards a combinatorial proof of the PCP-Theorem. In *45th IEEE Symposium on Foundations of Computer Science*, pages 155–164, 2004.

[65] I. Dinur and S. Safra. The importance of being biased. In *34th ACM Symposium on the Theory of Computing*, pages 33–42, 2002.

[66] J. Edmonds. Paths, Trees, and Flowers. *Canad. J. Math.*, Vol. 17, pages 449–467, 1965.

[67] S. Even. *Graph Algorithms.* Computer Science Press, 1979.

[68] 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.

[69] U. Feige, S. Goldwasser, L. Lovász and S. Safra. On the Complexity of Approximating the Maximum Size of a Clique. Unpublished manuscript, 1990.

[70] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Approximating Clique is almost NP-complete. *Journal of the ACM*, Vol. 43, pages 268–292, 1996. Preliminary version in *32nd FOCS*, 1991.

[71] U. Feige, D. Lapidot, and A. Shamir. Multiple Non-Interactive Zero-Knowledge Proofs Under General Assumptions. *SIAM Journal on Computing*, Vol. 29 (1), pages 1–28, 1999.

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

[73] E. Fischer. The art of uninformed decisions: A primer to property testing. *Bulletin of the European Association for Theoretical Computer Science*, Vol. 75, pages 97–126, 2001.

[74] G.D. Forney. *Concatenated Codes.* MIT Press, Cambridge, MA 1966.

[75] L. Fortnow, R. Lipton, D. van Melkebeek, and A. Viglas. Time-space lower bounds for satisfiability. *Journal of the ACM*, Vol. 52 (6), pages 835–865, November 2005.

[76] L. Fortnow, J. Rompel and M. Sipser. On the power of multi-prover interactive protocols. In *3rd IEEE Symp. on Structure in Complexity Theory*, pages 156–161, 1988. See errata in *5th IEEE Symp. on Structure in Complexity Theory*, pages 318–319, 1990.

[77] S. Fortune. A Note on Sparse Complete Sets. *SIAM Journal on Computing*, Vol. 8, pages 431–433, 1979.

[78] M. Fürer, O. Goldreich, Y. Mansour, M. Sipser, and S. Zachos. On Completeness and Soundness in Interactive Proof Systems. *Advances in Computing Research: a research annual*, Vol. 5 (Randomness and Computation, S. Micali, ed.), pages 429–442, 1989.

[79] M.L. Furst, J.B. Saxe, and M. Sipser. Parity, Circuits, and the Polynomial-Time Hierarchy. *Mathematical Systems Theory*, Vol. 17 (1), pages 13–27, 1984. Preliminary version in *22nd FOCS*, 1981.

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

[81] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.

[82] D. Gillman. A chernoff bound for random walks on expander graphs. In *34th IEEE Symposium on Foundations of Computer Science*, pages 680–691, 1993.

[83] O. Goldreich. *Foundation of Cryptography – Class Notes*. Computer Science Dept., Technion, Israel, Spring 1989. Superseded by [87, 88].

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

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

[86] O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*. Algorithms and Combinatorics series (Vol. 17), Springer, 1999.

[87] O. Goldreich. *Foundation of Cryptography: Basic Tools*. Cambridge University Press, 2001.

[88] O. Goldreich. *Foundation of Cryptography: Basic Applications*. Cambridge University Press, 2004.

[89] O. Goldreich. Short Locally Testable Codes and Proofs (Survey). *ECCC*, TR05-014, 2005.

[90] O. Goldreich. On Promise Problems (a survey in memory of Shimon Even [1935-2004]). *ECCC*, TR05-018, 2005.

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

[92] O. Goldreich, S. Goldwasser, and A. Nussboim. On the Implementation of Huge Random Objects. In *44th IEEE Symposium on Foundations of Computer Science*, pages 68–79, 2002.

[93] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, pages 653–750, July 1998.

[94] O. Goldreich and H. Krawczyk. On the Composition of Zero-Knowledge Proof Systems. *SIAM Journal on Computing*, Vol. 25, No. 1, February 1996, pages 169–192. Preliminary version in *17th ICALP*, 1990.

[95] 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.

[96] 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. 3, pages 691–729, 1991. Preliminary version in *27th FOCS*, 1986.

[97] 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.

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

[99] O. Goldreich and D. Ron. Property testing in bounded degree graphs. *Algorithmica*, pages 302–343, 2002.

[100] O. Goldreich and D. Ron. A sublinear bipartite tester for bounded degree graphs. *Combinatorica*, Vol. 19 (3), pages 335–373, 1999.

[101] O. Goldreich, R. Rubinfeld and M. Sudan. Learning polynomials with queries: the highly noisy case. *SIAM J. Discrete Math.*, Vol. 13 (4), pages 535–570, 2000.

[102] O. Goldreich, S. Vadhan and A. Wigderson. On interactive proofs with a laconic provers. *Computational Complexity*, Vol. 11, pages 1–53, 2002.

[103] O. Goldreich and A. Wigderson. Computational Complexity. In *The Princeton Companion to Mathematics*, to appear.

[104] 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 STOC*, 1982.

[105] 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 STOC*, 1985. Earlier versions date to 1982.

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

[107] 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. Extended abstract in *18th STOC*, 1986.

[108] S.W. Golomb. *Shift Register Sequences.* Holden-Day, 1967. (Aegean Park Press, revised edition, 1982.)

[109] J. Hartmanis and R.E. Stearns. On the Computational Complexity of of Algorithms. *Transactions of the AMS*, Vol. 117, pages 285–306, 1965.

[110] J. Håstad. Almost Optimal Lower Bounds for Small Depth Circuits. *Advances in Computing Research: a research annual*, Vol. 5 (Randomness and Computation, S. Micali, ed.), pages 143–170, 1989. Extended abstract in *18th STOC*, pages 6–20, 1986.

[111] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, Vol. 182, pages 105–142, 1999. Preliminary versions in *28th STOC* (1996) and *37th FOCS* (1996).

[112] J. Håstad. Getting optimal in-approximability results. In *29th ACM Symposium on the Theory of Computing*, pages 1–10, 1997.

[113] J. Håstad, R. Impagliazzo, L.A. Levin and M. Luby. A Pseudorandom Generator from any One-way Function. *SIAM Journal on Computing*, Volume 28, Number 4, pages 1364–1396, 1999. Preliminary versions by Impagliazzo *et. al.* in *21st STOC* (1989) and Håstad in *22nd STOC* (1990).

[114] J. Håstad and S. Khot. Query efficient PCPs with pefect completeness. In *42nd IEEE Symposium on Foundations of Computer Science*, pages 610–619, 2001.

[115] A. Healy, S. Vadhan and E. Viola. Using nondeterminism to amplify hardness. In *36th ACM Symposium on the Theory of Computing*, pages 192–201, 2004.

[116] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[117] D. Hochbaum (ed.). *Approximation Algorithms for NP-Hard Problems.* PWS, 1996.

[118] N. Immerman. Nondeterministic Space is Closed Under Complementation. *SIAM Journal on Computing*, Vol. 17, pages 760–778, 1988.

[119] R. Impagliazzo. Hard-core Distributions for Somewhat Hard Problems. In *36th IEEE Symposium on Foundations of Computer Science*, pages 538–545, 1995.

[120] R. Impagliazzo and L.A. Levin. No Better Ways to Generate Hard NP Instances than Picking Uniformly at Random. In *31st IEEE Symposium on Foundations of Computer Science*, pages 812–821, 1990.

[121] 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.

[122] R. Impagliazzo and A. Wigderson. Randomness vs Time: Derandomization under a Uniform Assumption. *Journal of Computer and System Science*, Vol. 63 (4), pages 672-688, 2001.

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

[124] M. Jerrum, A. Sinclair, and E. Vigoda. A Polynomial-Time Approximation Algorithm for the Permanent of a Matrix with Non-Negative Entries. *Journal of the ACM*, Vol. 51 (4), pages 671–697, 2004.

[125] M. Jerrum, L. Valiant, and V.V. Vazirani. Random Generation of Combinatorial Structures from a Uniform Distribution. *Theoretical Computer Science*, Vol. 43, pages 169–188, 1986.

[126] N. Kahale, Eigenvalues and Expansion of Regular Graphs. *Journal of the ACM*, Vol. 42 (5), pages 1091–1106, September 1995.

[127] R. Kannan, H. Venkateswaran, V. Vinay, and A.C. Yao. A Circuit-based Proof of Toda's Theorem. *Information and Computation*, Vol. 104 (2), pages 271–276, 1993.

[128] R.M. Karp. Reducibility among Combinatorial Problems. In *Complexity of Computer Computations*, R.E. Miller and J.W. Thatcher (eds.), Plenum Press, pages 85–103, 1972.

[129] R.M. Karp and R.J. Lipton. Some connections between nonuniform and uniform complexity classes. In *12th ACM Symposium on the Theory of Computing*, pages 302-309, 1980.

[130] R.M. Karp and M. Luby. Monte-Carlo algorithms for enumeration and reliability problems. In *24th IEEE Symposium on Foundations of Computer Science*, pages 56-64, 1983.

[131] R.M. Karp and V. Ramachandran: Parallel Algorithms for Shared-Memory Machines. In *Handbook of Theoretical Computer Science, Vol A: Algorithms and Complexity*, 1990.

[132] M. Karchmer and A. Wigderson. Monotone Circuits for Connectivity Require Super-logarithmic Depth. *SIAM J. Discrete Math.*, Vol. 3 (2), pages 255–265, 1990. Preliminary version in *20th STOC*, 1988.

[133] M.J. Kearns and U.V. Vazirani. *An introduction to Computational Learning Theory.* MIT Press, 1994.

[134] S. Khot and O. Regev. Vertex Cover Might be Hard to Approximate to within $2-\varepsilon$. In *18th IEEE Conference on Computational Complexity*, pages 379–386, 2003.

[135] V.M. Khrapchenko. A method of determining lower bounds for the complexity of Pi-schemes. In *Matematicheskie Zametki* 10 (1),pages 83–92, 1971 (in Russian). English translation in *Mathematical Notes of the Academy of Sciences of the USSR* 10 (1) 1971, pages 474–479.

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

[137] D.E. Knuth. *The Art of Computer Programming*, Vol. 2 (*Seminumerical Algorithms*). Addison-Wesley Publishing Company, Inc., 1969 (first edition) and 1981 (second edition).

[138] A. Kolmogorov. Three Approaches to the Concept of "The Amount Of Information". *Probl. of Inform. Transm.*, Vol. 1/1, 1965.

[139] E. Kushilevitz and N. Nisan. *Communication Complexity.* Cambridge University Press, 1996.

[140] R.E. Ladner. On the Structure of Polynomial Time Reducibility. *Journal of the ACM*, Vol. 22, 1975, pages 155–171.

[141] C. Lautemann. BPP and the Polynomial Hierarchy. *Information Processing Letters*, 17, pages 215–217, 1983.

[142] F.T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[143] L.A. Levin. Universal Search Problems. *Problemy Peredaci Informacii 9*, pages 115–116, 1973. Translated in *problems of Information Transmission 9*, pages 265–266.

[144] L.A. Levin. Randomness Conservation Inequalities: Information and Independence in Mathematical Theories. *Information and Control*, Vol. 61, pages 15–37, 1984.

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

[146] L.A. Levin. Fundamentals of Computing. *SIGACT News*, Education Forum, special 100-th issue, Vol. 27 (3), pages 89–110, 1996.

[147] M. Li and P. Vitanyi. *An Introduction to Kolmogorov Complexity and its Applications.* Springer Verlag, August 1993.

[148] C.-J. Lu, O. Reingold, S. Vadhan, and A. Wigderson. Extractors: optimal up to constant factors. In *35th ACM Symposium on the Theory of Computing*, pages 602–611, 2003.

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

[150] M. Luby and A. Wigderson. Pairwise Independence and Derandomization. TR-95-035, International Computer Science Institute (ICSI), Berkeley, 1995. ISSN 1075-4946.

[151] 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. Preliminary version in *31st FOCS*, 1990.

[152] F. MacWilliams and N. Sloane. *The theory of error-correcting codes.* North-Holland, 1981.

[153] G.A. Margulis. Explicit Construction of Concentrators. (In Russian.) *Prob. Per. Infor.*, Vol. 9 (4), pages 71–80, 1973. English translation in *Problems of Infor. Trans.*, pages 325–332, 1975.

[154] S. Micali. Computationally Sound Proofs. *SIAM Journal on Computing*, Vol. 30 (4), pages 1253–1298, 2000. Preliminary version in *35th FOCS*, 1994.

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

[156] P.B. Miltersen and N.V. Vinodchandran. Derandomizing Arthur-Merlin Games using Hitting Sets. *Journal of Computational Complexity*, to appear. Preliminary version in *40th FOCS*, 1999.

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

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

[159] J. Naor and M. Naor. Small-bias Probability Spaces: Efficient Constructions and Applications. *SIAM Journal on Computing*, Vol 22, 1993, pages 838–856.

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

[161] N. Nisan. Pseudorandom bits for constant depth circuits. *Combinatorica*, Vol. 11 (1), pages 63–70, 1991.

[162] N. Nisan. Pseudorandom Generators for Space Bounded Computation. *Combinatorica*, Vol. 12 (4), pages 449–461, 1992.

[163] N. Nisan. $\mathcal{RL} \subseteq \mathcal{SC}$. *Journal of Computational Complexity*, Vol. 4, pages 1-11, 1994.

[164] N. Nisan and A. Wigderson. Hardness vs Randomness. *Journal of Computer and System Science*, Vol. 49, No. 2, pages 149–167, 1994.

[165] N. Nisan and D. Zuckerman. Randomness is Linear in Space. *Journal of Computer and System Science*, Vol. 52 (1), pages 43–52, 1996.

[166] C.H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.

[167] C.H. Papadimitriou and M. Yannakakis. Optimization, Approximation, and Complexity Classes. In *20th ACM Symposium on the Theory of Computing*, pages 229–234, 1988.

[168] N. Pippenger and M.J. Fischer. Relations among complexity measures. *Journal of the ACM*, Vol. 26 (2), pages 361–381, 1979.

[169] E. Post. A Variant of a Recursively Unsolvable Problem. *Bull. AMS*, Vol. 52, pages 264–268, 1946.

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

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

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

[173] R. Raz. A Parallel Repetition Theorem. *SIAM Journal on Computing*, Vol. 27 (3), pages 763–803, 1998. Extended abstract in *27th STOC*, 1995.

[174] R. Raz and A. Wigderson. Monotone Circuits for Matching Require Linear Depth. *Journal of the ACM*, Vol. 39 (3), pages 736–744, 1992. Preliminary version in *22nd STOC*, 1990.

[175] A. Razborov. Lower bounds for the monotone complexity of some Boolean functions. In *Doklady Akademii Nauk SSSR*, Vol. 281, No. 4, 1985, pages 798–801. English translation in *Soviet Math. Doklady*, 31, pages 354–357, 1985.

[176] A. Razborov. Lower bounds on the size of bounded-depth networks over a complete basis with logical addition. In *Matematicheskie Zametki*, Vol. 41, No. 4, pages 598–607, 1987. English translation in *Mathematical Notes of the Academy of Sci. of the USSR*, Vol. 41 (4), pages 333–338, 1987.

[177] A.R. Razborov and S. Rudich. Natural Proofs. *Journal of Computer and System Science*, Vol. 55 (1), pages 24–35, 1997.

[178] O. Reingold. Undirected ST-Connectivity in Log-Space. In *37th ACM Symposium on the Theory of Computing*, pages 376–385, 2005.

[179] O. Reingold, S. Vadhan, and A. Wigderson. Entropy Waves, the Zig-Zag Graph Product, and New Constant-Degree Expanders and Extractors. *Annals of Mathematics*, Vol. 155 (1), pages 157–187, 2001. Preliminary version in *41st FOCS*, pages 3–13, 2000.

[180] H.G. Rice. Classes of Recursively Enumerable Sets and their Decision Problems. *Trans. AMS*, Vol. 89, pages 25–59, 1953.

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

[182] D. Ron. Property testing. In *Handbook on Randomization, Volume II*, pages 597–649, 2001. (Editors: S. Rajasekaran, P.M. Pardalos, J.H. Reif and J.D.P. Rolim.)

[183] R. Rubinfeld and M. Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal on Computing*, Vol. 25 (2), pages 252–271, 1996.

[184] M. Saks and S. Zhou. RSPACE(S) $\subseteq$ DSPACE($S^{3/2}$). In 36th *IEEE Symposium on Foundations of Computer Science*, pages 344–353, 1995.

[185] W.J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *JCSS*, Vol. 4 (2), pages 177-192, 1970.

[186] A. Selman. On the structure of NP. *Notices Amer. Math. Soc.*, Vol. 21 (6), page 310, 1974.

[187] R. Shaltiel. Recent Developments in Explicit Constructions of Extractors. *Bulletin of the EATCS 77*, pages 67–95, 2002.

[188] R. Shaltiel and C. Umans. Simple Extractors for All Min-Entropies and a New Pseudo-Random Generator. In *42nd IEEE Symposium on Foundations of Computer Science*, pages 648–657, 2001.

[189] C.E. Shannon. A Symbolic Analysis of Relay and Switching Circuits. *Trans. American Institute of Electrical Engineers*, Vol. 57, pages 713–723, 1938.

[190] C.E. Shannon. A mathematical theory of communication. *Bell Sys. Tech. Jour.*, Vol. 27, pages 623–656, 1948.

[191] C.E. Shannon. Communication Theory of Secrecy Systems. *Bell Sys. Tech. Jour.*, Vol. 28, pages 656–715, 1949.

[192] A. Shamir. IP = PSPACE. *Journal of the ACM*, Vol. 39, No. 4, pages 869–877, 1992. Preliminary version in *31st FOCS*, 1990.

[193] A. Shpilka. Lower Bounds for Matrix Product. *SIAM Journal on Computing*, pages 1185-1200, 2003.

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

[195] M. Sipser. *Introduction to the Theory of Computation.* PWS Publishing Company, 1997.

[196] R. Smolensky. Algebraic Methods in the Theory of Lower Bounds for Boolean Circuit Complexity. In *19th ACM Symposium on the Theory of Computing* pages 77–82, 1987.

[197] R.J. Solomonoff. A Formal Theory of Inductive Inference. *Information and Control*, Vol. 7/1, pages 1–22, 1964.

[198] 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.

[199] D.A. Spielman. *Advanced Complexity Theory*, Lectures 10 and 11. Notes (by D. Lewin and S. Vadhan), March 1997. Available from `http://www.cs.yale.edu/homes/spielman/AdvComplexity/1998/` as `lect10.ps` and `lect11.ps`.

[200] L.J. Stockmeyer. The Polynomial-Time Hierarchy. *Theoretical Computer Science*, Vol. 3, pages 1–22, 1977.

[201] L. Stockmeyer. The Complexity of Approximate Counting. In *15th ACM Symposium on the Theory of Computing*, pages 118–126, 1983.

[202] V. Strassen. Algebraic Complexity Theory. In *Handbook of Theoretical Computer Science: Volume A – Algorithms and Complexity*, J. van Leeuwen editor, MIT Press/Elsevier, 1990, pages 633–672.

[203] M. Sudan. Decoding of Reed Solomon codes beyond the error-correction bound. *Journal of Complexity*, Vol. 13 (1), pages 180–193, 1997.

[204] M. Sudan. Algorithmic introduction to coding theory. Lecture notes, Available from `http://theory.csail.mit.edu/~madhu/FT01/`, 2001.

[205] , M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the XOR Lemma. *Journal of Computer and System Science*, Vol. 62, No. 2, pages 236–266, 2001.

[206] R. Szelepcsenyi. A Method of Forced Enumeration for Nondeterministic Automata. *Acta Informatica*, Vol. 26, pages 279–284, 1988.

[207] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, Vol. 20 (5), pages 865–877, 1991.

[208] B.A. Trakhtenbrot. A Survey of Russian Approaches to *Perebor* (Brute Force Search) Algorithms. *Annals of the History of Computing*, Vol. 6 (4), pages 384–398, 1984.

[209] L. Trevisan. Constructions of Near-Optimal Extractors Using Pseudo-Random Generators. In *31st ACM Symposium on the Theory of Computing*, pages 141–148, 1998.

[210] V. Trifonov. An $O(\log n \log\log n)$ Space Algorithm for Undirected st-Connectivity. In *37th ACM Symposium on the Theory of Computing*, pages 623–633, 2005.

[211] C.E. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proc. Londom Mathematical Soceity*, Ser. 2, Vol. 42, pages 230–265, 1936. A Correction, *ibid.*, Vol. 43, pages 544–546.

[212] C. Umans. Pseudo-random generators for all hardness. *Journal of Computer and System Science*, Vol. 67 (2), pages 419–440, 2003.

[213] S. Vadhan. A Study of Statistical Zero-Knowledge Proofs. PhD Thesis, Department of Mathematics, MIT, 1999. Available from `http://www.eecs.harvard.edu/~salil/papers/phdthesis-abs.html`.

[214] S. Vadhan. An Unconditional Study of Computational Zero Knowledge. In *45th IEEE Symposium on Foundations of Computer Science*, pages 176–185, 2004.

[215] L.G. Valiant. The Complexity of Computing the Permanent. *Theoretical Computer Science*, Vol. 8, pages 189–201, 1979.

[216] L.G. Valiant. A theory of the learnable. *CACM*, Vol. 27/11, pages 1134–1142, 1984.

[217] L.G. Valiant and V.V. Vazirani. NP Is as Easy as Detecting Unique Solutions. *Theoretical Computer Science*, Vol. 47 (1), pages 85–93, 1986.

[218] J. von Neumann, First Draft of a Report on the EDVAC, 1945. Contract No. W-670-ORD-492, Moore School of Electrical Engineering, Univ. of Penn., Philadelphia. Reprinted (in part) in *Origins of Digital Computers: Selected Papers*, Springer-Verlag, Berlin Heidelberg, pages 383–392, 1982.

[219] J. von Neumann, Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen*, 100, pages 295–320, 1928.

[220] I. Wegener. *The Complexity of Boolean Functions.* Wiley-Teubner, 1987.

[221] I. Wegener. *Branching Programs and Binary Decision Diagrams – Theory and Applications.* SIAM Monographs on Discrete Mathematics and Applications, 2000.

[222] A. Wigderson. The amazing power of pairwise independence. In *26th ACM Symposium on the Theory of Computing*, pages 645–647, 1994.

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

[224] A.C. Yao. Separating the Polynomial-Time Hierarchy by Oracles. In *26th IEEE Symposium on Foundations of Computer Science*, pages 1-10, 1985.

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

[226] D. Zuckerman. Simulating BPP Using a General Weak Random Source. *Algorithmica*, Vol. 16, pages 367–391, 1996.

[227] D. Zuckerman. Randomness-Optimal Oblivious Sampling. *Journal of Random Structures and Algorithms*, Vol. 11, Nr. 4, December 1997, pages 345–367.