Appendix A

Glossary of Complexity Classes

Summary: This glossary includes self-contained definitions of most complexity classes mentioned in the book. Needless to say, the glossary offers a very minimal discussion of these classes and the reader is referred to the main text for further discussion. The items are organized by topics rather than by alphabetic order. Specifically, the glossary is partitioned into two parts, dealing separately with complexity classes that are defined in terms of algorithms and their resources (i.e., time and space complexity of Turing machines) and complexity classes defined in terms of non-uniform circuits (and referring to their size and depth). The algorithmic classes include time-complexity based classes (such as $\mathcal{P}$, $\mathcal{NP}$, co$\mathcal{NP}$, $\mathcal{BPP}$, $\mathcal{RP}$, co$\mathcal{RP}$, $\mathcal{PH}$, $\mathcal{E}$, $\mathcal{EXP}$ and $\mathcal{NEXP}$) and the space complexity classes $\mathcal{L}$, $\mathcal{NL}$, $\mathcal{RL}$ and $\mathcal{PSPACE}$. The non-uniform classes include the circuit classes $\mathcal{P}$/poly as well as $\mathcal{NC}^k$ and $\mathcal{AC}^k$.

Definitions (and basic results) regarding many other complexity classes are available at the constantly evolving *Complexity Zoo* [1].

A.1 Preliminaries

Complexity classes are sets of computational problems, where each class contains problems that can be solved with specific computational resources. To define a complexity class one specifies a model of computation, a complexity measure (like time or space), which is always measured as a function of the input length, and a bound on the complexity (of problems in the class).

We follow the tradition of focusing on decision problems, but refer to these problems using the terminology of promise problems (see Section 2.4.1). That is, we will refer to the problem of distinguishing inputs in Π_{yes} from inputs in Π_{no}, and denote the corresponding decision problem by $\Pi = (\Pi_{\rm yes}, \Pi_{\rm no})$. Standard decision problems are viewed as a special case in which $\Pi_{\rm yes} \cup \Pi_{\rm no} = \{0,1\}^*$, and the standard formulation of complexity classes is obtained by postulating that this is the case. We refer to this case as the case of a trivial promise.

The prevailing model of computation is that of Turing machines. This model captures the notion of (uniform) algorithms (see Section 1.2.3). Another important model is the one of non-uniform circuits (see Section 1.2.4). The term uniformity refers to whether the algorithm is the same one for every input length or whether a different "algorithm" (or rather a "circuit") is considered for each input length.

We focus on natural complexity classes, obtained by considering natural complexity measures and bounds. Typically, these classes contain natural computational problems (which are defined in Appendix G). Furthermore, almost all of these classes can be "characterized" by natural problems, which capture every problem in the class. Such problems are called complete for the class, which means that they are in the class and every problem in the class can be "easily" reduced to them, where "easily" means that the reduction takes less resources than whatever seems to be requires for solving each individual problem in the class. Thus, any efficient algorithm for a complete problem implies an algorithm of similar efficiency for *all* problems in the class.

Organization: The glossary is organized by topics (rather than by alphabetic order of the various items). Specifically, we partition the glossary to classes defined in terms of algorithmic resources (i.e., time and space complexity of Turing machines) and classes defined in terms of circuit (size and depth). The former (algorithm-based) classes are reviewed in Section A.2, while the latter (circuit-based) classes are reviewed in Section A.3.

A.2 Algorithm-based classes

The two main complexity measures considered in the context of (uniform) algorithms are the number of steps taken by the algorithm (i.e., its time complexity) and the amount of "memory" or "work-space" consumed by the computation (i.e., its space complexity). We review the time complexity based classes $\mathcal{P}$, $\mathcal{NP}$, co$\mathcal{NP}$, $\mathcal{BPP}$, $\mathcal{RP}$, co$\mathcal{RP}$, $\mathcal{ZPP}$, $\mathcal{PH}$, $\mathcal{E}$, $\mathcal{EXP}$ and $\mathcal{NEXP}$ as well as the space complexity classes $\mathcal{L}$, $\mathcal{NL}$, $\mathcal{RL}$ and $\mathcal{PSPACE}$.

By prepending the name of a complexity class (of decision problems) with the prefix "co" we mean the class of complement problems; that is, the problem $\Pi = (\Pi_{\rm yes}, \Pi_{\rm no})$ is in co$\mathcal{C}$ if and only if $(\Pi_{\rm no}, \Pi_{\rm yes})$ is in $\mathcal{C}$. Specifically, deciding membership in the set S is in the class co$\mathcal{C}$ if and only if deciding membership in the set $\{0,1\}^* \setminus S$ is in the class $\mathcal{C}$. Thus, the definition of co$\mathcal{NP}$ and co$\mathcal{RP}$ can be easily derived from the definitions of $\mathcal{NP}$ and $\mathcal{RP}$, respectively. Complexity classes defined in terms of symmetric acceptance criteria (e.g., deterministic and two-sided error randomized classes) are trivially closed under complementation (e.g., co$\mathcal{P} = \mathcal{P}$ and co$\mathcal{BPP} = \mathcal{BPP}$) and so we do not present their "co"-classes.

In other cases (most notably $\mathcal{NL}$), the closure property is highly non-trivial and we comment about it.

A.2.1 Time complexity classes

We start with classes that are closely related to polynomial-time computations (i.e., $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{BPP}$, $\mathcal{RP}$ and $\mathcal{ZPP}$), and latter consider the classes $\mathcal{PH}$, $\mathcal{E}$, $\mathcal{EXP}$ and $\mathcal{NEXP}$.

A.2.1.1 Classes closely related to polynomial time

The most prominent complexity classes are $\mathcal{P}$ and $\mathcal{NP}$, which are extensively discussed in Section 2.1. We also consider classes related to randomized polynomial-time, which are discussed in Section 6.1.

P and NP. The class $\mathcal{P}$ consists of all decision problem that can be solved in (deterministic) polynomial-time. A decision problem $\Pi = (\Pi_{\rm yes}, \Pi_{\rm no})$ is in $\mathcal{NP}$ if there exists a polynomial p and a (deterministic) polynomial-time algorithm V such that the following two conditions hold

1. For every $x \in \Pi_{\rm yes}$ there exists $y \in \{0,1\}^{p(|x|)}$ such that $V(x,y) = 1$.
2. For every $x \in \Pi_{\rm no}$ and every $y \in \{0,1\}^*$ it holds that $V(x,y) = 0$.

A string y satisfying Condition 1 is called an NP-witness (for x). Clearly, $\mathcal{P} \subseteq \mathcal{NP}$.

Reductions and NP-completeness (NPC). A problem is $\mathcal{NP}$-complete if it is in $\mathcal{NP}$ and every problem in $\mathcal{NP}$ is polynomial-time reducible to it, where polynomial-time reducibility is defined and discussed in Section 2.2. Loosely speaking, a polynomial-time reduction of problem Π to problem Π' is a polynomial-time algorithm that solves Π by making queries to a subroutine that solves problem Π', where the running-time of the subroutine is not counted in the algorithm's time complexity. Typically, NP-completeness is defined while restricting the reduction to make a single query and output its answer. Such a reduction, called a Karp-reduction, is represented by a polynomial-time computable mapping that maps yes-instances of Π to yes-instances of Π' (and no-instances of Π to no-instances of Π'). Hundreds of NP-complete problems are listed in [85].

Probabilistic polynomial-time (BPP, RP and ZPP). A decision problem $\Pi = (\Pi_{\rm yes}, \Pi_{\rm no})$ is in $\mathcal{BPP}$ if there exists a probabilistic polynomial-time algorithm A such that the following two conditions hold

1. For every $x \in \Pi_{\rm yes}$ it holds that $\mathsf{Pr}[A(x)=1] \geq 2/3$.
2. For every $x \in \Pi_{\rm no}$ it holds that $\mathsf{Pr}[A(x)=0] \geq 2/3$.

That is, the algorithm has two-sided error probability (of 1/3), which can be further reduced by repetitions. We stress that due to the two-sided error probability of $\mathcal{BPP}$, it is not known whether or not $\mathcal{BPP}$ is contained in $\mathcal{NP}$. In addition to the two-sided error class $\mathcal{BPP}$, we consider one-sided error and zero-error classes, denoted $\mathcal{RP}$ and $\mathcal{ZPP}$, respectively. A problem $\Pi = (\Pi_{\text{yes}}, \Pi_{\text{no}})$ is in $\mathcal{RP}$ if there exists a probabilistic polynomial-time algorithm A such that the following two conditions hold

1. For every $x \in \Pi_{\text{yes}}$ it holds that $\Pr[A(x)=1] \geq 1/2$.
2. For every $x \in \Pi_{\text{no}}$ it holds that $\Pr[A(x)=0] = 1$.

Again, the error probability can be reduced by repetitions, and thus $\mathcal{RP} \subseteq \mathcal{BPP} \cap \mathcal{NP}$. A problem $\Pi = (\Pi_{\text{yes}}, \Pi_{\text{no}})$ is in $\mathcal{ZPP}$ if there exists a probabilistic polynomial-time algorithm A, which may output a special ("don't know") symbol $\perp$, such that the following two conditions hold

1. For every $x \in \Pi_{\text{yes}}$ it holds that $\Pr[A(x) \in \{1, \perp\}] = 1$ and $\Pr[A(x)=1] \geq 1/2$.
2. For every $x \in \Pi_{\text{no}}$ it holds that $\Pr[A(x) \in \{0, \perp\}] = 1$ and $\Pr[A(x)=0] \geq 1/2$.

Note that $\mathcal{P} \subseteq \mathcal{ZPP} = \mathcal{RP} \cap \text{co}\mathcal{RP}$. When defined in terms of promise problems, all the aforementioned randomized classes have complete problems (w.r.t Karp-reductions), but the same is not known when considering only standard decision problems (with trivial promise).

The counting class $\#\mathcal{P}$. Functions in $\#\mathcal{P}$ count the number of solutions to an NP-type search problem (or, equivalently, the number of NP-witnesses for a yes-instance of a decision problem in $\mathcal{NP}$). Formally, a function f is in $\#\mathcal{P}$ if there exists a polynomial p and a (deterministic) polynomial-time algorithm V such that $f(x) = |\{y \in \{0,1\}^{p(|x|)} : V(x,y)=1\}|$. Indeed, p and V are as in the definition of $\mathcal{NP}$, and it follows that deciding membership in the set $\{x : f(x) \geq 1\}$ is in $\mathcal{NP}$. Clearly, $\#\mathcal{P}$ problems are solvable in polynomial space. Surprisingly, the permanent of positive integer matrices is $\#\mathcal{P}$-complete (i.e., it is in $\#\mathcal{P}$ and any function in $\#\mathcal{P}$ is polynomial-time reducible to it).

Interactive proofs. A decision problem $\Pi = (\Pi_{\text{yes}}, \Pi_{\text{no}})$ has an interactive proof system if there exists a polynomial-time strategy V such that the following two conditions hold:

1. For every $x \in \Pi_{\text{yes}}$ there exists a prover strategy P such that the verifier V always accepts after interacting with the prover P on common input x.
2. For every $x \notin \Pi_{\text{no}}$ and every strategy P^*, the verifier V rejects with probability at least $\frac{1}{2}$ after interacting with P^* on common input x.

The corresponding class is denoted $\mathcal{IP}$, and turns out to equal $\mathcal{PSPACE}$. (For further details see Section 9.1.)

A.2.1.2 Other time complexity classes

The classes $\mathcal{E}$ and $\mathcal{EXP}$ corresponding to problems that can be solved (by a deterministic algorithm) in time $2^{O(n)}$ and $2^{\mathrm{poly}(n)}$, respectively, for n-bit long inputs. Clearly, $\mathcal{NP} \subseteq \mathcal{EXP}$. We also mention $\mathcal{NEXP}$, the class of problems that can be solved by a non-deterministic machine in $2^{\mathrm{poly}(n)}$ steps.[1]

In general, one may define a complexity class for every time bound and every type of machine (i.e., deterministic, probabilistic and non-deterministic), but polynomial and exponential bounds seem most natural and very robust. Another robust type of time bounds that is sometimes used is quasi-polynomial time (i.e., $\widetilde{\mathcal{P}}$ denotes the class of problems solvable by deterministic machines of time complexity $\exp(\mathrm{poly}(\log n))$).

The Polynomial-time hierarchy, $\mathcal{PH}$. For any natural number k, the k^{th} level of the polynomial-time hierarchy consists of problems $\Pi = (\Pi_{\mathrm{yes}}, \Pi_{\mathrm{no}})$ such that there a polynomial p and a polynomial-time algorithm V that satisfies the following two requirements:

1. For every $x \in \Pi_{\mathrm{yes}}$ there exists $y_1 \in \{0,1\}^{p(|x|)}$ such that for every $y_2 \in \{0,1\}^{p(|x|)}$ there exists $y_3 \in \{0,1\}^{p(|x|)}$ such that for every $y_4 \in \{0,1\}^{p(|x|)}$ ... it holds that $V(x, y_1, y_2, y_3, y_4, ..., y_k) = 1$. That is, the condition regarding x consists of k alternating quantifiers.

2. For every $x \in \Pi_{\mathrm{no}}$ the foregoing (k-alternating) condition does not hold. That is, for every $y_1 \in \{0,1\}^{p(|x|)}$ there exists $y_2 \in \{0,1\}^{p(|x|)}$ such that for every $y_3 \in \{0,1\}^{p(|x|)}$ there exists $y_4 \in \{0,1\}^{p(|x|)}$ ... it holds that $V(x, y_1, y_2, y_3, y_4, ..., y_k) = 0$.

Such a problem Π is said to be in Σ_k (and $\Pi_k \stackrel{\mathrm{def}}{=} \mathrm{co}\Sigma_k$). Indeed, $\mathcal{NP} = \Sigma_1$ corresponds to the special case where $k = 1$. Interestingly, $\mathcal{PH}$ is polynomial-time reducible to $\#\mathcal{P}$.

A.2.2 Space complexity

When defining space-complexity classes, one counts *only* the space consumed by the actual computation, and *not* the space occupied by the input and output. This is formalized by postulating that the input is read from a read-only device (resp., the output is written on a write-only device). Four important classes of decision problems are defined next.

[1]Alternatively, analogously to the definition of $\mathcal{NP}$, a problem $\Pi = (\Pi_{\mathrm{yes}}, \Pi_{\mathrm{no}})$ is in $\mathcal{NEXP}$ if there exists a polynomial p and a polynomial-time algorithm V such that the two conditions hold

1. For every $x \in \Pi_{\mathrm{yes}}$ there exists $y \in \{0,1\}^{2^{p(|x|)}}$ such that $V(x, y) = 1$.
2. For every $x \in \Pi_{\mathrm{no}}$ and every $y \in \{0,1\}^*$ it holds that $V(x, y) = 0$.

- The class $\mathcal{L}$ consists of problems solvable in logarithmic space. That is, a problem Π is in $\mathcal{L}$ if there exists a standard (i.e., deterministic) algorithm of logarithmic space-complexity for solving Π. This class contains some simple computational problems (e.g., matrix multiplication), and arguably captures the most space-efficient computations. Interestingly, $\mathcal{L}$ contains the problem of deciding connectivity of (undirected) graphs.

- Classes of problems solvable by randomized algorithms of logarithmic space-complexity include $\mathcal{RL}$ and $\mathcal{BPL}$, which are defined analogously to $\mathcal{RP}$ and $\mathcal{BPP}$. That is, $\mathcal{RL}$ corresponds to algorithms with one-sided error probability, whereas $\mathcal{BPL}$ allows two-sided error.

- The class $\mathcal{NL}$ is the non-deterministic analogue of $\mathcal{L}$, and is traditionally defined in terms of non-deterministic machines of logarithmic space-complexity.[2] The class $\mathcal{NL}$ contains the problem of deciding whether there exists a directed path between two given vertexes in a given directed graph. In fact, the latter problem is complete for the class (under logarithmic-space reductions). Interestingly, co$\mathcal{NL}$ equals $\mathcal{NL}$.

- The class $\mathcal{PSPACE}$ consists of problems solvable in polynomial space. This class contains very difficult problems, including the computation of winning strategies for any "efficient 2-party games" (see Section 5.4).

Clearly, $\mathcal{L} \subseteq \mathcal{RL} \subseteq \mathcal{NL} \subseteq \mathcal{P}$ and $\mathcal{NP} \subseteq \mathcal{PSPACE} \subseteq \mathcal{EXP}$.

A.3 Circuit-based classes

We refer the reader to Section 1.2.4 for a definition of Boolean circuits as computing devices. The two main complexity measures considered in the context of (non-uniform) circuits are the number of gates (or wires) in the circuit (i.e., the circuit's size) and the length of the longest directed path from an input to an output (i.e., the circuit's depth).

Throughout this section, when we talk of circuits, we actually refer to families of circuits containing a circuit for each instance length, where the n-bit long instances of the computational problem are handled by the n^{th} circuit in the family. Similarly, when we talk of the size and depth of a circuit, we actually mean the (dependence on n of the) size and depth of the n^{th} circuit in the family.

General polynomial-size circuits (P/poly). The main motivation for the introduction of complexity classes based on (non-uniform) circuits is the development of lower-bounds. For example, the class of problems solvable by polynomial-size circuits, denoted $\mathcal{P}$/poly, is a (strict)[3] super-set of $\mathcal{P}$. Thus, showing that $\mathcal{NP}$ is not contained in $\mathcal{P}$/poly would imply $\mathcal{P} \neq \mathcal{NP}$. For further discussion see

[2]See further discussion of this definition in Section 5.3.

[3]In particular, $\mathcal{P}$/poly contains some decision problems that are not solvable by any uniform algorithm.

Appendix B.2. An alternative definition of $\mathcal{P}$/poly in terms of "machines that take advice" is provided in Section 3.1.2. We mention that if $\mathcal{NP} \subset \mathcal{P}$/poly then $\mathcal{PH} = \Sigma_2$.

The subclasses AC0 and TC0. The class $\mathcal{AC}^0$, discussed in Appendix B.2.3, consists of problems solvable by constant-depth polynomial-size circuits of *unbounded fan-in.* The analogue class that allows also (unbounded fan-in) majority-gates (or, equivalently, threshold-gates) is denoted $\mathcal{TC}^0$.

The subclasses AC and NC. Turning back to the standard basis (of $\neg$, $\vee$ and $\wedge$ gates), for any non-negative integer k, we denote by $\mathcal{NC}^k$ (resp., $\mathcal{AC}^k$) the class of problems solvable by polynomial-size circuits of *bounded fan-in* (resp., unbounded fan-in) having depth $O(\log^k n)$, where n is the input length. Clearly, $\mathcal{NC}^k \subseteq \mathcal{AC}^k \subseteq \mathcal{NC}^{k+1}$. A commonly referred class is $\mathcal{NC} \stackrel{\text{def}}{=} \cup_{k\in\mathbb{N}}\mathcal{NC}^k$.

We mention that the class $\mathcal{NC}^2 \supseteq \mathcal{NL}$ is the habitat of most natural computational problems of Linear Algebra: solving a linear system of equations as well as computing the rank, inverse and determinant of a matrix. The class $\mathcal{NC}^1$ contains all symmetric functions, regular languages as well as word problems for finite groups and monoids. The class $\mathcal{AC}^0$ contains all properties (of finite objects) that are expressible by first-order logic.

Uniformity. The foregoing classes make no reference to the complexity of constructing the adequate circuits, and it is plausible that there is no effective way of constructing these circuits (e.g., as in case of circuits that trivially solve undecidable problem regarding unary instances). A minimal notion of constructibility of such (polynomial-size) circuits is the existence of a polynomial time algorithm that given 1^n produces the n^{th} relevant circuit (i.e., the circuit that solves the problem on instances of length n). Such a notion of constructibility means that the family of circuits is "uniform" in some sense (rather than consisting of circuits that have no relation between one another). Stronger notions of uniformity (e.g., log-space constructibility) are more adequate for subclasses such as AC and NC. We mention that log-space uniform NC circuits correspond to parallel algorithms that use polynomially many processors and run in polylogarithmic time.

Appendix B

On the Quest for Lower Bounds

Alas, Philosophy, Medicine, Law, and unfortunately also Theology, have I studied in detail, and still remained a fool, not a bit wiser than before. Magister and even Doctor am I called, and for a decade am I sick and tired of pulling my pupils by the nose and understanding that we can know nothing.[1]

J.W. Goethe, Faust, Lines 354–364

Summary: This appendix briefly surveys some attempts at proving lower bounds on the complexity of natural computational problems. In the first part, devoted to Circuit Complexity, we describe lower bounds on the *size* of (restricted) circuits that solve natural computational problems. This can be viewed as a program whose long-term goal is proving that $\mathcal{P} \neq \mathcal{NP}$. In the second part, devoted to Proof Complexity, we describe lower bounds on the length of (restricted) propositional proofs of natural tautologies. This can be viewed as a program whose long-term goal is proving that $\mathcal{NP} \neq \text{co}\mathcal{NP}$.

We comment that while the activity in these areas is aimed towards developing proof techniques that may be applied to the resolution of the "big problems" (such as P versus NP), the current achievements (though very impressive) seem very far from reaching this goal. Current crown-jewel achievements in these areas take the form of tight (or strong) lower bounds on the complexity of computing (resp., proving) "relatively simple" functions (resp., claims) in *restricted* models of computation (resp., proof systems).

[1]This quote reflects a common sentiment, not shared by the author of the current book.

B.1 Preliminaries

Circuit complexity refers to a non-uniform model of computation (see Section 1.2.4), focusing on the size of such circuits, while ignoring the complexity of constructing adequate circuits. Similarly, proof complexity refers to proofs of tautologies, focusing on the length of such proofs, while ignoring the complexity of generating such proofs.

Both circuits and proofs are finite objects that are defined on top of the notion of a *directed acyclic graph* (dag), reviewed in Appendix G.1. In such a dag, vertices with no incoming edges are called inputs, vertices with no outgoing edges are called outputs, and the remaining vertices are called internal vertices. The size of a dag is defined as the number of its edges. We will be mostly interested in dags of "bounded fan-in" (i.e., for each vertex, the number of *incoming* edges is at most two).

In order to convert a dag into a computational device (resp., a proof), each internal vertex is labeled by a rule, which transforms values assigned to its predecessors to values at that vertex. Combined with any possible assignment of values to the inputs, these fixed rules induce an assignment of values to all the vertices of the dag (by a process that starts at the inputs, and assigns a value to each vertex based on the values of its predecessors (and according to the corresponding rule)).

- In the case of computation devices, the internal vertices are labeled by (binary or unary) functions over some fixed domain (e.g., a finite or infinite field). These functions are called gates, and the labeled dag is called a circuit. Such a circuit (with n inputs and m outputs) computes a finite function over the corresponding domain (mapping sequences of length n to sequences of length m).

- In the case of proofs, the internal vertices are labeled by sound deduction (or inference) rules of some fixed proof system. Any assignment of axioms (of the said system) to the inputs of this labeled dag yields a sequence of tautologies (at all vertices). Typically the dag is assumed to have a single output vertex, and the corresponding sequence of tautologies is viewed as a proof of the tautology assigned to the output.

We note that both models partially adhere to the paradigm of simplicity that underlies the definitions of (uniform) computational models (as discussed in Section 1.2.3): the aforementioned rules are simple by definition – they are applied to at most two values. However, unlike in the case of (uniform) computational models, the current models do not mandate a "uniform" consideration of all possible "inputs" (but rather allow a seperate consideration of each finite "input" length). For example, each circuit can compute only a finite function; that is, a function defined over a fixed number of values (i.e., fixed input length). Likewise, a dag that corresponds to a proof system, yields only proofs of tautologies that refer to a fixed number of axioms.[2]

[2]N.B., we refer to a fixed number of axioms, and not merely to a fixed number of axiom forms.

Focusing on circuits, we note that in order to allow the computation of functions that are defined for all input lengths, one must consider infinite sequences of dags, one for each length. This yields a model of computation in which each "machine" has an infinite description (when referring to all input lengths). Indeed, this significantly extends the power of the computation model beyond that of the notion of *algorithm* (discussed in Section 1.2.3). However, since we are interested in lower bounds here, this extension is certainly legitimate and hopefully fruitful: For example, one may hope that the finiteness of the individual circuits will facilitate the application of combinatorial techniques towards the analysis of the model's power and limitations. Furthermore, as we shall see, these models open the door to the introduction (and study) of meaningful restricted classes of computations.

Organization: The rest of this appendix is partitioned to three parts. In Section B.2 we consider Boolean circuits, which are the archetypical model of non-uniform computing devices. In Section B.3 we generalize the treatment by considering arithmetic circuits, which may be defined for every algebraic structure (where Boolean circuits are viewed as a special case referring to the two-element field, GF(2)). Lastly, in Section B.4, we consider proof complexity.

B.2 Boolean Circuit Complexity

In Boolean circuits the values assigned to all inputs as well as the values induced (by the computation) at all intermediate vertices and outputs are bits. The set of allowed gates is taken to be any *complete basis* (i.e., one that allows to compute *all* Boolean functions). The most popular choice of a complete basis is the set $\{\wedge, \vee, \neg\}$ corresponding to (two-bit) conjunction, (two-bit) disjunction and negation (of a single bit), respectively. (The specific choice of a complete basis hardly effects the study of circuit complexity.)

For a finite Boolean function f, we denote by $\mathcal{S}(f)$ the size of the smallest Boolean circuit computing f. We will be interested in sequences of functions $\{f_n\}$, where f_n is a function on n input bits, and will study their size complexity (i.e., $\mathcal{S}(f_n)$) asymptotically (as a function of n). With some abuse of notation, for $f(x) \overset{\text{def}}{=} f_{|x|}(x)$, we let $\mathcal{S}(f)$ denote the integer function that assigns to n the value $\mathcal{S}(f_n)$. Thus, we refer to the following definition.

Definition B.1 (circuit complexity): *Let $f : \{0,1\}^* \to \{0,1\}^*$ and $\{f_n\}$ be such that $f(x) = f_{|x|}(x)$ for every x. The* complexity *of f* (resp., $\{f_n\}$)*, denoted $\mathcal{S}(f)$* (resp., denoted $n \mapsto \mathcal{S}(f_n)$)*, is a function of n that represents the size of the smallest Boolean circuit computing f_n.*

We stress that different circuits (e.g., having a different number of inputs) are used for different f_n's. Still there may be a simple description of this sequence of circuits, say, an algorithm that on input n produces a circuit computing f_n. In case such

Recall that an axiom form like $\phi \vee \neg\phi$ yields an infinite number of axioms, each obtained by substituting the generic formula (or symbol) ϕ with a fixed propositional formula.

an algorithm exists and works in time polynomial in the size of its output, we say that the corresponding sequence of circuits is uniform. Note that if f has a uniform sequence of polynomial-size circuits then $f \in \mathcal{P}$. On the other hand, any $f \in \mathcal{P}$ has (a uniform sequence of) polynomial-size circuits. Consequently, a super-polynomial size lower-bound on any function in $\mathcal{NP}$ would imply that $\mathcal{P} \neq \mathcal{NP}$.

Definition B.1 makes no reference to the uniformity condition (and indeed the sequence of smallest circuits computing $\{f_n\}$ may be "highly nonuniform"). Actually, non-uniformity makes the circuit model stronger than Turing machines (or, equivalently, stronger than the model of uniform circuits): *there exist functions f that cannot be computed by Turing machines* (regardless of their running time), *but do have linear-size circuits.*[3] This raises the possibility that proving circuit lower-bounds is even harder than resolving the P vs. NP Question.

The common belief is that the extra power provided by non-uniformity is irrelevant to the P vs. NP Question; in particular, it is conjectured that NP-complete sets do not have polynomial-size circuits. This conjecture is supported by the fact that its failure will yield an unexpected collapse in the world of uniform computational complexity (see Section 3.2). Furthermore, the hope is that abstracting away the (supposedly irrelevant) uniformity condition will allow for combinatorial techniques to analyze the power and limitations of polynomial-size circuits (w.r.t NP-sets). This hope has materialized in the study of restricted classes of circuits (see Sections B.2.2 and B.2.3). Indeed, another advantage of the circuit model is that it offers a framework for describing naturally restricted models of computation.

We also mention that Boolean circuits are a natural computational model, corresponding to "hardware complexity" (which was indeed the original motivation for their introduction by Shannon [202]), and so their study is of independent interest. Moreover, some of the techniques for analyzing Boolean functions found applications elsewhere (e.g., in computational learning theory, combinatorics and game theory).

B.2.1 Basic Results and Questions

We have already mentioned several basic facts about Boolean circuits. Another basic fact is that *most Boolean functions require exponential size circuits*, which is due to the gap between the number of functions and the number of small circuits.

Thus, hard functions (i.e., functions that require large circuits and thus have no efficient algorithms) do exist, to say the least. However, the aforementioned hardness result is proved via a counting argument, which provides no way of pointing to any specific hard function. The situation is even worse: *super-linear* circuit-size lower-bounds are not known for any *explicit* function f, even when explicitness is defined in a very mild sense that only requires $f \in \mathcal{EXP}$.[4] One major open problem of circuit complexity is establishing such lower-bounds.

[3] See either Theorem 1.13 or Theorem 3.7.

[4] Indeed, a more natural (and still mild) notion of explicitness requires that $f \in \mathcal{E}$. This notion implies that the function's description (restricted to n-bit long inputs) can be constructed in time that is polynomial in the length of the description.

Open Problem B.2 *Find an explicit function $f : \{0,1\}^* \to \{0,1\}$ (or even $f : \{0,1\}^* \to \{0,1\}^*$ such that $|f(x)| = O(|x|)$) for which $\mathcal{S}(f)$ is not $O(n)$.*

A particularly basic special case of this open problem is the question of *whether addition is easier to perform than multiplication.* Let $\mathtt{ADD}_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^{n+1}$ and $\mathtt{MULT}_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^{2n}$, denote the addition and multiplication functions, respectively, applied to a pair of integers (presented in binary). For addition we have an optimal upper bound; that is, $\mathcal{S}(\mathtt{ADD}_n) = O(n)$. For multiplication, the standard (elementary school) quadratic-time algorithm can be greatly improved (via Discrete Fourier Transforms) to almost-linear time, yielding $\mathcal{S}(\mathtt{MULT}_n) = \widetilde{O}(n)$. Now, the question is *whether or not there exist linear-size circuits for multiplication* (i.e., is $\mathcal{S}(\mathtt{MULT}_n) = O(n)$)?

Unable to report on any super-linear lower-bound (for an explicit function), we turn to restricted types of Boolean circuits. There has been some remarkable successes in developing techniques for proving strong lower-bounds for natural restricted classes of circuits. We describe the most important ones, and refer the reader to [46, 235] for further detail.

Recall that general Boolean circuits can compute every function. In contrast, restricted types of circuits (e.g., monotone circuits) may only be able to compute a subclass of all functions (e.g., monotone functions), and in such a case we shall seek lower-bounds on the size of such restricted circuits that compute a function in the corresponding subclass. Such a restriction is appealing provided that the corresponding class of functions and the computations represented by the restricted circuits are natural (from a conceptual or practical viewpoint). The models discussed below satisfy this condition.

B.2.2 Monotone Circuits

One very natural restriction on circuits arises by forbidding negation (in the set of gates), namely allowing only $\wedge$ and $\vee$ gates. The resulting circuits are called monotone and they can compute a function $f : \{0,1\}^n \to \{0,1\}$ if and only if f is monotone with respect to the standard partial order on n-bit strings (i.e., $x \preceq y$ iff for every bit position i we have $x_i \leq y_i$). An extremely natural question in this context is *whether or not non-monotone operations* (in the circuit) *help in computing monotone functions*?

Before turning to this question, we note that most monotone functions require exponential size circuits (let alone monotone ones).[5] Still, proving a super-polynomial lower-bound on the monotone circuit complexity of an explicit monotone function was open for several decades, till the invention of the so-called *approximation method* (by Razborov [187]).

Let $\mathtt{CLIQUE}_n$ be the function that, given a graph on n vertices (by its adjacency matrix), outputs 1 if and only if the graph contains a complete subgraph of size

[5]A key observation is that it suffices to consider the set of n-bit monotone functions that evaluate to 1 (resp., to 0) on each string $x = x_1 \cdots x_n$ satisfying $\sum_{i=1}^n x_i > \lfloor n/2 \rfloor$ (resp., $\sum_{i=1}^n x_i < \lfloor n/2 \rfloor$). Note that each such function is specified by $\binom{n}{\lfloor n/2 \rfloor}$ bits.

(say) $\sqrt{n}$. This function is clearly monotone, and $\texttt{CLIQUE} = \{\texttt{CLIQUE}_n\}$ is known to be NP-complete.

Theorem B.3 ([187], improved in [7]): *There are no polynomial-size monotone circuits for* CLIQUE.

We note that the lower-bounds are sub-exponential in the number of vertices (i.e., $\mathcal{S}(\texttt{CLIQUE}_n) = \exp(\Omega(n^{1/8}))$), and that similar lower-bounds are known for functions in $\mathcal{P}$. Thus, *there exists an exponential separation between monotone circuit complexity and non-monotone circuit complexity*, where this separation refers (of course) to the computation of monotone functions.

B.2.3 Bounded-Depth Circuits

The next restriction refers to the structure of the circuit (or rather to its underling graph): *we allow all gates, but limit the depth of the circuit.* The depth of a dag is simply the length of the longest directed path in it. So in a sense, depth captures the *parallel time* to compute the function: if a circuit has depth d, then the function can be evaluated by enough processors in d phases (where in each phase many gates are evaluated in parallel). Indeed, parallel time is a natural and important computational resource, referring to the following basic question: *can one speed up computation by using several computers in parallel?* Determining which computational tasks can be "parallelized" when many processors are available and which are "inherently sequential" is clearly a fundamental question.

We will restrict d to be a constant, which still is interesting not only as a measure of parallel time but also due to the relation of this model to expressibility in first order logic as well as to the *Polynomial-time Hierarchy* (defined in Section 3.2). In the current setting (of constant-depth circuits), we allow *unbounded fan-in* (i.e., $\wedge$-gates and $\vee$-gates taking any number of incoming edges), as otherwise each output bit can depend only on a constant number of input bits.

Let PAR (for parity) denote the sum modulo two of the input bits, and MAJ (for majority) be 1 if and only if there are more 1's than 0's among the input bits. The invention of the *random restriction method* (by Furst, Saxe, and Sipser [83]) led to the following basic result.

Theorem B.4 ([83], improved in [239, 115]): *For all constant d, the functions* PAR *and* MAJ *have no polynomial size circuit of depth d.*

The aforementioned improvement (of Håstad [115], following Yao [115]) gives a relatively tight lower-bound of $\exp(\Omega(n^{1/(d-1)}))$ on the size of n-input PAR circuits of depth d.

Interestingly, MAJ remains hard (for constant-depth polynomial-size circuits) even if the circuits are also allowed (unbounded fan-in) PAR-gates (this result is based on yet another proof technique: *approximation by polynomials* [209, 188]). However, the "converse" does not hold (i.e., constant-depth polynomial-size circuits with MAJ-gates can compute PAR), and in general the class of constant-depth polynomial-size circuits with MAJ-gates (denoted $\mathcal{TC}^0$) seems quite powerful. In particular, nobody has managed to *prove that there are functions in $\mathcal{NP}$ that cannot be computed by such circuits*, even if their depth is restricted to 3.

B.2.4 Formula Size

The final restriction is again structural – we require the underlying dag to be a tree (i.e., a dag in which each vertex has at most one *outgoing* edge). Intuitively, this forbids the computation from reusing a previously computed intermediate value (and if this value is needed again then it has to be recomputed). Thus, the resulting Boolean circuits are simply Boolean formulae. (Indeed, we are back to the basic model allowing negation ($\neg$), and $\wedge, \vee$ gates *of fan-in 2.*)

Formulae are natural not only for their prevalent mathematical use, but also because their size can be related to the depth of general circuits and to the *memory* requirements of Turing machines (i.e., their space complexity). One of the oldest results on Circuit Complexity, is that PAR and MAJ have nontrivial lower-bounds in this model. The proof follows a simple combinatorial (or information theoretic) argument.

Theorem B.5 [144]: *Boolean formulae for n-bit* PAR *and* MAJ *require $\Omega(n^2)$ size.*

This should be contrasted with the linear-size circuits that exist for both functions.[6] Encouraged by Theorem B.5, one may hope to see super-polynomial lower-bounds on the formula-size of explicit functions. This is indeed a famous open problem.

Open Problem B.6 *Find an explicit Boolean function f that requires super-polynomial size formulae.*

An equivalent formulation of this open problem calls for proving a super-logarithmic lower-bound on the depth of formulae (or circuits) computing f.

One appealing method for addressing such challenges is the *communication complexity method* (of Karchmer and Wigderson [141]). This method asserts that the depth of a formula for a Boolean function f equals the communication complexity in the following two party game, G_f. In the game, the first party is given $x \in f^{-1}(1) \cap \{0,1\}^n$, the second party is given $y \in f^{-1}(0) \cap \{0,1\}^n$, and their goal is to find a bit location on which x and y disagree (i.e., i such that $x_i \neq y_i$, which clearly exists). To that end, the party exchange messages, according to a predetermined protocol, and the question is what is the communication complexity (in terms of total number of bits exchanged on the worst-case input pair) of the best such protocol. We stress that no computational restrictions are placed on the parties in the game/protocol.

Note that proving a super-logarithmic lower-bound on the communication complexity of the game G_f will establish a super-logarithmic lower-bound on the depth of formulae (or circuits) computing f (and thus a super-polynomial lower-bound on the size of formulae computing f). We stress the fact that a lower-bound of a purely information theoretic nature implies a computational lower-bound!

[6]We comment that $\mathcal{S}(\texttt{PAR}) = O(n)$ is trivial, but $\mathcal{S}(\texttt{MAJ}) = O(n)$ is not.

We mention that the communication complexity method has a *monotone version* such that the depth of *monotone* circuits is related to the communication complexity of protocols that are required to find an i such that $x_i > y_i$ (rather than any i such that $x_i \neq y_i$).[7] In fact, the monotone version is better known than the general one, due to its success in leading to linear lower-bounds on the monotone depth of natural problems such as perfect matching (established by Raz and Wigderson [186]).

B.3 Arithmetic Circuits

We now leave the Boolean rind, and discuss circuits over general fields. Fixing any field F, the gates of the dag will now be the standard $+$ and $\times$ operations of the field, yielding a so-called arithmetic circuit. The inputs of the dag will be assigned elements of the field F, and these values induce an assignment of values (in F) to all other vertices. Thus, an arithmetic circuit with n inputs and m outputs computes a polynomial map $p : F^n \to F^m$, and every such polynomial map is computed by some circuit (modulo the convention of allowing some inputs to be set to some constants, most importantly the constant -1).[8]

Arithmetic circuits provide a natural description of methods for computing polynomial maps, and consequently their size is a natural measure of the complexity of such maps. We denote by $\mathcal{S}_F(p)$ the size of a smallest circuit computing the polynomial map p (and when no subscript is specified, we mean that $F = \mathcal{Q}$ (the field of rational numbers)). As usual, we shall be interested in sequences of functions, one per each input size, and will study the corresponding circuit-size asymptotically.

We note that, for any *fixed* finite field, arithmetic circuits can simulate Boolean circuits (on Boolean inputs) with only constant factor loss in size. Thus, the study of arithmetic circuits focuses more on infinite fields, where lower bounds may be easier to obtain.

As in the Boolean case, the existence of hard functions is easy to establish (via dimension considerations, rather than counting argument), and we will be interested in *explicit* (families of) polynomials. Roughly speaking, a polynomial is called explicit if there exists an efficient algorithm that, when given a degree sequence (which specifies a monomial), outputs the (finite description of the) corresponding coefficient.

An important parameter, which is absent in the Boolean model, is the *degree* of the polynomial(s) computed. It is obvious, for example, that a degree d polynomial (even in one variable, i.e., $n = 1$) requires size at least $\log d$. We briefly consider the univariate case (where d is the only measure of "problem size"), which already contains striking and important open problems. Then we move to the general

[7]Note that since f is monotone, $f(x) = 1$ and $f(y) = 0$ implies the existence of an i such that $x_i = 1$ and $y_i = 0$.

[8]This allows the emulation of adding a constant, multiplication by a constant, and subtraction. We mention that, for the purpose of computing polynomials (over infinite fields), division can be efficiently emulated by the other operations.

multivariate case, in which (as usual) the number of variables (i.e., n) will be the main parameter (and we shall assume that $d \leq n$). We refer the reader to [86, 215] for further detail.

B.3.1 Univariate Polynomials

How tight is the $\log d$ lower-bounds for the size of an arithmetic circuit computing a degree d polynomial? A simple dimension argument shows that for most degree d polynomials p, it holds that $\mathcal{S}(p) = \Omega(d)$. However, we know of no explicit one:

Open Problem B.7 *Find an explicit polynomial p of degree d, such that $\mathcal{S}(p)$ is not $O(\log d)$.*

To illustrate this open problem, we consider the following two concrete polynomials $p_d(x) = x^d$ and $q_d(x) = (x+1)(x+2)\cdots(x+d)$. Clearly, $\mathcal{S}(p_d) \leq 2\log d$ (via repeated squaring), so the trivial lower-bound is essentially tight. On the other hand, it is a major open problem to determine $\mathcal{S}(q_d)$, and the common conjecture is that $\mathcal{S}(q_d)$ is not polynomial in $\log d$. To realize the importance of this conjecture, we state the following proposition:

Proposition B.8 *If $\mathcal{S}(q_d) = \mathrm{poly}(\log d)$, then the integer factorization problem can be solved by polynomial-size circuits.*

Recall that it is widely believed that the integer factorization problem is intractable (and, in particular, does not have polynomial-size circuits).

Proof Sketch: Proposition B.8 follows by observing that $q_d(t) = ((t+d)!)/(t!)$ and that a small circuit for computing q_d yields an efficient way of obtaining the value $((t+d)!)/(t!) \bmod N$ (by emulating the computation of the former circuit modulo N). Observing that $(\sum_{i=1}^{\ell} K_i)! = \prod_{i=1}^{\ell} q_{K_i}(\sum_{j=i+1}^{\ell} K_j)$, it follows that the value of $(K!) \bmod N$ can be obtained by using circuits for the polynomials $\langle q_{2^i} : i = 1,.., \lfloor \log_2 K \rfloor \rangle$. Next, observe that $(K!) \bmod N$ and N are relatively prime if and only if all prime factors of N are bigger than K. Thus, given a composite N (and circuits for $\langle q_{2^i} : i = 1,.., \lfloor \log_2 N \rfloor \rangle$), we can find a factor of N by performing a binary search for a suitable K. □

B.3.2 Multivariate Polynomials

We are now back to polynomials with n variables. To make n our only "problem size" parameter, it is convenient to restrict ourselves to polynomials whose total degree is at most n.

Once again, almost every polynomial p in n variables requires size $\mathcal{S}(p) \geq \exp(\Omega(n))$, and we seek explicit polynomial (families) that are hard. Unlike in the Boolean world, here there are slightly nontrivial lower-bounds (via elementary tools from algebraic geometry).

Theorem B.9 [26]: $\mathcal{S}(x_1^n + x_2^n + \cdots + x_n^n) = \Omega(n \log n)$.

The same techniques extend to prove a similar lower-bound for other natural polynomials such as the symmetric polynomials and the determinant. Establishing a stronger lower-bound for any explicit polynomial is a major open problem. Another open problem is obtaining a super-linear lower-bound for a polynomial map of constant (even 1) total degree. Outstanding candidates for the latter open problem are the *linear* maps computing the Discrete Fourier Transform over the Complex numbers, or the Walsh transform over the Rationals (for both $O(n \log n)$-time algorithms are known, but no super-linear lower-bounds are known).

We now focus on specific polynomials of central importance. The most natural and well studied candidate for the last open problem is the matrix multiplication function $\mathtt{MM}$: let A, B be two $m \times m$ matrices over F, and define $\mathtt{MM}_n(A, B)$ to be the sequence of $n = m^2$ values of the entries of the matrix $A \times B$. Thus, $\mathtt{MM}_n$ is a sequence of n explicit bilinear forms over the $2n$ input variables (which represent the entries of both matrices). It is known that $\mathcal{S}_{\mathrm{GF}(2)}(\mathtt{MM}_n) \geq 3n$ (cf., [206]). On the other hand, the obvious algorithm that takes $O(m^3) = O(n^{3/2})$ steps can be improved.

Theorem B.10 [62]: *For every field F, it holds that $\mathcal{S}_F(\mathtt{MM}_n) = o(n^{1.19})$.*

So what is the complexity of $\mathtt{MM}$ (even if one counts only multiplication gates)? Is it linear or almost-linear or is it the case that $\mathcal{S}(\mathtt{MM}) > n^\alpha$ for some $\alpha > 1$? This is indeed a famous open problem.

We next consider the determinant and permanent polynomials ($\mathtt{DET}$ and $\mathtt{PER}$, resp.) over the $n = m^2$ variables representing an $m \times m$ matrix. While $\mathtt{DET}$ plays a major role in classical mathematics, $\mathtt{PER}$ is somewhat esoteric in that context (though it appears in Statistical Mechanics and Quantum Mechanics). In the context of complexity theory both polynomials are of great importance, because they capture natural complexity classes. The function $\mathtt{DET}$ has relatively low complexity (and is related to the class of polynomials having polynomial-sized arithmetic formulae), whereas $\mathtt{PER}$ seems to have high complexity (and is complete for the counting class $\#\mathcal{P}$ (see §6.2.1)). Thus, it is conjectured that $\mathtt{PER}$ is *not* polynomial-time reducible to $\mathtt{DET}$. One restricted type of reduction that makes sense in this algebraic context is a reduction by projection.

Definition B.11 (projections): *Let $p_n : F^n \to F^\ell$ and $q_N : F^N \to F^\ell$ be polynomial maps and $x_1, ..., x_n$ be variables over F. We say that there is a* projection from p_n to q_N over F, *if there exists a function $\pi : [N] \to \{x_1, ..., x_n\} \cup F$ such that $p_n(x_1, ..., x_n) \equiv q_N(\pi(1), ..., \pi(N))$.*

Clearly, if there is a projection from p_n to q_N then $\mathcal{S}_F(p_n) \leq \mathcal{S}_F(q_N)$. Let $\mathtt{DET}_m$ and $\mathtt{PER}_m$ denote the functions $\mathtt{DET}$ and $\mathtt{PER}$ restricted to m-by-m matrices. It is known that there is a projection from $\mathtt{PER}_m$ to $\mathtt{DET}_{3^m}$, but to yield a polynomial-time reduction one would need a projection of $\mathtt{PER}_m$ to $\mathtt{DET}_{\mathrm{poly}(m)}$. Needless to say, it is conjectured that no such projection exists.

B.4 Proof Complexity

It is common practice to classify proofs according to the level of their difficulty, but can this appealing classification be put on sound grounds? This is essentially the task undertaken by Proof Complexity. It seeks to classify theorems according to the difficulty of proving them, much like Circuit Complexity seeks to classify functions according to the difficulty of computing them. Furthermore, just like in circuit complexity, we shall also refer to a few (restricted) models, called *proof systems*, which represent various methods of reasoning. Thus, the difficulty of proving various theorems will be measured with respect to various proof systems.

We will consider only propositional proof systems, and so the theorems (in these systems) will be propositional *tautologies*. Each of these systems will be *complete* and *sound*; that is, each tautology and only a tautology will have a proof relative to these systems. The formal definition of a proof system spells out what we take for granted: the efficiency of the verification procedure. In the following definition the efficiency of the verification procedure refers to its running-time measured in terms of the *total length of the alleged theorem and proof.*[9]

Definition B.12 [61]: *A* (propositional) proof system *is a polynomial-time Turing machine M such that a formula T is a tautology if and only if there exists a string π, called a* proof, *such that $M(\pi, T) = 1$.*

In agreement with standard formalisms, the proof is viewed as coming before the theorem. Definition B.12 guarantees the completeness and soundness of the proof system as well as verification efficiency (relative to the total length of the alleged proof–theorem pair). Note that Definition B.12 allows proofs of arbitrary length, suggesting that the length of the proof π is a measure of the *complexity* of the tautology T *with respect to the proof system M*.

For each tautology T, let $\mathcal{L}_M(T)$ denote the length of the shortest proof of T in M (i.e., the length of the shortest string π such that M accepts (π, T)). That is, $\mathcal{L}_M$ captures the *proof complexity* of various tautologies with respect to the proof system M. Abusing notation, we let $\mathcal{L}_M(n)$ denotes the maximum $\mathcal{L}_M(T)$ over all tautologies T of length n. (By definition, for every proof system M, the value $\mathcal{L}_M(n)$ is well-defined and so $\mathcal{L}_M$ is a total function over the natural numbers.) The following simple theorem provides a basic connection between proof complexity (with respect to any propositional proof system) and computational complexity (i.e., the NP-vs-coNP Question).

Theorem B.13 [61]: *There exists a propositional proof system M such that the function $\mathcal{L}_M$ is upper-bounded by a polynomial if and only if $\mathcal{NP} = \text{co}\mathcal{NP}$.*

In particular, a propositional proof system M such that $\mathcal{L}_M$ is upper-bounded by a polynomial coincides with a NP-proof system (as in Definition 2.5) for the set of propositional tautologies, which is a co$\mathcal{NP}$-complete set.

[9]Indeed, this convention differs from the convention emplyed in Chapter 9, where the complexity of verification (i.e., verifier's running-time) was measured as a function of the *length of the alleged theorem*. Both approaches were mentioned in Section 2.1, where the two approaches coincide because in Section 2.1 we mandated proofs of length polynomial in the alleged theorem.

The long-term goal of Proof Complexity is establishing super-polynomial lower-bounds on the length of proofs in any propositional proof system (and thus establishing $\mathcal{NP} \neq \text{co}\mathcal{NP}$). It is natural to start this formidable project by first considering simple (and thus weaker) proof systems, and then moving on to more and more complex ones. Moreover, various natural proof systems, capturing basic (restricted) types and "primitives" of reasoning as well as natural tautologies, suggest themselves as objects for this study. In the rest of this section we focus on such restricted proof systems.

Different branches of Mathematics such as logic, algebra and geometry give rise to different proof systems, often implicitly. A typical system would have a set of axioms and a set of deduction rules. A proof (in this system) would proceed to derive the desired tautology in a sequence of steps, each producing a formula (often called a line of the proof), which is either an axiom, or follows from previous formulae via one of the deduction rules. Regarding these proof systems, we make two observations. First, proofs in these systems can be easily verified by an algorithm and thus they fit the general framework of Definition B.12. Second, these proof systems perfectly fit the model of a dag with internal vertices lbeled by deduction rules (as in Section B.1): When assigning axioms to the inputs, the application of the deduction rules at the internal vertices yields a proof of the tautology assigned to each output.[10]

For various proof systems Π, we turn to study the proof length $\mathcal{L}_\Pi(T)$ of tautologies T in proof system Π. The first observation, revealing a major difference between proof complexity and circuit complexity, is that the trivial counting argument *fails*. The reason is that, while the number of functions on n bits is 2^{2^n}, there are at most 2^n tautologies of this length. Thus, in proof complexity, even the *existence* of a hard tautology, not necessarily an explicit one, would be of interest (and, in particular, if established for all propositional proof systems then it would yield $\mathcal{NP} \neq \text{co}\mathcal{NP}$). (Note that here we refer to hard instances of of a problem and not to hard problems.) Anyhow, as we shall see, most known proof-length lower-bounds (with respect to restricted proof systems) apply to very natural (let alone explicit) tautologies.

An important convention: There is an equivalent and somewhat more convenient view of (simple) proof systems, namely as (simple) refutation systems. First, recalling that 3SAT is NP-complete, note that the negation of any (propositional) tautology can be written as a conjunction of clauses, where each clause is a disjunction of only 3 literals (variables or their negation). Now, if we take these clauses as axioms and derive (using the rules of the system) a obvious contradiction (e.g., the negation of an axiom, or better yet the empty clause), then we have proved the tautology (since we have proved that its negation yields a contradiction). Proof complexity often takes the refutation viewpoint, and often exchanges "tautology" with its negation ("contradiction").

[10]General proof systems as in Definition B.12 can also be adapted to this formalism, by considering a deduction rule that corresponds to a single step of the machine M. However, the deduction rules considered below are even simpler, and more importantly they are more natural.

Organization: The rest of this section is divided to three parts, referring to logical, algebraic and geometric proof systems. We will briefly describe important representative and basic results in each of these domains, and refer the reader to [27] for further detail (and, in particular, to adequate references).

B.4.1 Logical Proof Systems

The proof systems in this section will all have lines that are Boolean formulae, and the differences will be in the structural limits imposed on these formulae. The most basic proof system, called Frege system, puts no restriction on the formulae manipulated by the proof. It has one derivation rule, called the cut rule: $A \vee C, B \vee \neg C \vdash A \vee B$ (for any propositional formulae A, B and C). Adding any other sound rule, like *modus ponens*, has little effect on the length of proofs in this system.

Frege systems are basic in the sense that (in several variants) they are the most common systems in Logic. Indeed, polynomial length proofs in Frege systems naturally corresponds to "polynomial-time reasoning" about feasible objects. The major open problem in proof complexity is finding any tautology (i.e., a family of tautologies) that has no polynomial-long proof in the Frege system.

Since lower-bounds for Frege systems seem intractable at the moment, we turn to subsystems of Frege which are interesting and natural. The most widely studied system (of refutation) is Resolution, whose importance stems from its use by most propositional (as well as first order) automated theorem provers. The formulae allowed as lines in Resolution are clauses (disjunctions), and so the *cut rule* simplifies to the resolution rule: $A \vee x, B \vee \neg x \vdash A \vee B$, for any clauses A, B and variable x.

The gap between the power of general Frege systems and Resolution is reflected by the existence of tautologies that are easy for Frege and hard for Resolution. A specific example is provided by the pigeonhole principle, denoted PHP_n^m, which is a propositional tautology that expresses the fact that there is no one-to-one mapping of m pigeons to $n < m$ holes.

Theorem B.14 $\mathcal{L}_{\text{Frege}}(\mathsf{PHP}_n^{n+1}) = n^{O(1)}$ *but* $\mathcal{L}_{\text{Resolution}}(\mathsf{PHP}_n^{n+1}) = 2^{\Omega(n)}$

B.4.2 Algebraic Proof Systems

Just as a natural contradiction in the Boolean setting is an unsatisfiable collection of clauses, a natural contradiction in the algebraic setting is a system of polynomials without a common root. Moreover, CNF formulae can be easily converted to a system of polynomials, one per clause, over any field. One often adds the polynomials $x_i^2 - x_i$ which ensure Boolean values.

A natural proof system (related to Hilbert's Nullstellensatz, and to computations of Grobner bases in symbolic algebra programs) is Polynomial Calculus, abbreviated PC. The lines in this system are polynomials (represented explicitly by all coefficients), and it has two deduction rules: For any two polynomials g, h, the rule $g, h \vdash g + h$, and for any polynomial g and variable x_i, the rule $g, x_i \vdash x_i g$. Strong length lower-bounds (obtained from degree lower-bounds) are known for this system. For example, encoding the pigeonhole principle $\mathtt{PHP}_n^m$ as a contradicting set of constant degree polynomials, we have the following lower-bound.

Theorem B.15 *For every n and every $m > n$, it holds that $\mathcal{L}_{\mathrm{PC}}(\mathtt{PHP}_n^m) \geq 2^{n/2}$, over every field.*

B.4.3 Geometric Proof Systems

Yet another natural way to represent contradictions is by a set of regions in space that have empty intersection. Again, we care mainly about discrete (say, Boolean) domains, and a wide source of interesting contradictions are integer programs arising from Combinatorial Optimization. Here, the constraints are (affine) linear inequalities with integer coefficients (so the regions are subsets of the Boolean cube carved out by half-spaces). The most basic system is called Cutting Planes (CP), and its lines are linear inequalities with integer coefficients. The deduction rules of PC are (the obvious) addition of inequalities, and the (less obvious) division of the coefficients by a constant (and rounding, taking advantage of the integrality of the solution space).

While $\mathtt{PHP}_n^m$ is "easy" in this system, *exponential lower-bounds are known for other tautologies.* We mention that they are obtained from the *monotone circuit* lower bounds of Section B.2.2.

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, given in Sections 7.1, 8.2 and 9.2, respectively.[1] Using these basic primitives, the appendix provides a treatment of basic cryptographic applications such as Encryption, Signatures, and General Cryptographic Protocols.

[1]These augmentations are important for cryptography, but are not central to complexity theory and thus were omitted from the main text.

C.1 Introduction and Preliminaries

The rigorous treatment and vast expansion of cryptography is one of the major achievements of theoretical computer science. In particular, classical notions such as secure encryption and unforgeable signatures were placed on sound grounds, and new (unexpected) directions and connections were uncovered. Furthermore, this development was coupled with the introduction of novel concepts such as computational indistinguishability, pseudorandomness, and zero-knowledge interactive proofs, which are of independent interest (see Sections 7.1, 8.2 and 9.2, respectively). Indeed, modern cryptography is strongly coupled with complexity theory (in contrast to "classical" cryptography which is strongly related to information theory).

C.1.1 The Underlying Principles

Modern cryptography is concerned with the construction of information systems that are robust against malicious attempts aimed at causing these systems to violate their prescribed functionality. The prescribed functionality may be the secret and authenticated communication of information over an insecure channel, 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 channel).

C.1.1.1 Coping with adversaries

Needless to say, 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, 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.

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 in such a case its security is proved relying on the features of these tools.

Example: Starting with the wish to ensure secret (resp., reliable) communication over insecure channels, the definitional stage leads to the formulation of the notion of secure encryption schemes (resp., signature schemes). Next, such schemes are constructed by using simpler primitives such as one-way functions, and the security of the construction is proved via a "reducibility argument" (which demonstrates how inverting the one-way function "reduces" to violating the claimed security of the construction; cf., Section 7.1.2).

C.1.1.2 The use of computational assumptions

Like in the case of the foregoing example, most of the tools and applications of cryptography exist only if some sort of computational hardness exists. Specifically, these tools and applications 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, as discussed in Sections 7.1.1 and C.2, proving that one-way functions exist seems even harder than proving that $\mathcal{P} \neq \mathcal{NP}$. 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 whatever we want" (i.e., the existence of a solution to some natural cryptographic problem)? Well, firstly, we need to know what we want; that is, we must first clarify what *exactly* we want, which means going through the typically complex definitional stage. But once this stage is completed and a definition is obtained, can we just assume the existence of a system satisfying this definition? Not really: the mere existence of a definition does not imply that it can be satisfied by any system.

The way to demonstrate that a cryptographic definition is viable (and that the corresponding intuitive security concern can be satisfied) is to prove that it can be satisfied 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 were even believed to be false), have been shown to hold by "reduction" to widely believed assumptions (without which most of cryptography collapses anyhow).

In summary: *not all assumptions are equal.* Thus, "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 gain confidence about the solvability of the new task, but we also obtain a solution based on a primitive that, being well-known, typically has several candidate implementations.

C.1.2 The Computational Model

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 Organization and Beyond

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, we start by presenting the natural concern underlying it, then define the problem, and finally demonstrate that the problem may be solved. In the latter step, our focus is on demonstrating the feasibility of solving the problem, not on providing a practical solution.

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.

Actual organization: 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, which play 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, 8.2, 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 [91, 92]. Furthermore, the first part (i.e., Basic Tools) corresponds to [91], whereas the second part (i.e., Basic Applications) corresponds to [92]. 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 much 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 [183]). 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 [193]: $x \mapsto x^e \bmod N$, 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 f_i). On input $i \in \overline{I}$ and $x \in D_i$,* (the evaluation) *algorithm F* always *outputs $f_i(x)$.*

2. Hard to invert:[2] *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.[3] 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).

[2]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.)

[3]Indeed, a more adequate term would be a collection of trapdoor permutations, but the shorter (and less precise) term is the commonly used one.

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, the function $f_{N,e}$ can be inverted by raising the image 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).

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 hypothetical insecurity of the resulting construction) into a violation of the hypothesis (i.e., insecurity 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 a typical f-image y, 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 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 many advanced cryptographic 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 the function f', where $f'(x,r) = (f(x), r)$.*

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 [146] or having large "linear-complexity" (as defined in [112]). 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. [108, 238]). 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 8.2.3), and consider two variants.

Definition C.4 (computational indistinguishability, Definition 8.4 revised[4]): *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.*

[4]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.)

See further discussion in Section 8.2.3. In particular, recall that for "efficiently constructible" distributions, indistinguishability by a single sample (as in Definition C.4) implies indistinguishability by multiple samples (as in Definition 8.5).

Extension to ensembles indexed by strings. We consider a natural extension of Definition C.4 in which, 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.

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 8.2.

C.3.3 Pseudorandom Functions

Recall that 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 [95], 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*

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

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. Thus, 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:[5] 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 $G(s)$. Let

$$G_{\sigma_{|s|}\cdots\sigma_2\sigma_1}(s) \stackrel{\text{def}}{=} G_{\sigma_{|s|}}(\cdots G_{\sigma_2}(G_{\sigma_1}(s))\cdots),$$

define $f_s(x_1x_2\cdots x_n) \stackrel{\text{def}}{=} G_{x_n\cdots x_2x_1}(s)$, and consider the function ensemble $\{f_s : \{0,1\}^{|s|} \to \{0,1\}^{|s|}\}_{s\in\{0,1\}^*}$. 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 the function ensemble $\{f_s\}_{s\in\{0,1\}^*}$ is pseudorandom. The proof uses the hybrid technique (cf. Section 8.2.3): The i^{th} hybrid, denoted 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 the indistinguishability of neighboring hybrids

[5]See details in [91, Sec. 3.6.2].

follows from our indistinguishability hypothesis (by using a reducibility argument). Specifically, we show that the ability to distinguish H_n^i from H_n^{i+1} yields an ability to distinguish multiple samples of $G(U_n)$ from multiple samples of U_{2n} (by placing on the fly, halves of the given samples at adequate vertices of the $i+1^{\rm st}$ level). $\square$

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. [91, Sec. 3.6.4] and [92, Apdx. C.2]).

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. The answer provided by the simulation paradigm 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.

The definitional approach to security represented by the simulation paradigm (and more generally the entire definitional approach surveyed in this appendix) may be considered overly cautious, because it seems to prohibit also "non-harmful" gains of some "far fetched" adversaries.[6] We warn against this impression. Firstly, there

[6]Indeed, according to the simulation paradigm, a system is called secure only if all possible

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 system 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 cryptographic system 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 9.1.2). 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 $z \in \{0,1\}^{p(|x|)}$.*

adversaries can be adequately simulated by adequate benign behavior. Thus, this approach considers also "far fetched" adversaries and does not disregard "non-harmful" gains that cannot be simulated.

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 [91, Sec. 4.3.4].

C.4.3 A General Result 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 [169, 118], *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).[7] 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 (Theorem 9.11, revisited)): *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 that 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.4.3.2). 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. At a later (i.e., second) stage, the commitment is "opened" and it is guaranteed that the "opening" can yield only a single value, which is determined

[7]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.4.3.2). 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. [169]): 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 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. For example, U may be instructed to send several different commitments (cf., §C.4.3.1) to a single secret value of its choice. 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 Definitional Variations and Related Notions

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 §9.1.5.2. We mention that argument systems may be more efficient than interactive proofs as well as provide stronger zero-knowledge guarantees. Specifically, almost-perfect zero-knowledge arguments for $\mathcal{NP}$ can be constructed based on any one-way function [172], where almost-perfect zero-knowledge means that the simulator's output is statistically close to the verifier's view in the real interaction (see a discussion in §C.4.4.1). Note that stronger security guarantee for the prover (as provided by almost-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. One 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 [98] 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 later refuted by Barak [25].

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)[8] 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 the case of "public-coin" proofs this is done without significantly increasing the prover's computational effort; see [228]).

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 extensively considered in the literature:

1. Perfect Zero-Knowledge requires that the two probability ensembles be identically distributed.[9]

2. Statistical (or Almost-Perfect) Zero-Knowledge requires that these probability ensembles be statistically close (i.e., the variation distance between them should be 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 [227].

[8]The term "honest verifier" is more appealing when considering an alternative (equivalent) formulation of Definition C.9. In the alternative definition (see [91, 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.

[9]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.

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).

Proofs of Knowledge. 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). 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 (NIZK) 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.[10] We mention that NIZK 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), which in turn motivate various augmentations of the basic definition of NIZK (see [91, Sec. 4.10] and [92, Sec. 5.4.4.4]). Such NIZK proofs for any NP-set can be constructed based on standard intractability assumptions (e.g., intractability of factoring), but even constructing basic NIZK proof systems seems more difficult than constructing *interactive* zero-knowledge proof systems.

Witness Indistinguishability. The notion of witness indistinguishability was suggested in [76] 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

[10]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.

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 protocols.* We refer, in particular, to the technique of [75] 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 [204], 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 [204]. 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 [66] (with the most popular candidate suggested by Rivest, Shamir, and Adleman [193]). 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).[11] 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 [108].

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 $|h(x)| = \text{poly}(|x|)$ 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)

[11] 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).

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).[12] As we shall see, this definition (also originating in [108]) is equivalent to Definition C.11.

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 $|z_n|=\mathrm{poly}(n)$, it holds that*

$$|\mathsf{Pr}[A(e,E_e(x_n),z_n)=1]-\mathsf{Pr}[A(e,E_e(y_n),z_n)=1]|=\mu(n)$$

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

[12]Indeed, satisfying this condition requires using a probabilistic encryption algorithm.

all-zero message from the encryption of the all-ones message.[13] This explains the association of the robust definitions of security with the paradigm of *probabilistic encryption*, an association that originates in the title of the pioneering work of Goldwasser and Micali [108].

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[14], 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. [92, 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*

[13]The same holds for (stateless) *private-key* encryption schemes, when considering the security of encrypting several messages (rather than a single message as in the foregoing text). 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.

[14]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).

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 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 of a private-key encryption scheme that uses pseudorandom functions as defined in Section C.3.3. The key-generation algorithm consists of uniformly selecting a seed $s \in \{0,1\}^n$ 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:

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 the security of public-key encryption schemes *even in the case of encrypting a single message.*

Following Goldwasser and Micali [108], 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 of public-key encryption schemes that employ a collection of trapdoor permutations, as defined in Definition C.3. Let $\{f_i : D_i \rightarrow D_i\}_i$ be such a collection, and let b be a corresponding hard-core predicate. In the first scheme, 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. (8.10)), and the fact that the proof of the pseudorandomness of Eq. (8.10) 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 [92, 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[15] 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 [92, Sec. 5.4.5].

[15]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 [92, Apdx. C.1].

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.

Signature Schemes: The need to discuss "digital signatures" (cf. [66, 182]) 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 requirements:

- 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 requirements:

- 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 rather approved) by the signer.

C.6.1 Definitions

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.*

One popular suggestion is signing messages by applying the inverse of a trapdoor permutation, where the trapdoor is used as a signing-key and the permutation itself is used (in the forward direction) towards verification. We warn that, in general, this scheme does not satisfy Definition C.15 (e.g., the permutation may be a homomorphism of some group).

C.6.2 Constructions

Secure *message authentication schemes* can be constructed using pseudorandom functions (or rather the generalized notion of pseudorandom functions discussed at the end of Section C.3.3). Specifically, the key-generation algorithm consists of uniformly 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 may be rather long). 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. 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 [110], 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 and public verification-key.[16] Intuitively, the gain in terms of security is that a full-fledged chosen message attack cannot be launched on a fixed instance of the underlying signature schemes (i.e., on the fixed verification-key that was published by the user 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 (which is coupled with the fixed and public verification-key), to random strings (or rather random verification-keys) as well as additional signatures that are each relative to a random and independently distributed signing-key (which is coupled with a freshly generated verification-key).

Authentication trees. The security benefits of the refreshing paradigm are amplified when combining it with the use of *authentication trees*. The idea is to use the public verification-key (only) 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

[16]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.

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, 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).[17] Hence, it suffices to use a signature scheme that is secure as long as it is applied 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 preimages. (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. In order to bridge the gap between (one-time) signature schemes that are applicable for signing short messages and schemes that are applicable for signing long messages, we use the *hashing paradigm.* 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 is sound provided that the hashing function belongs to a family of *collision-resistant hashing* (a.k.a *collision-free hashing*) functions. Loosely speaking, the collision-resistant requirement means that, 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

[17]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.

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 [171] or [92, 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 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[18] 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 [92, Sec. 7.7.2].

[18]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 $\ell' = \text{poly}(\ell)$), and F' is a function mapping $(m+1)$-long sequences to m-long sequences.

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 the ideal setting then the actual protocol "emulates the ideal setting" (i.e., "emulates a trusted party"), and thus is deemed secure. This approach can be applied in a variety of models, and is used to define the goals of security in these models.[19] We first discuss some of the parameters used in defining various models, and next demonstrate the application of the foregoing approach in two important cases. For further details, see [92, 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.[20] 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: Most works in cryptography assume that communication is *synchronous* and that point-to-point channels exist between every pair of processors (i.e., a *complete network*). It is further assumed that the adversary cannot modify (or omit or insert) messages sent over any communication channel that connects honest parties. In the *standard model*, the adversary may tap all communication channels, and thus obtain any message sent between honest parties. In an alternative model, called the private-channel model, one *postulates*

[19]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.

[20]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.

that the adversary cannot obtain messages sent between any pair of honest parties. Indeed, in some cases, the private-channel model can be emulated by the standard model (e.g., by using a secure encryption scheme).

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.[21]

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, and computationally-bounded adversary, and do not assume the existence of private channels. Our aim is to define multi-

[21]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 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).

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 "effective malfunctioning" of parties in secure protocols is restricted to what is postulated in the corresponding ideal model.

In light of the foregoing, we start by defining an ideal model (or rather the misbehavior allowed in it). 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. First, the members of this dishonest minority share their original inputs and decide 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 members of the dishonest minority share the output-messages and determine their local outputs based on all they know (i.e., their initial inputs and their received output-messages).

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' and when the trusted party employs the functionality f.*

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 all 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 defining security with respect to passive adversaries, both the scope of the real-model adversaries and the scope of the ideal-model adversaries change. 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 protocols 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, and 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, if the output of each party depends on the inputs of both parties, then 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.[22] The same phenomenon occurs even in the case that the two parties just wish to generate a common random value. In order to account for this phenomenon, when considering active adversaries in the two-party setting, we do not consider such premature suspension of the execution a breach of security. Consequently, we consider an ideal model in which 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 party. 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. If the first party is dishonest, then 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.)

[22]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 (being in majority and) by joining forces can do things that dishonest minorities cannot do: See §C.7.3.2.

We mention that an alternative way of dealing with the problem of premature suspension of execution (i.e., abort) is to restrict the 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 [92, 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 [100, 240, 101].

In the standard channel model. *Assuming the existence of enhanced*[23] *trapdoor permutations*, secure multi-party computation is possible in the following three models (cf. [100, 240, 101] and details in [92, Chap. 7]):

1. Passive adversaries, for any number of dishonest parties.

2. Active adversaries that may control only a minority of the parties.

3. Active adversaries, for any number of dishonest parties, provided that suspension of execution is not considered a violation of security (cf. §C.7.1.3).

In all these cases, the adversaries are computationally-bounded and non-adaptive. On the other hand, the adversaries 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. [34, 53]):

1. Passive adversaries that may control only a minority of the parties.

2. Active adversaries that may control only less than one third of the parties.

In both cases the adversaries may be adaptive.

[23]See Footnote 15.

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 [100, 240, 101]. These constructions proceed in two steps (see details in [92, 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 parties depending on their intermediate internal states (during the entire execution). 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 sake of simplicity, we consider here only the special case of *deterministic* m-ary functionalities (i.e., functions). We assume that the m 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 mod 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 (where 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 a 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 $z_{i,i} \stackrel{\rm def}{=} x_i y_i$.
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 [101]. A detailed description and full proofs appear in [92, 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 out 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 sender 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 triple $(\alpha, x_j, \sigma_j \oplus b(f_\alpha^{-1}(x_j)))$, 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.[24] (See [92, Sec. 7.3.2] for a detailed proof of security.)

[24]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.

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 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 (mentioned in §C.4.3.2) is using zero-knowledge proofs 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. [92, 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 its random-input with all other parties using a "Verifiable Secret Sharing" (VSS) protocol (cf. [92, 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 foregoing compilation was first outlined in [100, 101]. A detailed description and full proofs appear in [92, 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.

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 [92, 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 turns out that we cannot get all that we may want (and this is without mentioning the problems involved in preserving security under concurrent composition; cf. [92, 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.

Appendix D

Probabilistic Preliminaries and Advanced Topics in Randomization

What is this? Chicken Curry and Seafood Salad?
Fine, but in the same plate? This is disgusting!

Johan Håstad at Grendel's, Cambridge (1985)

Summary: This appendix lumps together some preliminaries regarding probability theory and some advanced topics related to the role and use of randomness in computation. Needless to say, each of these topics appears in a separate section.

The probabilistic preliminaries include our conventions regarding random variables, which are used throughout the book. Also included are overviews of three useful probabilistic inequalities: Markov's Inequality, Chebyshev's Inequality, and Chernoff Bound.

The advanced topics include hashing, sampling, and randomness extraction. For hashing, we describe constructions of pairwise (and t-wise independent) hashing functions and (a few variants of) the Leftover Hashing Lemma (used a few times in the main text). We then review the "complexity of sampling": that is, the number of samples and the randomness complexity involved in estimating the average value of an arbitrary function defined over a huge domain. Finally, we provide an overview on the question of extracting almost perfect randomness from sources of weak (or defected) randomness.

D.1 Probabilistic preliminaries

Probability plays a central role in complexity theory (see, for example, Chapters 6–9). We assume that the reader is familiar with the basic notions of probability theory. In this section, we merely present the probabilistic notations that are used throughout the book and three useful probabilistic inequalities.

D.1.1 Notational Conventions

Throughout the entire book we refer only to *discrete* probability distributions. Specifically, the underlying probability space consists of the set of all strings of a certain length ℓ, taken with uniform probability distribution. That is, the sample space is the set of all ℓ-bit long strings, and each such string is assigned probability measure $2^{-\ell}$. Traditionally, *random variables* are defined as functions from the sample space to the reals. Abusing the traditional terminology, we use the term random variable also when referring to functions mapping the sample space into the set of binary strings. We often do not specify the probability space, but rather talk directly about random variables. For example, we may say that X is a random variable assigned values in the set of all strings such that $\mathsf{Pr}[X = 00] = \frac{1}{4}$ and $\mathsf{Pr}[X = 111] = \frac{3}{4}$. (Such a random variable may be defined over the sample space $\{0,1\}^2$, so that $X(11) = 00$ and $X(00) = X(01) = X(10) = 111$.) One important case of a random variable is the output of a randomized process (e.g., a probabilistic polynomial-time algorithm, as in Section 6.1).

All our probabilistic statements refer to random variables that are defined beforehand. Typically, we may write $\mathsf{Pr}[f(X) = 1]$, where X is a random variable defined beforehand (and f is a function). An important convention is that *all occurrences of the same symbol in a probabilistic statement refer to the same* (unique) *random variable*. Hence, if $B(\cdot,\cdot)$ is a Boolean expression depending on two variables, and X is a random variable then $\mathsf{Pr}[B(X,X)]$ denotes the probability that $B(x,x)$ holds when x is chosen with probability $\mathsf{Pr}[X=x]$. For example, for every random variable X, we have $\mathsf{Pr}[X=X] = 1$. We stress that if we wish to discuss the probability that $B(x,y)$ holds when x and y are chosen independently with identical probability distribution, then we will define *two* independent random variables each with the same probability distribution. Hence, if X and Y are two independent random variables then $\mathsf{Pr}[B(X,Y)]$ denotes the probability that $B(x,y)$ holds when the pair (x,y) is chosen with probability $\mathsf{Pr}[X=x] \cdot \mathsf{Pr}[Y=y]$. For example, for every two independent random variables, X and Y, we have $\mathsf{Pr}[X = Y] = 1$ only if both X and Y are trivial (i.e., assign the entire probability mass to a single string).

Throughout the entire book, U_n denotes a random variable uniformly distributed over the set of all strings of length n. Namely, $\mathsf{Pr}[U_n = \alpha]$ equals 2^{-n} if $\alpha \in \{0,1\}^n$ and equals 0 otherwise. We often refer to the distribution of U_n as the uniform distribution (neglecting to qualify that it is uniform over $\{0,1\}^n$). In addition, we occasionally use random variables (arbitrarily) distributed over $\{0,1\}^n$ or $\{0,1\}^{\ell(n)}$, for some function $\ell : \mathbb{N}\to\mathbb{N}$. Such random variables are typically denoted by X_n, Y_n, Z_n, etc. We stress that in some cases X_n is distributed over

$\{0,1\}^n$, whereas in other cases it is distributed over $\{0,1\}^{\ell(n)}$, for some function ℓ (which is typically a polynomial). We often talk about probability ensembles, which are infinite sequence of random variables $\{X_n\}_{n\in\mathbb{N}}$ such that each X_n ranges over strings of length bounded by a polynomial in n.

Statistical difference. The statistical distance (a.k.a variation distance) between the random variables X and Y is defined as

$$\frac{1}{2}\cdot\sum_v |\mathsf{Pr}[X=v]-\mathsf{Pr}[Y=v]| \;=\; \max_S\{\mathsf{Pr}[X\in S]-\mathsf{Pr}[Y\in S]\}. \qquad \text{(D.1)}$$

We say that X is δ-close (resp., δ-far) to Y if the statistical distance between them is at most (resp., at least) δ.

D.1.2 Three Inequalities

The following probabilistic inequalities are very useful. These inequalities refer to random variables that are assigned real values and provide upper-bounds on the probability that the random variable deviates from its expectation.

D.1.2.1 Markov's Inequality

The most basic inequality is Markov's Inequality that applies to any random variable with bounded maximum or minimum value. For simplicity, this inequality is stated for random variables that are lower-bounded by zero, and reads as follows: *Let X be a non-negative random variable and v be a non-negative real number. Then*

$$\mathsf{Pr}\left[X\geq v\right]\leq\frac{\mathsf{E}(X)}{v} \qquad \text{(D.2)}$$

Equivalently, $\mathsf{Pr}[X\geq r\cdot\mathsf{E}(X)]\leq\frac{1}{r}$. The proof amounts to the following sequence:

$$\begin{aligned}\mathsf{E}(X) &= \sum_x \mathsf{Pr}[X=x]\cdot x\\ &\geq \sum_{x<v}\mathsf{Pr}[X=x]\cdot 0+\sum_{x\geq v}\mathsf{Pr}[X=x]\cdot v\\ &= \mathsf{Pr}[X\geq v]\cdot v\end{aligned}$$

D.1.2.2 Chebyshev's Inequality

Using Markov's inequality, one gets a potentially stronger bound on the deviation of a random variable from its expectation. This bound, called Chebyshev's inequality, is useful when having additional information concerning the random variable (specifically, a good upper bound on its variance). For a random variable X of finite expectation, we denote by $\mathsf{Var}(X)\stackrel{\rm def}{=}\mathsf{E}[(X-\mathsf{E}(X))^2]$ the variance of X, and observe that $\mathsf{Var}(X) = \mathsf{E}(X^2) - \mathsf{E}(X)^2$. Chebyshev's Inequality then reads as follows: *Let X be a random variable, and $\delta > 0$. Then*

$$\mathsf{Pr}\left[|X - \mathsf{E}(X)| \geq \delta\right] \leq \frac{\mathsf{Var}(X)}{\delta^2} \tag{D.3}$$

Proof: We define a random variable $Y \stackrel{\text{def}}{=} (X - \mathsf{E}(X))^2$, and apply Markov's inequality. We get

$$\begin{aligned} \mathsf{Pr}\left[|X - \mathsf{E}(X)| \geq \delta\right] &= \mathsf{Pr}\left[(X - \mathsf{E}(X))^2 \geq \delta^2\right] \\ &\leq \frac{\mathsf{E}[(X - \mathsf{E}(X))^2]}{\delta^2} \end{aligned}$$

and the claim follows. ■

Corollary (Pairwise Independent Sampling)**:** Chebyshev's inequality is particularly useful in the analysis of the error probability of approximation via repeated sampling. It suffices to assume that the samples are picked in a pairwise independent manner, where $X_1, X_2, ..., X_n$ are pairwise independent if for every $i \neq j$ and every α, β it holds that $\mathsf{Pr}[X_i = \alpha \wedge X_j = \beta] = \mathsf{Pr}[X_i = \alpha] \cdot \mathsf{Pr}[X_j = \beta]$. The corollary reads as follows: *Let $X_1, X_2, ..., X_n$ be pairwise independent random variables with identical expectation, denoted μ, and identical variance, denoted σ^2. Then, for every $\varepsilon > 0$, it holds that*

$$\mathsf{Pr}\left[\left|\frac{\sum_{i=1}^n X_i}{n} - \mu\right| \geq \varepsilon\right] \leq \frac{\sigma^2}{\varepsilon^2 n} \tag{D.4}$$

Proof: Define the random variables $\overline{X}_i \stackrel{\text{def}}{=} X_i - \mathsf{E}(X_i)$. Note that the $\overline{X}_i$'s are pairwise independent, and each has zero expectation. Applying Chebyshev's inequality to the random variable $\sum_{i=1}^n \frac{X_i}{n}$, and using the linearity of the expectation operator, we get

$$\begin{aligned} \mathsf{Pr}\left[\left|\sum_{i=1}^n \frac{X_i}{n} - \mu\right| \geq \varepsilon\right] &\leq \frac{\mathsf{Var}\left[\sum_{i=1}^n \frac{X_i}{n}\right]}{\varepsilon^2} \\ &= \frac{\mathsf{E}\left[\left(\sum_{i=1}^n \overline{X}_i\right)^2\right]}{\varepsilon^2 \cdot n^2} \end{aligned}$$

Now (again using the linearity of expectation)

$$\mathsf{E}\left[\left(\sum_{i=1}^n \overline{X}_i\right)^2\right] = \sum_{i=1}^n \mathsf{E}\left[\overline{X}_i^2\right] + \sum_{1 \leq i \neq j \leq n} \mathsf{E}\left[\overline{X}_i \overline{X}_j\right]$$

By the pairwise independence of the $\overline{X}_i$'s, we get $\mathsf{E}[\overline{X}_i \overline{X}_j] = \mathsf{E}[\overline{X}_i] \cdot \mathsf{E}[\overline{X}_j]$, and using $\mathsf{E}[\overline{X}_i] = 0$, we get

$$\mathsf{E}\left[\left(\sum_{i=1}^n \overline{X}_i\right)^2\right] = n \cdot \sigma^2$$

The corollary follows. ■

D.1.2.3 Chernoff Bound

When using pairwise independent sample points, the error probability in the approximation decreases linearly with the number of sample points (see Eq. (D.4)). When using totally independent sample points, the error probability in the approximation can be shown to decrease exponentially with the number of sample points. (Recall that the random variables $X_1, X_2, ..., X_n$ are said to be totally independent if for every sequence $a_1, a_2, ..., a_n$ it holds that $\mathsf{Pr}[\wedge_{i=1}^n X_i = a_i] = \prod_{i=1}^n \mathsf{Pr}[X_i = a_i]$.) Probability bounds supporting the foregoing statement are given next. The first bound, commonly referred to as Chernoff Bound, concerns 0-1 random variables (i.e., random variables that are assigned as values either 0 or 1), and asserts the following. *Let $p \leq \frac{1}{2}$, and $X_1, X_2, ..., X_n$ be independent 0-1 random variables such that $\mathsf{Pr}[X_i = 1] = p$, for each i. Then, for every $\varepsilon \in (0, p]$, it holds that*

$$\mathsf{Pr}\left[\left|\frac{\sum_{i=1}^n X_i}{n} - p\right| > \varepsilon\right] \; < \; 2 \cdot e^{-c \cdot \varepsilon^2 \cdot n}\,, \text{ where } c = \max(2, \tfrac{1}{3p}). \qquad \text{(D.5)}$$

The more common formulation sets $c = 2$, but the case $c = 1/3p$ is very useful when p is small and one cares about a *multiplicative* deviation (e.g., $\varepsilon = p/2$).

Proof Sketch: We upper-bound $\mathsf{Pr}[\sum_{i=1}^n X_i - pn > \varepsilon n]$, and $\mathsf{Pr}[pn - \sum_{i=1}^n X_i > \varepsilon n]$ is bounded similarly. Letting $\overline{X}_i \stackrel{\text{def}}{=} X_i - \mathsf{E}(X_i)$, we apply Markov's inequality to the random variable $e^{\lambda \sum_{i=1}^n \overline{X}_i}$, where $\lambda \in (0, 1]$ will be determined to optimize the expressions that we derive. Thus, $\mathsf{Pr}[\sum_{i=1}^n \overline{X}_i > \varepsilon n]$ is upper-bounded by

$$\frac{\mathsf{E}[e^{\lambda \sum_{i=1}^n \overline{X}_i}]}{e^{\lambda \varepsilon n}} = e^{-\lambda \varepsilon n} \cdot \prod_{i=1}^n \mathsf{E}[e^{\lambda \overline{X}_i}]$$

where the equality is due to the independence of the random variables. To simplify the rest of the proof, we establish a sub-optimal bound as follows. Using a Taylor expansion of e^x (e.g., $e^x < 1 + x + x^2$ for $|x| \leq 1$) and observing that $\mathsf{E}[\overline{X}_i] = 0$, we get $\mathsf{E}[e^{\lambda \overline{X}_i}] < 1 + \lambda^2 \mathsf{E}[\overline{X}_i^2]$, which equals $1 + \lambda^2 p(1-p)$. Thus, $\mathsf{Pr}[\sum_{i=1}^n X_i - pn > \varepsilon n]$ is upper-bounded by $e^{-\lambda \varepsilon n} \cdot (1 + \lambda^2 p(1-p))^n < \exp(-\lambda \varepsilon n + \lambda^2 p(1-p)n)$, which is optimized at $\lambda = \varepsilon/(2p(1-p))$ yielding $\exp(-\frac{\varepsilon^2}{4p(1-p)} \cdot n) \leq \exp(-\varepsilon^2 \cdot n)$. □

The foregoing proof strategy can be applied in more general settings.[1] A more general bound, which refers to independent random variables that are each bounded but are not necessarily identical, is given next (and is commonly referred to as Hoefding Inequality). *Let $X_1, X_2, ..., X_n$ be n independent random variables, each ranging in the (real) interval $[a, b]$, and let $\mu \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^n \mathsf{E}(X_i)$ denote the average*

[1]For example, verify that the current proof actually applies to the case that $X_i \in [0, 1]$ rather than $X_i \in \{0, 1\}$, by noting that $\mathsf{Var}[X_i] \leq p(1-p)$ still holds.

expected value of these variables. Then, for every $\varepsilon > 0$,

$$\mathsf{Pr}\left[\left|\frac{\sum_{i=1}^{n} X_i}{n} - \mu\right| > \varepsilon\right] < 2 \cdot e^{-\frac{2\varepsilon^2}{(b-a)^2}\cdot n} \tag{D.6}$$

The special case (of Eq. (D.6)) that refers to identically distributed random variables is easy to derive from the foregoing Chernoff Bound (by recalling Footnote 1 and using a linear mapping of the interval $[a, b]$ to the interval $[0, 1]$). This special case is useful in estimating the average value of a (bounded) function defined over a large domain, especially when the desired error probability needs to be negligible (i.e., decrease faster than any polynomial in the number of samples). Such an estimate can be obtained provided that we can sample the function's domain (and evaluate the function).

D.1.2.4 Pairwise independent versus totally independent sampling

To demonstrate the difference between the sampling bounds provided in §D.1.2.2 and §D.1.2.3, we consider the problem of estimating the average value of a function $f : \Omega \to [0, 1]$. In general, we say that a random variable Z provides an (ε, δ)-approximation of a value v if $\mathsf{Pr}[|Z - v| > \varepsilon] \leq \delta$. By Eq. (D.6), the average value of f evaluated at $n = O((\varepsilon^{-2} \cdot \log(1/\delta))$ *independent* samples (selected uniformly in Ω) yield an (ε, δ)-approximation of $\mu = \sum_{x\in\Omega} f(x)/|\Omega|$. Thus, the number of sample points is polynomially related to ε^{-1} and logarithmically related to δ^{-1}. In contrast, by Eq. (D.4), an (ε, δ)-approximation by n *pairwise independent* samples calls for setting $n = O(\varepsilon^{-2} \cdot \delta^{-1})$. We stress that, *in both cases the number of samples is polynomially related to the desired accuracy of the estimation* (i.e., ε). *The only advantage of totally independent samples over pairwise independent ones is in the dependency of the number of samples on the error probability* (i.e., δ).

D.2 Hashing

Hashing is extensively used in complexity theory (see, e.g., §6.2.2.2, Section 6.2.3, §6.2.4.2, §8.2.5.3, and §8.4.2.1). The typical application is for mapping arbitrary (unstructured) sets "almost uniformly" to a structured set of adequate size. Specifically, hashing is used for mapping an arbitrary 2^m-subset of $\{0, 1\}^n$ to $\{0, 1\}^m$ in an "almost uniform" manner.

For any fixed set S of cardinality 2^m, there exists a 1-1 mapping $f_S : S \to \{0, 1\}^m$, but this mapping is not necessarily efficiently computable (e.g., it may require "knowing" the entire set S). On the other hand, no single function $f : \{0, 1\}^n \to \{0, 1\}^m$ can map every 2^m-subset of $\{0, 1\}^n$ to $\{0, 1\}^m$ in a 1-1 manner (or even approximately so). Nevertheless, for every 2^m-subset $S \subset \{0, 1\}^n$, a random function $f : \{0, 1\}^n \to \{0, 1\}^m$ has the property that, with overwhelmingly high probability, f maps S to $\{0, 1\}^m$ such that no point in the range has too many f-preimages in S. The problem is that a truly random function is unlikely to have a succinct representation (let alone an efficient evaluation algorithm). We thus seek families of functions that have a "random mapping" property (as in Item 1 of the following definition), but do have a succinct representation as well as an efficient evaluation algorithm (as in Items 2 and 3 of the following definition).

D.2.1 Definitions

Motivated by the foregoing discussion, we consider families of functions $\{H_n^m\}_{m<n}$ such that the following properties hold:

1. For every $S \subset \{0,1\}^n$, with high probability, a function h selected uniformly in H_n^m maps S to $\{0,1\}^m$ in an "almost uniform" manner. For example, we may require that, for any $|S| = 2^m$ and each point y, with high probability over the choice of h, it holds that $|\{x \in S : h(x) = y\}| \leq \text{poly}(n)$.
2. The functions in H_n^m have succinct representation. For example, we may require that $H_n^m \equiv \{0,1\}^{\ell(n,m)}$, for some polynomial ℓ.
3. The functions in H_n^m can be efficiently evaluated. That is, there exists a polynomial-time algorithm that, on input a representation of a function, h (in H_n^m), and a string $x \in \{0,1\}^n$, returns $h(x)$. In some cases we make even more stringent requirements regarding the algorithm (e.g., that it runs in linear space).

Condition 1 was left vague on purpose. At the very least, we require that the expected size of $\{x \in S : h(x) = y\}$ equals $|S|/2^m$. We shall see (in Section D.2.3) that different interpretations of Condition 1 are satisfied by different families of hashing functions. We focus on t-wise independent hashing functions, defined next.

Definition D.1 (t-wise independent hashing functions): *A family H_n^m of functions from n-bit strings to m-bit strings is called* t-wise independent *if for every t distinct domain elements $x_1, ..., x_t \in \{0,1\}^n$ and every $y_1, ..., y_t \in \{0,1\}^m$ it holds that*

$$\mathsf{Pr}_{h \in H_n^m}[\wedge_{i=1}^t h(x_i) = y_i] \;=\; 2^{-t \cdot m}$$

That is, a uniformly chosen $h \in H_n^m$ maps every t domain elements to the range in a totally uniform manner. Note that for $t \geq 2$, it follows that the probability that a random $h \in H_n^m$ maps two distinct domain elements to the same image equals 2^{-m}. Such (families of) functions are called universal (cf. [50]), but we will focus on the stronger condition of t-wise independence.

D.2.2 Constructions

The following constructions are merely a re-interpretation of the constructions presented in §8.5.1.1. (Alternatively, one may view the constructions presented in §8.5.1.1 as a re-interpretation of the following two constructions.)

Construction D.2 (t-wise independent hashing): *For $t, m, n \in \mathbb{N}$ such that $m \leq n$, consider the following family of hashing functions mapping n-bit strings to m-bit strings. Each t-sequence $\overline{s} = (s_0, s_1, ..., s_{t-1}) \in \{0,1\}^{t \cdot n}$ describes a function*

$h_{\overline{s}} : \{0,1\}^n \to \{0,1\}^m$ such that $h_{\overline{s}}(x)$ equals the m-bit prefix of the binary representation of $\sum_{j=0}^{t-1} s_j x^j$, where the arithmetic is that of GF(2^n)*, the finite field of 2^n elements.*

Proposition 8.24 implies that Construction D.2 constitutes a family of t-wise independent hash functions. Typically, we will use either $t = 2$ or $t = \Theta(n)$. To make the construction totally explicit, we need an explicit representation of GF(2^n); see comment following Proposition 8.24. An alternative construction for the case of $t = 2$ may be obtained analogously to the pairwise independent generator of Proposition 8.25. Recall that a Toeplitz matrix is a matrix with all diagonals being homogeneous; that is, $T = (t_{i,j})$ is a Toeplitz matrix if $t_{i,j} = t_{i+1,j+1}$, for all i, j.

Construction D.3 (alternative pairwise independent hashing): *For $m \leq n$, consider the family of hashing functions in which each pair (T, b), consisting of a n-by-m Toeplitz matrix T and an m-dimensional vector b, describes a function $h_{T,b} : \{0,1\}^n \to \{0,1\}^m$ such that $h_{T,b}(x) = Tx + b$.*

Proposition 8.25 implies that Construction D.3 constitutes a family of pairwise independent hash functions. Note that a n-by-m Toeplitz matrix can be specified by $n + m - 1$ bits, yielding a description length of $n + 2m - 1$ bits. An alternative construction (analogous to Eq. (8.23) and requiring $m \cdot n + m$ bits of representation) uses arbitrary n-by-m matrices rather than Toeplitz matrices.

D.2.3 The Leftover Hash Lemma

We now turn to the "almost uniform" cover condition (i.e., Condition 1) mentioned in Section D.2.1. One concrete interpretation of this condition is given by the following lemma (and another interpretation is implied by it: see Theorem D.5).

Lemma D.4 *Let $m \leq n$ be integers, H_n^m be a family of pairwise independent hash functions, and $S \subseteq \{0,1\}^n$. Then, for every $y \in \{0,1\}^m$ and every $\varepsilon > 0$, for all but at most an $\frac{2^m}{\varepsilon^2 |S|}$ fraction of $h \in H_n^m$ it holds that*

$$(1-\varepsilon) \cdot \frac{|S|}{2^m} \; < \; |\{x \in S : h(x) = y\}| \; < \; (1+\varepsilon) \cdot \frac{|S|}{2^m}. \tag{D.7}$$

Note that by pairwise independence (or rather even by 1-wise independence), the expected size of $\{x \in S : h(x) = y\}$ is $|S|/2^m$, where the expectation is taken uniformly over all $h \in H_n^m$. The lemma upper bounds the fraction of h's that deviate from the expected behavior (i.e., for which $|h^{-1}(y) \cap S| \neq (1 \pm \varepsilon) \cdot |S|/2^m$). Needless to say, the bound is meaningful only in case $|S| > 2^m/\varepsilon^2$. Focusing on the case that $|S| > 2^m$ and setting $\varepsilon = \sqrt[3]{2^m/|S|}$, we infer that *for all but at most an ε fraction of $h \in H_n^m$ it holds that* $|\{x \in S : h(x) = y\}| = (1 \pm \varepsilon) \cdot |S|/2^m$. Thus, each range element has approximately the right number of h-preimages in the set S, under almost all $h \in H_n^m$.

Proof: Fixing an arbitrary set $S \subseteq \{0,1\}^n$ and an arbitrary $y \in \{0,1\}^m$, we estimate the probability that a uniformly selected $h \in H_n^m$ violates Eq. (D.7). We define random variables ζ_x, over the aforementioned probability space, such that $\zeta_x = \zeta_x(h)$ equal 1 if $h(x) = y$ and $\zeta_x = 0$ otherwise. The expected value of $\sum_{x \in S} \zeta_x$ is $\mu \stackrel{\rm def}{=} |S| \cdot 2^{-m}$, and we are interested in the probability that this sum deviates from the expectation. Applying Chebyshev's Inequality, we get

$$\mathsf{Pr}\left[\left|\mu - \sum_{x \in S} \zeta_x\right| \geq \varepsilon \cdot \mu\right] \;<\; \frac{\mu}{\varepsilon^2 \mu^2}$$

because $\mathsf{Var}[\sum_{x \in S} \zeta_x] < |S| \cdot 2^{-m}$ by the pairwise independence of the ζ_x's and the fact that $\mathsf{E}[\zeta_x] = 2^{-m}$. The lemma follows. ■

A generalization (called mixing). The proof of Lemma D.4 can be easily extended to show that *for every set $T \subset \{0,1\}^m$ and every $\varepsilon > 0$, for all but at most an $\frac{2^m}{|T| \cdot |S| \varepsilon^2}$ fraction of $h \in H_n^m$ it holds that $|\{x \in S : h(x) \in T\}| = (1 \pm \varepsilon) \cdot |T| \cdot |S|/2^m$.* (Hint: redefine $\zeta_x = \zeta(h) = 1$ if $h(x) \in T$ and $\zeta_x = 0$ otherwise.) This assertion is meaningful provided that $|T| \cdot |S| > 2^m/\varepsilon^2$, and in the case that $m = n$ it is called a mixing property.

An extremely useful corollary. The aforementioned generalization of Lemma D.4 asserts that, for any fixed set of preimages $S \subset \{0,1\}^n$ and any fixed sets of images $T \subset \{0,1\}^m$, most functions in H_n^m behave well with respect to S and T (in the sense that they map approximately the adequate fraction of S (i.e., $|T|/2^m$) to T). A seemingly stronger statement, which is (non-trivially) implied by Lemma D.4 itself, reverses the order of quantification with respect to T; that is, for all adequate sets S, most functions in H_n^m map S to $\{0,1\}^m$ in an almost uniform manner (i.e., assign each set T approximately the adequate fraction of S, where here the approximation is up to an additive deviation). As we shall see, this is a consequence of the following theorem.

Theorem D.5 (a.k.a Leftover Hash Lemma): *Let H_n^m and $S \subseteq \{0,1\}^n$ be as in Lemma D.4, and define $\varepsilon = \sqrt[3]{2^m/|S|}$. Consider random variables X and H that are uniformly distributed on S and H_n^m, respectively. Then, the statistical distance between $(H, H(X))$ and (H, U_m) is at most 2ε.*

It follows that, *for X and ε as in Theorem D.5 and any $\alpha > 0$, for all but at most an α fraction of the functions $h \in H_n^m$ it holds that $h(X)$ is $(2\varepsilon/\alpha)$-close to U_m.*[2] (Using the terminology of the subsequent Section D.4, we may say that Theorem D.5 asserts that H_n^m yields a strong extractor (with parameters to be spelled out there).)

Proof: Let V denote the set of pairs (h, y) that violate Eq. (D.7), and $\overline{V} \stackrel{\rm def}{=} (H_n^m \times \{0,1\}^m) \setminus V$. Then for every $(h, y) \in \overline{V}$ it holds that

$$\begin{aligned} \mathsf{Pr}[(H, H(X)) = (h, y)] &= \mathsf{Pr}[H = h] \cdot \mathsf{Pr}[h(X) = y] \\ &= (1 \pm \varepsilon) \cdot \mathsf{Pr}[(H, U_m) = (h, y)]. \end{aligned}$$

[2]This follows by defining a random variable $\zeta = \zeta(h)$ such that ζ equals the statistical distance between $h(X)$ and U_m, and applying Markov's inequality.

On the other hand, by the setting of ε and Lemma D.4 (which imply that $\mathsf{Pr}[(H,y) \in V] \le \varepsilon$ for every $y \in \{0,1\}^m$), we have $\mathsf{Pr}[(H,U_m) \in V] \le \varepsilon$. It follows that

$$\begin{aligned} \mathsf{Pr}[(H,H(X)) \in V] &= 1 - \mathsf{Pr}[(H,H(X)) \in \overline{V}] \\ &\le 1 - \mathsf{Pr}[(H,U_m)) \in \overline{V}] + \varepsilon \ \le \ 2\varepsilon. \end{aligned}$$

Using all these upper-bounds, we upper-bounded the statistical difference between $(H,H(X))$ and (H,U_m), denoted Δ, by separating the contribution of V and $\overline{V}$. Specifically, we have

$$\begin{aligned} \Delta &= \frac{1}{2} \cdot \sum_{(h,y)\in H_n^m \times \{0,1\}^m} |\mathsf{Pr}[(H,H(X))=(h,y)] - \mathsf{Pr}[(H,U_m)=(h,y)]| \\ &\le \frac{\varepsilon}{2} + \frac{1}{2} \cdot \sum_{(h,y)\in V} |\mathsf{Pr}[(H,H(X))=(h,y)] - \mathsf{Pr}[(H,U_m)=(h,y)]| \ , \end{aligned}$$

where the first term upper-bounds the contribution of all pairs $(h,y) \in \overline{V}$. Hence,

$$\begin{aligned} \Delta &\le \frac{\varepsilon}{2} + \frac{1}{2} \cdot \sum_{(h,y)\in V} (\mathsf{Pr}[(H,H(X))=(h,y)] + \mathsf{Pr}[(H,U_m)=(h,y)]) \\ &\le \frac{\varepsilon}{2} + \frac{1}{2} \cdot (2\varepsilon + \varepsilon) \ , \end{aligned}$$

where the first inequality is trivial (i.e., $|\alpha - \beta| \le \alpha + \beta$ for any non-negative α and β), and the second inequality uses the foregoing upper-bounds (i.e., $\mathsf{Pr}[(H,H(X)) \in V] \le 2\varepsilon$ and $\mathsf{Pr}[(H,U_m) \in V] \le \varepsilon$). The theorem follows. ■

An alternative proof of Theorem D.5. Define the collision probability of a random variable Z, denote $\mathsf{cp}(Z)$, as the probability that two independent samples of Z yield the same result. Alternatively, $\mathsf{cp}(Z) \stackrel{\rm def}{=} \sum_z \mathsf{Pr}[Z=z]^2$. Theorem D.5 follows by combining the following two facts:

1. A general fact: *If $Z \in [N]$ and $\mathsf{cp}(Z) \le (1+4\epsilon^2)/N$ then Z is ϵ-close to the uniform distribution on $[N]$.*

   We prove the contra-positive: Assuming that the statistical distance between Z and the uniform distribution on $[N]$ equals δ, we show that $\mathsf{cp}(Z) \ge (1+4\delta^2)/N$. This is done by defining $L \stackrel{\rm def}{=} \{z : \mathsf{Pr}[Z=z] < 1/N\}$, and lower-bounding $\mathsf{cp}(Z)$ by using the fact that the collision probability is minimized on uniform distributions. Specifically, considering the uniform distributions on L and $[N] \setminus L$ respectively, we have

   $$\mathsf{cp}(Z) \ge |L| \cdot \left(\frac{\mathsf{Pr}[Z \in L]}{|L|}\right)^2 + (N - |L|) \cdot \left(\frac{\mathsf{Pr}[Z \in [N] \setminus L]}{N - |L|}\right)^2_{.} \qquad \text{(D.8)}$$

   Using $\delta = \rho - \mathsf{Pr}[Z \in L]$, where $\rho = |L|/N$, the r.h.s of Eq. (D.8) equals $\frac{(\rho-\delta)^2}{\rho N} + \frac{(1-(\rho-\delta))^2}{(1-\rho)N} = \left(1 + \frac{\delta^2}{(1-\rho)\rho}\right) \cdot \frac{1}{N} \ge \left(1+4\delta^2\right) \cdot \frac{1}{N}$.

2. *The collision probability of* $(H, H(X))$ *is at most* $(1 + (2^m/|S|))/(|H_n^m| \cdot 2^m)$. (Furthermore, this holds even if H_n^m is only universal.)

   The proof is by a straightforward calculation. Specifically, note that $\mathsf{cp}(H, H(X)) = |H_n^m|^{-1} \cdot \mathsf{E}_{h \in H_n^m}[\mathsf{cp}(h(X))]$, whereas $\mathsf{E}_{h \in H_n^m}[\mathsf{cp}(h(X))] = |S|^{-2} \sum_{x_1, x_2 \in S} \mathsf{Pr}[H(x_1) = H(x_2)]$. The sum equals $|S| + (|S|^2 - |S|) \cdot 2^{-m}$, and so $\mathsf{cp}(H, H(X)) < |H_n^m|^{-1} \cdot (2^{-m} + |S|^{-1})$.

It follows that $(H, H(X))$ is $2\sqrt{2^m/|S|}$-close to (H, U_m), which is actually a stronger bound than the one asserted by Theorem D.5.

Stronger uniformity via higher independence. Recall that Lemma D.4 asserts that for each point in the range of the hash function, with high probability over the choice of the hash function, *this fixed point* has approximately the expected number of preimages in S. A stronger condition asserts that, with high probability over the choice of the hash function, *every point in its range* has approximately the expected number of preimages in S. Such a guarantee can be obtained when using n-wise independent hash functions (rather than using pairwise independent hash functions).

Lemma D.6 *Let* $m \leq n$ *be integers,* H_n^m *be a family of* n*-wise independent hash functions, and* $S \subseteq \{0,1\}^n$. *Then, for every* $\varepsilon \in (0,1)$, *for all but at most an* $2^m \cdot (n \cdot 2^m/\varepsilon^2|S|)^{n/2}$ *fraction of the functions* $h \in H_n^m$, *it is the case that Eq.* (D.7) *holds for every* $y \in \{0,1\}^m$.

Indeed, the lemma should be used with $2^m < \varepsilon^2|S|/4n$. In particular, using $m = \log_2|S| - \log_2(5n/\varepsilon^2)$ guarantees that with high probability (i.e., $1 - 2^m \cdot 5^{-n/2} \geq 1 - (4/5)^{n/2}$) each range elements has $(1 \pm \varepsilon) \cdot |S|/2^m$ preimages in S. Under this setting of parameters $|S|/2^m = 5n/\varepsilon^2$, which is $\text{poly}(n)$ whenever $\varepsilon = 1/\text{poly}(n)$. Needless to say, this guarantee is stronger than the conclusion of Theorem D.5.

Proof: The proof follows the footsteps of the proof of Lemma D.4, taking advantage of the fact that here the random variables (i.e., the ζ_x's) are n-wise independent. For $t = n/2$, this allows using the so-called $2t$th *moment analysis*, which generalizes the second moment analysis of pairwise independent samplying (presented in §D.1.2.2). As in the proof of Lemma D.4, we fix any S and y, and define $\zeta_x = \zeta_x(h) = 1$ if and only if $h(x) = y$. Letting $\mu = \mathsf{E}[\sum_{x \in S} \zeta_x] = |S|/2^m$ and $\overline{\zeta}_x = \zeta_x - \mathsf{E}(\zeta_x)$, we start with Markov's inequality:

$$\begin{aligned} \mathsf{Pr}\left[\left|\mu - \sum_{x \in S} \zeta_x\right| \geq \varepsilon \cdot \mu\right] &\leq \frac{\mathsf{E}[(\sum_{x \in S} \overline{\zeta}_x)^{2t}]}{\varepsilon^{2t}\mu^{2t}} \\ &= \frac{\sum_{x_1,\ldots,x_{2t} \in S} \mathsf{E}[\prod_{i=1}^{2t} \overline{\zeta}_{x_i}]}{\varepsilon^{2t} \cdot (|S|/2^m)^{2t}} \qquad \text{(D.9)} \end{aligned}$$

Using $2t$-wise independence, we note that only the terms in Eq. (D.9) that do not vanish are those in which each variable appears with multiplicity. This mean that only terms having less than t distinct variables contribute to Eq. (D.9). Now, for every $j \leq t$, we have less than $\binom{|S|}{j} \cdot (2t!) < (2t!/j!) \cdot |S|^j$ terms with j distinct variables, and each such term contributes less than $(2^{-m})^j$ to the sum (because for every $e > 1$ it holds that $\mathsf{E}[\overline{\zeta}_{x_i}^e] < \mathsf{E}[\zeta_{x_i}] = 2^{-m}$). Thus, Eq. (D.9) is upper-bounded by

$$\frac{2t!}{(\varepsilon|S|/2^m)^{2t}} \cdot \sum_{j=1}^{t} \frac{(|S|/2^m)^j}{j!} \; < \; 2 \cdot \frac{2t!/t!}{(\varepsilon^2|S|/2^m)^t} \; < \; \left(\frac{2t \cdot 2^m}{\varepsilon^2|S|}\right)^t$$

where the first inequality assumes $|S| > n2^m$ (which is justified by the fact that the claim hold vacuously otherwise). This upper-bounds the probability that a random $h \in H_n^m$ violates Eq. (D.7) with respect to a fixed y. Using a union bound on all $y \in \{0,1\}^m$, the lemma follows. ■

D.3 Sampling

In many settings repeated sampling is used to estimate the average (or other statistics) of a huge set of values.[3] Namely, given a "value" function $\nu : \{0,1\}^n \to \mathbb{R}$, one wishes to approximate $\bar{\nu} \stackrel{\text{def}}{=} \frac{1}{2^n}\sum_{x\in\{0,1\}^n} \nu(x)$ without having to inspect the value of ν at each point of the domain. The obvious thing to do is sampling the domain at random, and obtaining an approximation to $\bar{\nu}$ by taking the average of the values of ν on the sample points. It turns out that certain "pseudorandom" sequences of sample points may serve almost as well as truly random sequences of sample points, and thus the foregoing problem is indeed related to Section 8.5.

D.3.1 Formal Setting

It is essential to have the range of the function ν be bounded (since otherwise no reasonable approximation is possible). For simplicity, we adopt the convention of having $[0,1]$ be the range of ν, and the problem for other (predetermined) ranges can be treated analogously. Our notion of approximation depends on two parameters: accuracy (denoted ε) and error probability (denoted δ). We wish to have an algorithm that, with probability at least $1-\delta$, gets within ε of the correct value. This leads to the following definition.

Definition D.7 (sampler): *A* sampler *is a randomized oracle machine that on input parameters n* (length), ε (accuracy) *and* δ (error), *and oracle access to* any *function $\nu : \{0,1\}^n \to [0,1]$, outputs, with probability at least $1-\delta$, a value that is at most ε away from $\bar{\nu} \stackrel{\text{def}}{=} \frac{1}{2^n}\sum_{x\in\{0,1\}^n} \nu(x)$. Namely,*

$$\mathsf{Pr}[|\text{sampler}^{\nu}(n,\varepsilon,\delta) - \bar{\nu}| > \varepsilon] \; < \; \delta$$

where the probability is taken over the internal coin tosses of the sampler.
A non-adaptive sampler *is a sampler that consists of two deterministic algorithms: a* sample generating *algorithm, G, and a* evaluation algorithm*, V. On input n, ε, δ*

[3]Indeed, this problem was already mentioned in §D.1.2.4.

and a random seed *of adequate length, algorithm G generates a sequence of queries, denoted $s_1, ..., s_m \in \{0,1\}^n$. Algorithm V is given the corresponding sequence of ν-values (i.e., $\nu(s_1), ..., \nu(s_m)$) and outputs an estimate to $\bar{\nu}$.*

We are interested in "the complexity of sampling" quantified as a function of the parameters n, ε and δ. Specifically, we will consider three complexity measures: The sample complexity (i.e., the number of oracle queries made by the sampler); the randomness complexity (i.e., the length of the random seed used by the sampler); and the computational complexity (i.e., the running-time of the sampler). We say that a sampler is efficient if its running-time is polynomial in the total length of its queries (i.e., polynomial in both its sample complexity and in n). We will focus on efficient samplers. Furthermore, we will be most interested in efficient samplers that have optimal (up-to a constant factor) sample complexity, and will seek to minimize the randomness complexity of such samplers. Note that minimizing the randomness complexity without referring to the sample complexity makes no sense.

D.3.2 Known Results

We note that all the following positive results refer to non-adaptive samplers, whereas the lower bound hold also for general samplers. For more details on these results, see [90, Sec. 3.6.4] and the references therein.

The naive sampler. The straightforward method (a.k.a the naive sampler) consists of *uniformly and independently* selecting sufficiently many sample points (queries), and outputting the average value of the function on these points. Using Chernoff Bound it follows that $O(\frac{\log(1/\delta)}{\varepsilon^2})$ sample points suffice. As indicated next, the naive sampler is optimal (up-to a constant factor) in its sample complexity, but is quite wasteful in randomness.

It is known that $\Omega(\frac{\log(1/\delta)}{\varepsilon^2})$ samples are needed in any sampler, and that any sampler that makes $s(n,\varepsilon,\delta)$ queries must have randomness complexity at least $n + \log_2(1/\delta) - \log_2 s(n,\varepsilon,\delta) - O(1)$. These lower bounds are tight (as demonstrated by non-explicit and inefficient samplers). The foregoing facts guide our quest for improvements, which is aimed at finding more randomness-efficient ways of *efficiently* generating sample sequences that can be used in conjunction with an appropriate evaluation algorithm V. (We stress that V need not necessarily take the average of the values of the sampled points.)

The pairwise-independent sampler. Using a pairwise-independence generator (cf. §8.5.1.1) for generating sample points, along with the natural evaluation algorithm (which outputs the average of the values of these points), we can obtain a great saving in the randomness complexity: In particular, using a seed of length $2n$, we can generate $O(1/\delta\varepsilon^2)$ pairwise-independent sample points, which (by Eq. (D.4)) suffice for getting accuracy ε with error δ. Thus, this (Pairwise-Independent) sampler uses $2n$ coin tosses rather than the $\Omega((\log(1/\delta))\varepsilon^{-2} \cdot n)$ coin tosses used by the naive sampler. Furthermore, for constant $\delta > 0$, the Pairwise-Independent Sampler is optimal up-to a constant factor in both its sample and randomness complexities. However, for small δ (i.e., $\delta = o(1)$), this sampler is wasteful in sample complexity.

The Median-of-Averages sampler. A new idea is required for going further, and a relevant tool – random walks on expander graphs (see Sections 8.5.3 and E.2) – is needed too. Specifically, we combine the Pairwise-Independent Sampler with the Expander Random Walk Generator (of Proposition 8.29) to obtain a new sampler. The new sampler uses a t-long random walk on an expander with vertex set $\{0,1\}^{2n}$ for *generating a sequence of* $t \stackrel{\text{def}}{=} O(\log(1/\delta))$ *related seeds for* t *invocations of the Pairwise-Independent Sampler*, where each of these invocations uses the corresponding $2n$ bits to generate a sequence of $O(1/\varepsilon^2)$ samples in $\{0,1\}^n$. The new sampler, called the Median-of-Averages Sampler, outputs the median of the t values obtained in these t invocation of the Pairwise-Independent Sampler. In analyzing this sampler, we first note that each of the foregoing t invocations returns a value that, with probability at least 0.9, is ε-close to $\bar{\nu}$. By Theorem 8.28 (see also Exercise 8.44), with probability at least $1-\exp(-t) = 1-\delta$, *most* of these t invocations return an ε-close approximation. Hence, *the median among these* t *values is an* (ε, δ)*-approximation to the correct value.* The resulting sampler has sample complexity $O(\frac{\log(1/\delta)}{\varepsilon^2})$ and randomness complexity $2n+O(\log(1/\delta))$, which is optimal up-to a constant factor in both complexities.

Further improvements. The randomness complexity of the Median-of-Averages Sampler can be decreased from $2n + O(\log(1/\delta))$ to $n + O(\log(1/\delta\varepsilon))$, while maintaining its (optimal) sample complexity (of $O(\frac{\log(1/\delta)}{\varepsilon^2})$). This is done by replacing the Pairwise Independent Sampler by a sampler that picks a random vertex in a suitable expander, samples all its neighbors, and outputs the average value seen.

Averaging Samplers. Averaging (a.k.a. "Oblivious") samplers are non-adaptive samplers in which the evaluation algorithm is the natural one: that is, it merely outputs the average of the values of the sampled points. Indeed, the Pairwise-Independent Sampler is an averaging sampler, whereas the Median-of-Averages Sampler is not. Interestingly, averaging samplers have applications for which ordinary non-adaptive samplers do not suffice. Averaging samplers are closely related to randomness extractors, defined and discussed in the subsequent Section D.4.

An odd perspective. Recall that a non-adaptive sampler consists of a sample generator G and an evaluator V such that for every $\nu:\{0,1\}^n \to [0,1]$ it holds that

$$\mathsf{Pr}_{(s_1,...,s_m)\leftarrow G(U_k)}[|V(\nu(s_1),...,\nu(s_m)) - \bar{\nu}| > \varepsilon] \ < \ \delta, \qquad \text{(D.10)}$$

where k denotes the length of the sampler's (random) seed. Thus, we may view G as a pseudorandom generator that is subjected to a class of distinguishers that is determined by a fixed algorithm V and an arbitrary function $\nu : \{0,1\}^n \to [0,1]$. Specifically, assuming that V works well when the m samples are distributed uniformly and independently (i.e., $\mathsf{Pr}[|V(\nu(U_n^{(1)}),...,\nu(U_n^{(m)})) - \bar{\nu}| > \varepsilon] \ < \ \delta$), we require G to generate sequences that satisfy the corresponding condition (as stated in Eq. (D.10)). What is a bit odd about the foregoing perspective is that, except for the case of averaging samplers, the class of distinguishers considered here is effected by a component (i.e., the evaluator V) that is potentially custom-made to help the generator G fool the distinguisher.[4]

D.3.3 Hitters

Hitters may be viewed as a relaxation of samplers. Specifically, considering only Boolean functions, hitters are required to generate a sample that contains a point evaluating to 1 whenever at least an ε fraction of the function values equal 1. That is, a hitter is a randomized algorithm that on input parameters n (length), ε (accuracy) and δ (error), outputs a list of n-bit strings such that, for every set $S \subseteq \{0,1\}^n$ of density greater than ε, with probability at least $1-\delta$, the list contains at least one element of S. Note the correspondence to the (ε, δ)-hitting problem defined in Section 8.5.3.

Needless to say, any sampler yields a hitter (with respect to essentially the same parameters n, ε and δ).[5] However, hitting is strictly easier than evaluating the density of the target set: $O(1/\varepsilon)$ (pairwise independent) random samples suffice to hit any set of density ε with constant probability, whereas $\Omega(1/\varepsilon^2)$ samples are needed for approximating the average value of a Boolean function up to accuracy ε (with constant error probability). Indeed, adequate simplifications of the samplers discussed in Appendix D.3.2 yield hitters with sample complexity proportional to $1/\varepsilon$ (rather than to $1/\varepsilon^2$).

D.4 Randomness Extractors

Extracting almost-perfect randomness from sources of weak (i.e., defected) randomness is crucial for the actual use of randomized algorithms, procedures and protocols. The latter are analyzed assuming that they are given access to a perfect random source, while in reality one typically has access only to sources of weak (i.e., highly imperfect) randomness. This gap is bridged by using randomness extractors, which are efficient procedures that (possibly with the help of little extra randomness) convert any source of weak randomness into an almost-perfect random source. Thus, randomness extractors are devices that greatly enhance the quality

[4]Another aspect in which samplers differ from the various pseudorandom generators discussed in Chapter 8 is in the aim to minimize, rather than maximize, the number of "blocks" (denoted here by m) in the output sequence. However, also in the case of samplers the aim is to maximize the block-length (denoted here by n).

[5]Specifically, any sampler with respect to the parameters n, ε and δ, yields a hitter with respect to the parameters n, 2ε and δ. (The need for slackness is easily demonstrated by noting that estimating the average with accuracy $\varepsilon = 1/2$ is trivial, whereas hitting is non-trivial for any accuracy (density) $\varepsilon < 1$.) The claim is obvious for non-adaptive samplers, but actually holds also for adaptive samplers. Note that adaptivity does not provide any advantage in the context of hitters, because one may assume (without loss of generality) that all prior samples missed the target set S.

of random sources. In addition, randomness extractors are related to several other fundamental problems, to be further discussed later.

One key parameter, which was avoided in the foregoing discussion, is the class of weak random sources from which we need to extract almost perfect randomness. Needless to say, it is preferable to make as little assumptions as possible regarding the weak random source. In other words, we wish to consider a wide class of such sources, and require that the randomness extractor (often referred to as the extractor) "works well" for any source in this class. A general class of such sources is defined in §D.4.1.1, but first we wish to mention that even for very restricted classes of sources no deterministic extractor can work.[6] To overcome this impossibility result, two approaches are used:

Seeded extractors: The first approach consists of considering randomized extractors that use a relatively small amount of randomness (in addition to the weak random source). That is, these extractors obtain two inputs: a short truly random seed and a relatively long sequence generated by an arbitrary source that belongs to the specified class of sources. This suggestion is motivated in two different ways:

1. The application may actually have access to an almost-perfect random source, but bits from this high-quality source are much more expensive than bits from the weak (i.e., low-quality) random source. Thus, it makes sense to obtain few high-quality bits from the almost-perfect source and use them to "purify" the cheap bits obtained from the weak (low-quality) source. Thus, combining many cheap (but low-quality) bits with few high-quality (but expensive) bits, we obtain many high-quality bits.
2. In some applications (e.g., when using randomized algorithms), it may be possible to invoke the application multiple times, and use the "typical" outcome of these invocations (e.g., rule by majority in the case of a decision procedure). For such applications, we may proceed as follows: first we obtain an outcome r of the weak random source, then we invoke the application multiple times such that for every possible seed s we invoke the application feeding it with $\texttt{extract}(s, r)$, and finally we use the "typical" outcome of these invocations. Indeed, this is analogous to the context of derandomization (see Section 8.3), and likewise this alternative is typically not applicable to cryptographic and/or distributed settings.

Few independent sources: The second approach consists of considering deterministic extractors that obtain samples from a few (say two) *independent* sources of weak randomness. Such extractors are applicable in any setting (including in cryptography), provided that the application has access to the required number of independent weak random sources.

[6]For example, consider the class of sources that output n-bit strings such that no string occurs with probability greater than $2^{-(n-1)}$ (i.e., twice its probability weight under the uniform distribution).

In this section we focus on the first type of extractors (i.e., the *seeded extractors*). This choice is motivated both by the relatively more mature state of the research of seeded extractors and by the closer connection between seeded extractors and other topics in complexity theory.

D.4.1 Definitions and various perspectives

We first present a definition that corresponds to the foregoing motivational discussion, and later discuss its relation to other topics in complexity.

D.4.1.1 The Main Definition

A very wide class of weak random sources corresponds to sources in which no specific output is too probable. That is, the class is parameterized by a (probability) bound β and consists of all sources X such that for every x it holds that $\mathsf{Pr}[X = x] \leq \beta$. In such a case, we say that X has min-entropy[7] at least $\log_2(1/\beta)$. Indeed, we represent sources as random variables, and assume that they are distributed over strings of a fixed length, denoted n. An (n,k)-source is a source that is distributed over $\{0,1\}^n$ and has min-entropy at least k.

An interesting special case of (n,k)-sources is that of sources that are uniform over some subset of 2^k strings. Such sources are called (n,k)-flat. A useful observation is that *each (n,k)-source is a convex combination of (n,k)-flat sources.*

Definition D.8 (extractor for (n,k)-sources):

1. *An algorithm* $\mathrm{Ext}:\{0,1\}^d\times\{0,1\}^n\rightarrow\{0,1\}^m$ *is called an* extractor with error ε for the class $\mathcal{C}$ *if for every source X in $\mathcal{C}$ it holds that* $\mathrm{Ext}(U_d,X)$ *is ε-close to U_m. If $\mathcal{C}$ is the class of (n,k)-sources then* Ext *is called a* (k,ε)-extractor.

2. *An algorithm* Ext *is called a* strong extractor with error ε for $\mathcal{C}$ *if for every source X in $\mathcal{C}$ it holds that $(U_d, \mathrm{Ext}(U_d,X))$ is ε-close to (U_d,U_m). A* strong (k,ε)-extractor *is defined analogously.*

Using the aforementioned "decomposition" of (n,k)-sources into (n,k)-flat sources, it follows that Ext *is a (k,ε)-extractor if and only if it is an extractor with error ε for the class of (n,k)-flat sources.* (A similar claim holds for strong extractors.) Thus, much of the technical analysis is conducted with respect to the class of (n,k)-flat sources. For example, by analyzing the case of (n,k)-flat sources it is easy to see that, for $d = \log_2(n/\varepsilon^2) + O(1)$, there exists a (k,ε)-extractor Ext : $\{0,1\}^d \times \{0,1\}^n \rightarrow \{0,1\}^k$. (The proof employs the Probabilistic Method and uses a union bound on the (finite) set of all (n,k)-flat sources.)[8]

[7]Recall that the entropy of a random variable X is defined as $\sum_x \mathsf{Pr}[X = x]\cdot\log_2(1/\mathsf{Pr}[X = x])$. Indeed the min-entropy of X equals $\min_x\{\log_2(1/\mathsf{Pr}[X = x])\}$, and is always upper-bounded by its entropy.

[8]Indeed, the key fact is that the number of (n,k)-flat sources is $N \stackrel{\rm def}{=} \binom{2^n}{2^k}$. The probability that a random function Ext : $\{0,1\}^d \times \{0,1\}^n \rightarrow \{0,1\}^k$ is not an extractor with error ε for a

We seek, however, explicit extractors; that is, extractors that are implementable by polynomial-time algorithms. We note that the evaluation algorithm of any family of pairwise independent hash functions mapping n-bit strings to m-bit strings constitutes a (strong) (k, ε)-extractor for $\varepsilon = 2^{-\Omega(k-m)}$ (see Theorem D.5). However, these extractors necessarily use a long seed (i.e., $d \geq 2m$ must hold (and in fact $d = n + 2m - 1$ holds in Construction D.3)). In Section D.4.2 we survey constructions of efficient (k, ε)-extractors that obtain logarithmic seed length (i.e., $d = O(\log(n/\varepsilon))$). But before doing so, we provide a few alternative perspectives on extractors.

An important note on logarithmic seed length. The case of logarithmic seed length (i.e., $d = O(\log(n/\varepsilon))$) is of particular importance for a variety of reasons. Firstly, when emulating a randomized algorithm using a defected random source (as in Item 2 of the motivational discussion of seeded extractors), the overhead is exponential in the length of the seed. Thus, the emulation of a generic probabilistic polynomial-time algorithm can be done in polynomial time only if the seed length is logarithmic. Similarly, the applications discussed in §D.4.1.2 and §D.4.1.3 are feasible only if the seed length is logarithmic. Lastly, we note that logarithmic seed length is an absolute lower-bound for (k, ε)-extractors, whenever $k < n - n^{\Omega(1)}$ (and the extractor is non-trivial (i.e., $m \geq 1$ and $\varepsilon < 1/2$)).

D.4.1.2 Extractors as averaging samplers

There is a close relationship between extractors and averaging samplers (which are defined towards the end of Section D.3.2). We shall first show that any averaging sampler gives rise to an extractor. Let $G : \{0,1\}^n \to (\{0,1\}^m)^t$ be the sample generating algorithm of an averaging sampler having accuracy ε and error probability δ. That is, G uses n bits of randomness and generates t sample points in $\{0,1\}^m$ such that, for every $f : \{0,1\}^m \to [0,1]$ with probability at least $1 - \delta$, the average of the f-values of these t pseudorandom points resides in the interval $[\overline{f} \pm \varepsilon]$, where $\overline{f} \overset{\text{def}}{=} \mathsf{E}[f(U_m)]$. Define $\text{Ext} : [t] \times \{0,1\}^n \to \{0,1\}^m$ such that $\text{Ext}(i, r)$ is the i^{th} sample generated by $G(r)$. We shall prove that Ext is a $(k, 2\varepsilon)$-extractor, for $k = n - \log_2(\varepsilon/\delta)$.

Suppose towards the contradiction that there exists a (n, k)-flat source X such that for some $S \subset \{0,1\}^m$ it is the case that $\mathsf{Pr}[\text{Ext}(U_d, X) \in S] > \mathsf{Pr}[U_m \in S] + 2\varepsilon$, where $d = \log_2 t$ and $[t] \equiv \{0,1\}^d$. Define

$$B = \{x \in \{0,1\}^n : \mathsf{Pr}[\text{Ext}(U_d, x) \in S] > (|S|/2^m) + \varepsilon\}.$$

Then, $|B| > \varepsilon \cdot 2^k = \delta \cdot 2^n$. Defining $f(z) = 1$ if $z \in S$ and $f(z) = 0$ otherwise, we have $\overline{f} \overset{\text{def}}{=} \mathsf{E}[f(U_m)] = |S|/2^m$. But, for every $r \in B$ the f-average of the sample

fixed (n, k)-flat source is upper-bounded by $p \overset{\text{def}}{=} 2^{2^k} \cdot \exp(-\Omega(2^{d+k}\varepsilon^2))$, because p bounds the probability that when selecting 2^{d+k} random k-bit long strings there exists a set $T \subset \{0,1\}^k$ that is hit by more than $((|T|/2^k) + \varepsilon) \cdot 2^{d+k}$ of these strings. Note that for $d = \log_2(n/\varepsilon^2) + O(1)$ it holds that $N \cdot p \ll 1$. In fact, the same analysis applies to the extraction of $m = k + \log_2 n$ bits (rather than k bits).

$G(r)$ is greater than $\overline{f}+\varepsilon$, in contradiction to the hypothesis that the sampler has error probability δ (with respect to accuracy ε).

We now turn to show that extractors give rise to averaging samplers. Let $\mathrm{Ext} : \{0,1\}^d \times \{0,1\}^n \to \{0,1\}^m$ be a (k,ε)-extractor. Consider the sample generation algorithm $G : \{0,1\}^n \to (\{0,1\}^m)^{2^d}$ define by $G(r) = (\mathrm{Ext}(s,r))_{s\in\{0,1\}^d}$. We prove that G corresponds to an averaging sampler with accuracy ε and error probability $\delta = 2^{-(n-k-1)}$.

Suppose towards the contradiction that there exists a function $f : \{0,1\}^m \to [0,1]$ such that for $\delta 2^n = 2^{k+1}$ strings $r \in \{0,1\}^n$ the average f-value of the sample $G(r)$ deviates from $\overline{f} \stackrel{\rm def}{=} \mathsf{E}[f(U_m)]$ by more than ε. Suppose, without loss of generality, that for at least half of these r's the average is greater than $\overline{f}+\varepsilon$, and let B denote the set of these r's. Then, for X that is uniformly distributed on B and is thus a (n,k)-source, we have

$$\mathsf{E}[f(\mathrm{Ext}(U_d,X))] > \mathsf{E}[f(U_m)] + \varepsilon,$$

which (using $|f(z)| \leq 1$ for every z) contradicts the hypothesis that $\mathrm{Ext}(U_d,X)$ is ε-close to U_m.

D.4.1.3 Extractors as randomness-efficient error-reductions

As may be clear from the foregoing discussion, extractors yield randomness-efficient methods for error-reduction. This is the case because *error-reduction is a special case of the sampling problem*, obtained by considering Boolean functions. Specifically, for a two-sided error decision procedure A, consider the function $f_x : \{0,1\}^{\rho(|x|)} \to \{0,1\}$ such that $f_x(r) = 1$ if $A(x,r) = 1$ and $f_x(r) = 0$ otherwise. Assuming that the probability that A is correct is at least $0.5+\varepsilon$ (say $\varepsilon = 1/6$), error reduction amounts to providing a sampler with accuracy ε and any desired error probability $\delta \ll \varepsilon$ for the Boolean function f_x. Thus, by §D.4.1.2, any (k,ε)-extractor $\mathrm{Ext} : \{0,1\}^d \times \{0,1\}^n \to \{0,1\}^{\rho(|x|)}$ with $k = n - \log(1/\delta) - 1$ yields the desired error-reduction, provided that 2^d is feasible (e.g., $2^d = \mathrm{poly}(\rho(|x|))$, where $\rho(\cdot)$ represents the randomness complexity of the original algorithm A). The question of interest here is how does n (which represents the randomness complexity of the corresponding sampler) grow as a function of $\rho(|x|)$ and δ.

Error-reduction using the extractor $\mathrm{Ext}:[\mathrm{poly}(\rho(|x|))]\times\{0,1\}^n \to \{0,1\}^{\rho(|x|)}$

	error probability	randomness complexity
original algorithm	$1/3$	$\rho(\lvert x\rvert)$
resulting algorithm	δ (may depend on $\lvert x\rvert$)	n (function of $\rho(\lvert x\rvert)$ and δ)

Needless to say, the answer to the foregoing question depends on the quality of the extractor that we use. In particular, using Part 1 of the forthcoming Theorem D.10, we note that for every $\alpha > 1$, one can obtain $n = O(\rho(|x|)) + \alpha \log_2(1/\delta)$, for any $\delta > 2^{-\mathrm{poly}(\rho(|x|))}$. Note that, for $\delta < 2^{-O(\rho(|x|))}$, this bound on the randomness-complexity of error-reduction is better than the bound of $n = \rho(|x|) + O(\log(1/\delta))$ that is provided (for the reduction of one-sided error) by the Expander Random

Walk Generator (of Section 8.5.3), albeit the number of samples here is larger (i.e., $\text{poly}(\rho(|x|)/\delta)$ rather than $O(\log(1/\delta))$).

Mentioning the reduction of *one-sided* error-probability brings us to a corresponding relaxation of the notion of an extractor, which is called a disperser. Loosely speaking, a (k,ε)-disperser is only required to hit (with positive probability) any set of density greater than ε in its image, rather than produce a distribution that is ε-close to uniform.

Definition D.9 (dispersers): *An algorithm* $\text{Dsp} : \{0,1\}^d \times \{0,1\}^n \to \{0,1\}^m$ *is called a* (k,ε)-disperser *if for every* (n,k)*-source* X *the support of* $\text{Dsp}(U_d, X)$ *covers at least* $(1-\varepsilon)\cdot 2^m$ *points. Alternatively, for every set* $S \subset \{0,1\}^m$ *of size greater than* $\varepsilon 2^m$ *it holds that* $\Pr[\text{Dsp}(U_d, X) \in S] > 0$.

Dispersers can be used for the reduction of one-sided error analogously to the use of extractors for the reduction of two-sided error. Specifically, regarding the aforementioned function f_x (and assuming that $\Pr[f_x(U_{\ell(|x|)})=1] > \varepsilon$), we may use any (k,ε)-disperser $\text{Dsp} : \{0,1\}^d \times \{0,1\}^n \to \{0,1\}^{\ell(|x|)}$ towards finding a point z such that $f_x(z) = 1$. Indeed, if $\Pr[f_x(U_{\ell(|x|)})=1] > \varepsilon$ then there are less than 2^k points z such that $(\forall s\in\{0,1\}^d)\, f_x(\text{Dsp}(s,z)) = 0$, and thus the one-sided error can be reduced from $1-\varepsilon$ to $2^{-(n-k)}$ while using n random bits. (Note that dispersers are closely related to hitters (cf. Appendix D.3.3), analogously to the relation of extractors and averaging samplers.)

D.4.1.4 Other perspectives

Extractors and dispersers have an appealing interpretation in terms of bipartite graphs. Starting with dispersers, we view any (k,ε)-disperser $\text{Dsp} : \{0,1\}^d \times \{0,1\}^n \to \{0,1\}^m$ as a bipartite graph $G = ((\{0,1\}^n, \{0,1\}^m), E)$ such that $E = \{(x, \text{Dsp}(s,x)) : x \in \{0,1\}^n, s \in \{0,1\}^d\}$. This graph has the property that *any* subset of 2^k vertices on the left (i.e., in $\{0,1\}^n$) has a neighborhood that contains at least a $1-\varepsilon$ fraction of the vertices of the right, which is remarkable in the typical case where d is small (e.g., $d = O(\log n/\varepsilon)$) and $n \gg k \geq m$ whereas $m = \Omega(k)$ (or at least $m = k^{\Omega(1)}$). Furthermore, if Dsp is efficiently computable then this bipartite graph is strongly constructible in the sense that, given a vertex on the left, one can efficiently find each of its neighbors. Any (k,ε)-extractor $\text{Ext} : \{0,1\}^d \times \{0,1\}^n \to \{0,1\}^m$ yields an analogous graph with an even stronger property: the neighborhood multi-set of *any* subset of 2^k vertices on the left covers the vertices on the right in an almost uniform manner.

An odd perspective. In addition to viewing extractors as averaging samplers, which in turn may be viewed within the scope of the pseudorandomness paradigm, we mention here an even more odd perspective. Specifically, randomness extractors may be viewed as randomized algorithms (distinguishers) designed on purpose such that to be fooled by any weak random source (but not by an even worse source). Specifically, for any (k,ε)-extractor $\text{Ext} : \{0,1\}^d \times \{0,1\}^n \to \{0,1\}^m$, where $\varepsilon \leq 1/100$, $m = k = \omega(\log n/\varepsilon)$ and $d = O(\log n/\varepsilon)$, consider the following class of distinguishers (or tests), parameterized by subsets of $\{0,1\}^m$: for $S \subset \{0,1\}^m$, the test T_S satisfies $\mathsf{Pr}[T_S(x)=1] = \mathsf{Pr}[\mathrm{Ext}(U_d, x) \in S]$ (i.e., on input $x \in \{0,1\}^n$, the test uniformly selects $s \in \{0,1\}^d$ and outputs 1 if and only if $\mathrm{Ext}(s,x) \in S$). Then, as shown next, any (n,k)-source is "pseudorandom" with respect to this class of distinguishers, but sufficiently "non-(n,k)-sources" are not "pseudorandom" with respect to this class of distinguishers.

1. For every (n,k)-source X and every $S \subset \{0,1\}^m$, the test T_S does not distinguish X from U_n (i.e., $\mathsf{Pr}[T_S(X) = 1] = \mathsf{Pr}[T_S(U_n) = 1] \pm 2\varepsilon$), because $\mathrm{Ext}(U_d, X)$ is 2ε-close to $\mathrm{Ext}(U_d, U_n)$ (since each is ε-close to U_m).
2. On the other hand, for every $(n, k-d-4)$-flat source Y there exists a set S such that T_S distinguish Y from U_n with gap at least 0.9 (e.g., for S that equals the support of $\mathrm{Ext}(U_d, Y)$, it holds that $\mathsf{Pr}[T_S(Y)=1] = 1$ but $\mathsf{Pr}[T_S(U_n)=1] \leq \mathsf{Pr}[U_m \in S] + \varepsilon = 2^{d+(k-d-4)-m} + \varepsilon < 0.1$). Furthermore, any source that has entropy below $(k/4) - d$ will be detected as defected by this class (with probability at least 2/3).[9]

Thus, this weird class of tests deems each (n,k)-source as "pseudorandom" while deeming sources of significantly lower entropy (e.g., entropy lower than $(k/4) - d$) as non-pseudorandom. Indeed, this perspective stretches the pseudorandomness paradigm quite far.

D.4.2 Constructions

Recall that we seek explicit constructions of extractors; that is, functions $\mathrm{Ext} : \{0,1\}^d \times \{0,1\}^n \to \{0,1\}^m$ that can be computed in polynomial-time. The question, of course, is of parameters; that is, having explicit (k,ε)-extractors *with m as large as possible and d as small as possible.* We first note that, except in "pathological" cases[10], both $m \leq k+d-(2\log_2(1/\varepsilon)-O(1))$ and $d \geq \log_2((n-k)/\varepsilon^2)-O(1)$ must hold, regardless of the explicitness requirement. The aforementioned bounds are in fact tight; that is, there exists (non-explicit) (k,ε)-extractors with $m = k + d - 2\log_2(1/\varepsilon) - O(1)$ and $d = \log_2((n-k)/\varepsilon^2) + O(1)$. The obvious goal is meeting these bounds via explicit constructions.

D.4.2.1 Some known results

Despite tremendous progress on this problem (and occasional claims regarding "optimal" explicit constructions), the ultimate goal was not reached yet. Nevertheless, the known explicit constructions are pretty close to being optimal.

Theorem D.10 (explicit constructions of extractors): *Explicit (k,ε)-extractors of the form* $\mathrm{Ext} : \{0,1\}^d \times \{0,1\}^n \to \{0,1\}^m$ *exist for the following cases (i.e., settings of the parameters d and m):*

[9]For any such source Y, the distribution $Z = \mathrm{Ext}(U_d, Y)$ has entropy at most $k/4 = m/4$, and thus is 0.7-far from U_m (and 2/3-far from $\mathrm{Ext}(U_d, U_n)$). The lower-bound on the statistical distance between Z and U_m can be proved by the contra-positive: if Z is δ-close to U_m then its entropy is at least $(1-\delta)\cdot m - 1$ (e.g., by using Fano's inequality, see [63, Thm. 2.11.1]).

[10]That is, for $\varepsilon < 1/2$ and $m > d$.

1. *For $d = O(\log n/\varepsilon)$ and $m = (1-\alpha)\cdot(k-O(d))$, where $\alpha > 0$ is an arbitrarily small constant and provided that $\varepsilon > \exp(-k^{1-\alpha})$.*

2. *For $d = (1+\alpha)\cdot\log_2 n$ and $m = k/\mathrm{poly}(\log n)$, where $\varepsilon, \alpha > 0$ are arbitrarily small constants.*

Proofs of Part 1 and Part 2 can be found in [113] and [201], respectively. We note that, for sake of simplicity, we did not quote the best possible bounds. Furthermore, we did not mention additional incomparable results (which are relevant for different ranges of parameters).

We refrain from providing an overview of the proof of Theorem D.10, but rather review the proof of a weaker result that provides *explicit $(n^\gamma, \mathrm{poly}(1/n))$-extractors for the case of $d = O(\log n)$ and $m = n^{\Omega(\gamma)}$, where $\gamma > 0$ is an arbitrarily small constant.* Indeed, in §D.4.2.2, we review the conceptual insight that underlies this result (as well as much of the subsequent developments in the area).

D.4.2.2 The pseudorandomness connection

We conclude this section with an overview of a fruitful connection between extractors and certain pseudorandom generators. The connection, discovered by Trevisan [222], is surprising in the sense that it goes in a non-standard direction: it transforms certain pseudorandom generators into extractors. As argued throughout this book (most conspicuously at the end of Section 7.1.2), computational objects are typically more complex than the corresponding information theoretical objects. Thus, if pseudorandom generators and extractors are at all related (which was not suspected before [222]) then this relation should not be expected to help in the construction of extractors, which seem an information theoretic object. Nevertheless, the discovery of this relation did yield a breakthrough in the study of extractors.[11]

Teaching note: The current text assumes familiarity with pseudorandom generators and in particular with the Nisan–Wigderson Generator (presented in §8.3.2.1).

But before describing the connection, let us wonder for a moment. Just looking at the syntax, we note that pseudorandom generators have a single input (i.e., the seed), while extractors have two inputs (i.e., the n-bit long source and the d-bit long seed). But taking a second look at the Nisan–Wigderson Generator (i.e., the combination of Construction 8.17 with an amplification of worst-case to average-case hardness), we note that this construction can be viewed as taking two inputs: a d-bit long seed and a "hard" predicate on d'-bit long strings (where $d' = \Omega(d)$).[12] Now, an appealing idea is to use the n-bit long source as a (truth-table) description of a (worse-case) hard predicate (which indeed means setting $n = 2^{d'}$). The key observation is that *even if the source is only weakly random then it is likely to represent a predicate that is hard on the worst-case.*

[11] We note that once the connection became better understood, influence started going in the "right" direction: from extractors to pseudorandom generators.

[12] Indeed, to fit the current context, we have modified some notations. In Construction 8.17 the length of the seed is denoted by k and the length of the input for the predicate is denoted by m.

Recall that the aforementioned construction is supposed to yield a pseudorandom generator whenever it starts with a hard predicate. In the current context, where there are no computational restrictions, pseudorandomness is supposed to hold against any (computationally unbounded) distinguisher, and thus here pseudorandomness means being statistically close to the uniform distribution (on strings of the adequate length, denoted ℓ). Intuitively, this makes sense only if the observed sequence is shorter that the amount of randomness in the source (and seed), which is indeed the case (i.e., $\ell < k + d$, where k denotes the min-entropy of the source). Hence, there is hope to obtain a good extractor this way.

To turn the hope into a reality, we need a proof (which is sketched next). Looking again at the Nisan–Wigderson Generator, we note that the proof of indistinguishability of this generator provides a black-box procedure for computing the underlying predicate when given oracle access to any potential distinguisher. Specifically, in the proofs of Theorems 7.19 and 8.18 (which holds for any $\ell = 2^{\Omega(d')}$)[13], this black-box procedure was implemented by a *relatively small circuit* (which depends on the underlying predicate). Hence, this procedure contains relatively little information (regarding the underlying predicate), on top of the observed ℓ-bit long output of the extractor/generator. Specifically, for some fixed polynomial p, the amount of information encoded in the procedure (and thus available to it) is upper-bound by $b \stackrel{\text{def}}{=} p(\ell)$, while the procedure is suppose to compute the underlying predicate correctly on each input. That is, b bits of information are supposed to fully determine the underlying predicate, which in turn is identical to the n-bit long source. However, if the source has min-entropy exceeding b, then it cannot be fully determine using only b bits of information. It follows that the foregoing construction constitutes a $(b + O(1), 1/6)$-extractor (outputting $\ell = b^{\Omega(1)}$ bits), where the constant $1/6$ is the one used in the proof of Theorem 8.18 (and the argument holds provided that $b = n^{\Omega(1)}$). Note that this extractor uses a seed of length $d = O(d') = O(\log n)$. The argument can be extended to obtain $(k, \text{poly}(1/k))$-extractors that output $k^{\Omega(1)}$ bits using a seed of length $d = O(\log n)$, provided that $k = n^{\Omega(1)}$.

We note that the foregoing description has only referred to two abstract properties of the Nisan–Wigderson Generator: (1) the fact that this generator uses any worst-case hard predicate as a black-box, and (2) the fact that its analysis uses any distinguisher as a black-box. In particular, we viewed the amplification of worst-case hardness to inapproximability (performed in Theorem 7.19) as part of the construction of the pseudorandom generator. An alternative presentation, which is more self-contained, replaces the amplification step of Theorem 7.19 by a direct argument in the current (information theoretic) context and plugs the resulting predicate directly into Construction 8.17. The advantages of this alternative include using a simpler amplification (since amplification is simpler in the information theoretic setting than in the computational setting), and deriving transparent construction and analysis (which mirror Construction 8.17 and Theorem 8.18, respectively).

[13]Recalling that $n = 2^{d'}$, the restriction $\ell = 2^{\Omega(d')}$ implies $\ell = n^{\Omega(1)}$.

The alternative presentation. The foregoing analysis transforms a generic distinguisher into a procedure that computes the underlying predicate correctly on each input, which fully determines this predicate. Hence, an upper-bound on the information available to this procedure yields an upper-bound on the number of possible outcomes of the source that are bad for the extractor. In the alternative presentation, we transforms a generic distinguisher into a procedure that only approximates the underlying predicate; that is, the procedure yields a function that is relatively close to the underlying predicate. If the potential underlying predicates are far apart, then this yields the desired bound (on the number of bad source-outcomes that correspond to such predicates). Thus, the idea is to encode the n-bit long source by an error correcting code of length $n' = \text{poly}(n)$ and relative distance $0.5 - (1/n)^2$, and use the resulting codeword as a truth-table of a predicate for Construction 8.17.[14] Such codes (coupled with efficient encoding algorithms) do exist (see §E.1.2.5), and the benefit in using them is that each n'-bit long string (determined by the information available to the aforementioned approximation procedure) may be $(0.5 - (1/n))$-close to at most $O(n^2)$ codewords[15] (which correspond to potential predicates). Thus, each approximation procedure rules out at most $O(n^2)$ potential predicates (i.e., source outcomes). In summary, *the resulting extractor converts the n-bit input x into a codeword $x' \in \{0,1\}^{n'}$, viewed as a predicate over $\{0,1\}^{d'}$* (where $d' = \log_2 n'$), *and evaluates this predicate at the ℓ projections of the d-bit long seed, where these projections* (to d' bits) *are determined by the corresponding set system* (i.e., the ℓ-long sequence of d'-subsets of $[d]$ that is used in Construction 8.17). The analysis mirrors the proof of Theorem 8.18, and yields a bound of $2^{O(\ell^2)} \cdot O(n^2)$ on the number of bad outcomes for the source, where $O(\ell^2)$ upper-bounds the amount of information encoded in (and available to) the approximation procedure, and $O(n^2)$ upper-bounds the number of source-outcomes that correspond to codewords that are each $(0.5 - (1/n))$-close to any fixed approximation procedure.

D.4.2.3 Recommended reading

The interested reader is referred to a survey of Shaltiel [200]. This survey contains a comprehensive introduction to the area, including an overview of the ideas that underly the various constructions. In particular, the survey describes the approaches used before the discovery of the pseudorandomness connection, the connection itself (and the constructions that arise from it), and the "third generation" of constructions that followed.

The aforementioned survey predates the most recent constructions (of extractors) that extract a constant fraction of the min-entropy using a logarithmically long seed (cf. Part 1 of Theorem D.10). Such constructions were first presented in [159] and improved (using different ideas) in [113]. Indeed, we refer to reader to [113], which provides a self-contained description of the best known extractor (for almost all settings of the relevant parameters).

[14]Indeed, the use of this error correcting code replaces the hardness-amplification step of Theorem 7.19.

[15]See Appendix E.1.4.

Appendix E

Explicit Constructions

It is easier for a camel to go through the eye of a needle, than for a rich man to enter into the kingdom of God.

Matthew, 19:24.

Complexity theory provides a clear definition of the intuitive notion of an explicit construction. Furthermore, it also suggests a hierarchy of different levels of explicitness, referring to the ease of constructing the said object.

The *basic levels of explicitness* are provided by considering the complexity of fully constructing the object (e.g., the time it takes to print the truth-table of a finite function). In this context, explicitness often means outputting a full description of the object in time that is polynomial in the length of that description. *Stronger levels of explicitness* emerge when considering the complexity of answering natural queries regarding the object (e.g., the time it takes to evaluate a fixed function at a given input). In this context, (strong) explicitness often means answering such queries in polynomial-time.

The aforementioned themes are demonstrated in our brief review of explicit constructions of *error correcting codes* and *expander graphs*. These constructions are, in turn, used in various parts of the main text.

Summary: This appendix provides a brief overview of aspects of coding theory and expander graphs that are most relevant to complexity theory. Starting with coding theory, we review several popular constructions of error correcting codes, culminating in the construction of a "good" binary code (i.e., a code that achieves constant relative distance and constant rate). The latter code is obtained by "concatenating" a Reed-Solomon code with a "mildly explicit" construction of a "good" binary code (which is applied to small pieces of information). We also briefly review the notions of locally testable and locally decodable codes, and present a useful "list decoding bound" (i.e., an upper-bound on the number of codewords that are close to any single sequence).

Turning to expander graphs, we review two standard definitions of expansion (representing combinatorial and algebraic perspectives), and two properties of expanders that are related to (single-step and multi-step) random walks on them. We also spell-out two levels of explicitness of graphs, which correspond to the aforementioned notions of basic and strong explicitness. Finally, we review two explicit constructions of expander graphs.

E.1 Error Correcting Codes

In this section we highlight some issues and aspects of coding theory that are most relevant to the current book. The interested reader is referred to [217] for a more comprehensive treatment of the computational aspects of coding theory. Structural aspects of coding theory, which are at the traditional focus of that field, are covered in standard textbook such as [163].

E.1.1 Basic Notions

Loosely speaking, an error correcting code is a mapping of strings to longer strings such that any two different strings are mapped to a corresponding pair of strings that are far apart (and not merely different). Specifically, $C : \{0,1\}^k \to \{0,1\}^n$ is a (binary) code of distance d if for every $x \neq y \in \{0,1\}^k$ it holds that $C(x)$ and $C(y)$ differ on at least d bit positions. Indeed, *the relation between k, n and d is of major concern: typically, the aim is having a large distance* (i.e., large d) *without introducing too much redundancy*[1] (i.e., have n as small as possible with respect to k (and d)).

It will be useful to extend the foregoing definition to sequences over an arbitrary (finite) alphabet Σ, and to use some notations. Specifically, for $x \in \Sigma^m$, we denote the i^{th} symbol of x by x_i (i.e., $x = x_1 \cdots x_m$), and consider codes over Σ (i.e., mappings of Σ-sequences to Σ-sequences). The mapping (code) $C : \Sigma^k \to \Sigma^n$ has distance d if for every $x \neq y \in \Sigma^k$ it holds that $|\{i : C(x)_i \neq C(y)_i\}| \geq d$. The members of $\{C(x) : x \in \Sigma^k\}$ are called codewords (and in some texts this set itself is called a code).

In general, we define a metric, called Hamming distance, over the set of n-long sequences over Σ. The Hamming distance between y and z, where $y, z \in \Sigma^n$, is defined as the number of locations on which they disagree (i.e., $|\{i : y_i \neq z_i\}|$). The Hamming weight of such sequences is defined as the number of non-zero elements (assuming that one element of Σ is viewed as zero). Typically, Σ is associated with an additive group, and in this case the distance between y and z equals the Hamming weight of $w = y - z$, where $w_i = y_i - z_i$ (for every i).

[1]Note that a trivial way of obtaining distance d is to duplicate each symbol d times. This ("repetition") code satisfies $n = d \cdot k$, while we shall seek $n \ll d \cdot k$. Indeed, as we shall see, one can obtain simultaneously $n = O(k)$ and $d = \Omega(k)$.

Asymptotics. We will actually consider infinite families of codes; that is, $\{C_k : \Sigma_k^k \to \Sigma_k^{n(k)}\}_{k\in S}$, where $S \subseteq \mathbb{N}$ (and typically $S = \mathbb{N}$). (N.B., we allow Σ_k to depend on k.) We say that such a family has distance $d : \mathbb{N} \to \mathbb{N}$ if for every $k \in S$ it holds that C_k has distance $d(k)$. Needless to say, both $n = n(k)$ (called the block-length) and $d(k)$ depend on k, and *the aim is having a linear dependence* (i.e., $n(k) = O(k)$ and $d(k) = \Omega(n(k))$). In such a case, one talks of the relative rate of the code (i.e., the constant $k/n(k)$) and its relative distance (i.e., the constant $d(k)/n(k)$). In general, we will often refer to *relative distances* between sequences. For example, for $y, z \in \Sigma^n$, we say that y and z are ε-close (resp., ε-far) if $|\{i : y_i \neq z_i\}| \leq \varepsilon \cdot n$ (resp., $|\{i : y_i \neq z_i\}| \geq \varepsilon \cdot n$).

Explicitness. A mild notion of explicitness refers to constructing the list of all codewords in time that is polynomial in its length (which is exponential in k). A more standard notion of explicitness refers to generating a specific codeword (i.e., producing $C(x)$ when given x), which coincides with the encoding task mentioned next. Stronger notions of explicitness refer to other computational problems concerning codes (e.g., various decoding tasks).

Computational problems. The most basic computational tasks associated with codes are encoding and decoding (under noise). The definition of the encoding task is straightforward (i.e., map $x \in \Sigma_k^k$ to $C_k(x)$), and an efficient algorithm is required to compute each symbol in $C_k(x)$ in poly$(k, \log|\Sigma_k|)$-time.[2] When defining the decoding task we note that "minimum distance decoding" (i.e., given $w \in \Sigma_k^{n(k)}$, find x such that $C_k(x)$ is closest to w (in Hamming distance)) is just one natural possibility. Two related variants, regarding a code of distance d, are:

Unique decoding: Given $w \in \Sigma_k^{n(k)}$ that is at Hamming distance less than $d(k)/2$ from some codeword $C_k(x)$, retrieve the corresponding decoding of $C_k(x)$ (i.e., retrieve x).

Needless to say, this task is well-defined because there cannot be two different codewords that are each at Hamming distance less than $d(k)/2$ from w.

List decoding: Given $w \in \Sigma_k^{n(k)}$ and a parameter d' (which may be greater than $d(k)/2$), output a list of all codewords (or rather their decoding) that are at Hamming distance at most d' from w. (That is, the task is outputting the list of all $x \in \Sigma_k^k$ such that $C_k(x)$ is at distance at most d' from w.)

Typically, one considers the case that $d' < d(k)$. See Section E.1.4 for a discussion of upper-bounds on the number of codewords that are within a certain distance from a generic sequence.

Two additional computational tasks are considered in Section E.1.3.

[2]The foregoing formulation is not the one that is common in coding theory, but it is the most natural one for our applications. On one hand, this formulation is applicable also to codes with super-polynomial block-length. On the other hand, this formulation does not support a discussion of practical algorithms that compute the codeword faster than is possible when computing each of the codeword's bits separately.

Linear codes. Associating Σ_k with some finite field, we call a code $C_k : \Sigma_k^k \to \Sigma_k^{n(k)}$ linear if it satisfies $C_k(x+y) = C_k(x) + C_k(y)$, where x and y (resp., $C_k(x)$ and $C_k(y)$) are viewed as k-dimensional (resp., $n(k)$-dimensional) vectors over Σ_k, and the arithmetic is of the corresponding vector space. A useful property of linear codes is that their distance equals the Hamming weight of the lightest codeword other than $C_k(0^k)$ $(= 0^{n(k)})$; that is, $\min_{x \neq y}\{|\{i : C_k(x)_i \neq C_k(y)_i\}|\}$ equals $\min_{x \neq 0^k}\{|\{i : C_k(x)_i \neq 0\}|\}$. Another useful property of linear codes is that the code is fully specified by a k-by-$n(k)$ matrix, called the generating matrix, that consists of the codewords of some fixed basis of Σ_k^k. That is, the set of all codewords is obtained by taking all $|\Sigma_k|^k$ different linear combination of the rows of the generating matrix.

E.1.2 A Few Popular Codes

Our focus will be on explicitly constructible codes; that is, (families of) codes of the form $\{C_k : \Sigma_k^k \to \Sigma_k^{n(k)}\}_{k \in S}$ that are coupled with efficient encoding and decoding algorithms. But before presenting several such codes, let us consider a non-explicit code (having "good parameters"); that is, the following result asserts the existence of certain codes without pointing to any specific code (let alone an explicit one).

Proposition E.1 (on the distance of random linear codes): *Let $n, d, t : \mathbb{N} \to \mathbb{N}$ be such that, for all sufficiently large k, it holds that*

$$n(k) \ \geq \ \max\left(2d(k),\ \frac{k + t(k)}{1 - H_2(d(k)/n(k))}\right), \tag{E.1}$$

where $H_2(\alpha) \stackrel{\text{def}}{=} \alpha \log_2(1/\alpha) + (1-\alpha)\log_2(1/(1-\alpha))$. Then, for all sufficiently large k, with probability greater than $1 - 2^{-t(k)}$, a random linear transformation of $\{0,1\}^k$ to $\{0,1\}^{n(k)}$ constitutes a code of distance $d(k)$.

Indeed, for asserting that most random linear codes are good it suffices to set $t = 1$, while for merely asserting the existence of a good linear code even setting $t = 0$ will do. Also, *for every constant $\delta \in (0, 0.5)$ there exists a constant $\rho > 0$ and an infinite family of codes $\{C_k : \{0,1\}^k \to \{0,1\}^{k/\rho}\}_{k \in \mathbb{N}}$ of relative distance δ.* Specifically, any constant $\rho \geq (1 - H_2(\delta))$ will do.

Proof: We consider a uniformly selected k-by-$n(k)$ generating matrix over GF(2), and upper-bound the probability that it yields a linear code of distance less than $d(k)$. We use a union bound on all possible $2^k - 1$ linear combinations of the rows of the generating matrix, where for each such combination we compute the probability that it yields a codeword of Hamming weight less than $d(k)$. Observe that the result of each such linear combination is uniformly distributed over $\{0,1\}^{n(k)}$, and thus this codeword has Hamming weight less than $d(k)$ with probability $p \stackrel{\text{def}}{=} \sum_{i=0}^{d(k)-1} \binom{n(k)}{i} \cdot 2^{-n(k)}$. Clearly, for $d(k) \leq n(k)/2$, it holds that $p < d(k) \cdot 2^{-(1-H_2(d(k)/n(k)))\cdot n(k))}$, but actually $p \leq 2^{-(1-H_2(d(k)/n(k)))\cdot n(k))}$ holds as well (e.g., use [11, Cor. 14.6.3]). Using $(1 - H_2(d(k)/n(k))) \cdot n(k) \geq k + t(k)$, the proposition follows. ■

E.1.2.1 A mildly explicit version of Proposition E.1

Note that Proposition E.1 yields a deterministic algorithm that finds a linear code of distance $d(k)$ by conducting an exhaustive search over all possible generating matrices; that is, a good code can be found in time $\exp(k \cdot n(k))$. The time bound can be improved to $\exp(k + n(k))$, by constructing the generating matrix in iterations such that, at each iteration, the current set of rows is augmented with a single row while maintaining the natural invariance (i.e., all non-empty linear combinations of the current rows have weight at least $d(k)$). Thus, at each iteration, we conduct an exhaustive search over all possible values of the next ($n(k)$-bit long) row, and for each such candidate value we check whether the foregoing invariance holds (by considering all linear combinations of the previous rows and the current candidate).

Note that the proof of Proposition E.1 can be adapted to assert that, as long as we have less than k rows, a random choice of the next row will do with positive probability. Thus, the foregoing iterative algorithm finds a good code in time $\sum_{i=1}^{k} 2^{n(k)} \cdot 2^{i-1} \cdot \mathrm{poly}(n(k)) = \exp(n(k) + k)$. In the case that $n(k) = O(k)$, this yields an algorithm that runs in time that is polynomial in the size of the code (i.e., the number of codewords (i.e., 2^k)). Needless to say, this mild level of explicitness is inadequate for most coding applications; however, it will be useful to us in §E.1.2.5.

E.1.2.2 The Hadamard Code

The Hadamard code is the longest (non-repetitive) *linear* code over $\{0,1\} \equiv \mathrm{GF}(2)$. That is, $x \in \{0,1\}^k$ is mapped to the sequence of all $n(k) = 2^k$ possible linear combinations of its bits; that is, bit locations in the codewords are associated with k-bit strings such that location $\alpha \in \{0,1\}^k$ in the codeword of x holds the value $\sum_{i=1}^{k} \alpha_i x_i$. It can be verified that each non-zero codeword has weight 2^{k-1}, and thus this code has relative distance $d(k)/n(k) = 1/2$ (albeit its block-length $n(k)$ is exponential in k).

Turning to the computational aspects, we note that encoding is very easy. As for decoding, the warm-up discussion at the beginning of the proof of Theorem 7.7 provides a very fast probabilistic algorithm for unique decoding, whereas Theorem 7.8 itself provides a very fast probabilistic algorithm for list decoding.

We mention that the Hadamard code has played a key role in the proof of the PCP Theorem (Theorem 9.16); see §9.3.2.1.

A propos long codes. We mention that the longest (non-repetitive) binary code (called the Long-Code and introduced in [29]) is extensively used in the design of "advanced" PCP systems (see, e.g., [116, 117]). In this code, a k-bit long string x is mapped to the sequence of $n(k) = 2^{2^k}$ values, each corresponding to the evaluation of a different Boolean function at x; that is, bit locations in the codewords are associated with Boolean functions such that the location associated with $f:\{0,1\}^k \to \{0,1\}$ in the codeword of x holds the value $f(x)$.

E.1.2.3 The Reed–Solomon Code

Reed-Solomon codes can be defined for any adequate non-binary alphabet, where the alphabet is associated with a finite field of n elements, denoted $\mathrm{GF}(n)$. For any $k < n$, the code maps univariate polynomials of degree $k-1$ over $\mathrm{GF}(n)$ to their evaluation at all field elements. That is, $p \in \mathrm{GF}(n)^k$ (viewed as such a polynomial), is mapped to the sequence $(p(\alpha_1), ..., p(\alpha_n))$, where $\alpha_1, ..., \alpha_n$ is a canonical enumeration of the elements of $\mathrm{GF}(n)$.[3] This mapping is called a Reed-Solomon code with parameters k and n, and its distance is $n-k+1$ (because any non-zero polynomials of degree $k-1$ evaluates to zero at less than k points). Indeed, this code is linear (over $\mathrm{GF}(n)$), since $p(\alpha)$ is a linear combination of $p_0, ..., p_{k-1}$, where $p(\zeta) = \sum_{i=0}^{k-1} p_i \zeta^i$.

The Reed-Solomon code yields infinite families of codes with constant rate and constant relative distance (e.g., by taking $n(k) = 3k$ and $d(k) = 2k$), but the alphabet size grows with k (or rather with $n(k) > k$). Efficient algorithms for unique decoding and list decoding are known (see [216] and references therein). These computational tasks correspond to the extrapolation of polynomials based on a noisy version of their values at all possible evaluation points.

E.1.2.4 The Reed–Muller Code

Reed-Muller codes generalize Reed-Solomon codes by considering multi-variate polynomials rather than univariate polynomials. Consecutively, the alphabet may be any finite field, and in particular the two-element field $\mathrm{GF}(2)$. Reed-Muller codes (and variants of them) are extensively used in complexity theory; for example, they underly Construction 7.11 and the PCP constructed at the end of §9.3.2.2. The relevant property of these (non-binary) codes is that, under a suitable setting of parameters that satisfies $n(k) = \mathrm{poly}(k)$, they allow super fast "codeword testing" and "self-correction" (see discussion in Section E.1.3).

For any prime power q and parameters m and r, we consider the set, denoted $P_{m,r}$, of all m-variate polynomials of *total degree* at most r over $\mathrm{GF}(q)$. Each polynomial in $P_{m,r}$ is represented by the $k = \log_q |P_{m,r}|$ coefficients of all relevant monomials, where in the case that $r < q$ it holds that $k = \binom{m+r}{m}$. We consider the code $C : \mathrm{GF}(q)^k \to \mathrm{GF}(q)^n$, where $n = q^m$, mapping m-variate polynomials of total degree at most r to their values at all q^m evaluation points. That is, the m-variate polynomial p of total degree at most r is mapped to the sequence of values $(p(\overline{\alpha}_1), ..., p(\overline{\alpha}_n))$, where $\overline{\alpha}_1, ..., \overline{\alpha}_n$ is a canonical enumeration of all the m-tuples of $\mathrm{GF}(q)$. The relative distance of this code is lower-bounded by $(q-r)/q$ (cf., Lemma 6.8).

In typical applications one sets $r = \Theta(m^2 \log m)$ and $q = \mathrm{poly}(r)$, which yields $k > m^m$ and $n = \mathrm{poly}(r)^m = \mathrm{poly}(m^m)$. Thus we have $n(k) = \mathrm{poly}(k)$ but not $n(k) = O(k)$. As we shall see in Section E.1.3, the advantage (in comparison to the Reed-Solomon code) is that codeword testing and self-correction can be performed

[3] Alternatively, we may map $(v_1, ..., v_k) \in \mathrm{GF}(n)^k$ to $(p(\alpha_1), ..., p(\alpha_n))$, where p is the unique univariate polynomial of degree $k-1$ that satisfies $p(\alpha_i) = v_i$ for $i = 1, ..., k$. Note that this modification amounts to a linear transformation of the generating matrix.

at complexity related to $q = \text{poly}(\log n)$. Actually, most complexity applications use a variant in which only m-variate polynomials of *individual degree* $r' = r/m$ are encoded. In this case, an alternative presentation (analogous to the one presented in Footnote 3) is preferred: The information is viewed as a function $f : H^m \to \text{GF}(q)$, where $H \subset \text{GF}(q)$ is of size $r' + 1$, and is encoded by the evaluation at all points in $\text{GF}(q)^m$ of the (unique) m-variate polynomial of individual degree r' that extends the function f (see Construction 7.11).

E.1.2.5 Binary codes of constant relative distance and constant rate

Recall that we seek binary codes of constant relative distance and constant rate. Proposition E.1 asserts that such codes exists, but does not provide an explicit construction. The Hadamard code is explicit but does not have a constant rate (to say the least (since $n(k) = 2^k$)).[4] The Reed-Solomon code has constant relative distance and constant rate but uses a non-binary alphabet (which grows at least linearly with k). Thus, all codes we have reviewed so far fall short of providing an *explicit construction of binary codes of constant relative distance and constant rate.* We achieve the desired construction by using the paradigm of concatenated codes [78], which is of independent interest. (Concatenated codes may be viewed as a simple analogue of the proof composition paradigm presented in §9.3.2.2.)

Intuitively, concatenated codes are obtained by first encoding information, viewed as a sequence over a large alphabet, by some code and next encoding each resulting symbol, which is viewed as a sequence of over a smaller alphabet, by a second code. Formally, consider $\Sigma_1 \equiv \Sigma_2^{k_2}$ and two codes, $C_1 : \Sigma_1^{k_1} \to \Sigma_1^{n_1}$ and $C_2 : \Sigma_2^{k_2} \to \Sigma_2^{n_2}$. Then, the concatenated code of C_1 and C_2, maps $(x_1, ..., x_{k_1}) \in \Sigma_1^{k_1} \equiv \Sigma_2^{k_1 k_2}$ to $(C_2(y_1), ..., C_2(y_{n_1}))$, where $(y_1, ..., y_{n_1}) = C_1(x_1, ..., x_{k_1})$.

Note that the resulting code $C : \Sigma_2^{k_1 k_2} \to \Sigma_2^{n_1 n_2}$ has constant rate and constant relative distance if both C_1 and C_2 have these properties. Encoding in the concatenated code is straightforward. To decode a corrupted codeword of C, we view the input as an n_1-long sequence of blocks, where each block is an n_2-long sequence over Σ_2. Applying the decoder of C_2 to each block, we obtain n_1 sequences (each of length k_2) over Σ_2, and interpret each such sequence as a symbol of Σ_1. Finally, we apply the decoder of C_1 to the resulting n_1-long sequence (over Σ_1), and interpret the resulting k_1-long sequence (over Σ_1) as a $k_1 k_2$-long sequence over Σ_2. The key observation is that *if $w \in \Sigma_2^{n_1 n_2}$ is $\varepsilon_1\varepsilon_2$-close to $C(x_1, ..., x_{k_1}) = (C_2(y_1), ..., C_2(y_{n_1}))$ then at least $(1 - \varepsilon_1) \cdot n_1$ of the blocks of w are ε_2-close to the corresponding $C_2(y_i)$.*[5]

We are going to consider the concatenated code obtained by using the Reed-Solomon Code $C_1 : \text{GF}(n_1)^{k_1} \to \text{GF}(n_1)^{n_1}$ as the large code, setting $k_2 = \log_2 n_1$, and using the mildly explicit version of Proposition E.1 (see also §E.1.2.1) $C_2 : \{0,1\}^{k_2} \to \{0,1\}^{n_2}$ as the small code. We use $n_1 = 3k_1$ and $n_2 = O(k_2)$, and so the

[4]Binary Reed-Muller codes also fail to simultaneously provide constant relative distance and constant rate.

[5]This observation offers unique decoding from a fraction of errors that is the product of the fractions (of error) associated with the two original codes. Stronger statements regarding unique decoding of the concatenated code can be made based on more refined analysis (cf. [78]).

concatenated code is $C : \{0,1\}^k \to \{0,1\}^n$, where $k = k_1 k_2$ and $n = n_1 n_2 = O(k)$. The key observation is that C_2 can be constructed in $\exp(k_2)$-time, whereas here $\exp(k_2) = \mathrm{poly}(k)$. Furthermore, both encoding and decoding with respect to C_2 can be performed in time $\exp(k_2) = \mathrm{poly}(k)$. Thus, we get:

Theorem E.2 (an explicit good code): *There exists constants $\delta, \rho > 0$ and an explicit family of binary codes of rate ρ and relative distance at least δ. That is, there exists a polynomial-time* (encoding) *algorithm C such that $|C(x)| = |x|/\rho$* (for every x) *and a polynomial-time* (decoding) *algorithm D such that for every y that is $\delta/2$-close to some $C(x)$ it holds that $D(y) = x$. Furthermore, C is a linear code.*

The linearity of C is justified by using a Reed-Solomon code over the extension field $F = \mathrm{GF}(2^{k_2})$, and noting that this code induces a linear transformation over $\mathrm{GF}(2)$. Specifically, the value of a polynomial p over F at a point $\alpha \in F$ can be obtained as a linear transformation of the coefficient of p, when viewed as k_2-dimensional vectors over $\mathrm{GF}(2)$.

Relative distance approaching one half. Note that starting with a Reed-Solomon code of relative distance δ_1 and a smaller code C_2 of relative distance δ_2, we obtain a concatenated code of relative distance $\delta_1\delta_2$. Recall that, for any constant $\delta_1 < 1$, there exists a Reed-Solomon code $C_1 : \mathrm{GF}(n_1)^{k_1} \to \mathrm{GF}(n_1)^{n_1}$ of relative distance δ_1 and constant rate (i.e., $1 - \delta_1$). Thus, for any constant $\varepsilon > 0$, we may obtain an explicit code of constant rate and relative distance $(1/2) - \varepsilon$ (e.g., by using $\delta_1 = 1 - (\varepsilon/2)$ and $\delta_2 = (1 - \varepsilon)/2$). Furthermore, giving up on constant rate, we may start with a Reed-Solomon code of block-length $n_1(k_1) = \mathrm{poly}(k_1)$ and distance $n_1(k_1) - k_1$ over $[n_1(k_1)]$, and use a Hadamard code (encoding $[n_1(k_1)] \equiv \{0,1\}^{\log_2 n_1(k_1)}$ by $\{0,1\}^{n_1(k_1)}$) in the role of the small code C_2. This yields a (concatenated) binary code of block length $n(k) = n_1(k)^2 = \mathrm{poly}(k)$ and distance $(n_1(k) - k) \cdot n_1(k)/2$. Thus, *the resulting explicit code has relative distance $\frac{1}{2} - \frac{k}{2\sqrt{n(k)}} = \frac{1}{2} - o(1)$, provided that $n(k) = \omega(k^2)$.*

E.1.3 Two Additional Computational Problems

In this section we briefly review relaxations of two traditional coding theoretic tasks. The purpose of these relaxations is enabling the design of super-fast (randomized) algorithms that provide meaningful information. Specifically, these algorithms may run in sub-linear (e.g., poly-logarithmic) time, and thus cannot possibly solve the unrelaxed version of the corresponding problem.

Local testability. This task refers to testing whether a given word is a codeword (in a predetermine code), based on (randomly) inspecting few locations in the word. Needless to say, we can only hope to make an approximately correct decision; that is, accept each codeword and reject with high probability each word that is *far* from the code. (Indeed, this task is within the framework of property testing; see Section 10.1.2.)

Local decodability. Here the task is to recover a specified bit in the plaintext by (randomly) inspecting few locations in a mildly corrupted codeword. This task is somewhat related to the task of self-correction (i.e., recovering a specified bit in the codeword itself, by inspecting few locations in the mildly corrupted codeword).

Note that the Hadamard code is both locally testable and locally decodable as well as self-correctable (based on a constant number of queries into the word); these facts were demonstrated and extensively used in §9.3.2.1. However, the Hadamard code has an exponential block-length (i.e., $n(k) = 2^k$), and the question is whether one can achieve analogous results with respect to a shorter code (e.g., $n(k) = \text{poly}(k)$). As hinted in §E.1.2.4, the answer is positive (when we refer to performing these operations in time that is poly-logarithmic in k):

Theorem E.3 *For some constant $\delta > 0$ and polynomials $n, q : \mathbb{N} \to \mathbb{N}$, there exists an explicit family of codes $\{C_k : [q(k)]^k \to [q(k)]^{n(k)}\}_{k \in \mathbb{N}}$ of relative distance δ that can be locally testable and locally decodable in* $\text{poly}(\log k)$*-time. That is, the following three conditions hold.*

1. Encoding: *There exists a polynomial time algorithm that on input $x \in [q(k)]^k$ returns $C_k(x)$.*
2. Local Testing: *There exists a probabilistic polynomial-time oracle machine T that given k* (in binary)[6] *and oracle access to $w \in [q(k)]^{n(k)}$* (viewed as $w : [n(k)] \to [q(k)]$) *distinguishes the case that w is a codeword from the case that w is $\delta/2$-far from any codeword. Specifically:*
   (a) *For every $x \in [q(k)]^k$ it holds that $\Pr[T^{C_k(x)}(k) = 1] = 1$.*
   (b) *For every $w \in [q(k)]^{n(k)}$ that is $\delta/2$-far from any codeword of C_k it holds that $\Pr[T^w(k) = 1] \leq 1/2$.*

   As usual, the error probability can be reduced by repetitions.
3. Local Decoding: *There exists a probabilistic polynomial-time oracle machine D that given k and $i \in [k]$* (in binary) *and oracle access to any $w \in [q(k)]^{n(k)}$ that is $\delta/2$-close to $C_k(x)$ returns x_i; that is, $\Pr[D^w(k,i) = x_i] \geq 2/3$.*

   Self correction *holds too: there exists a probabilistic polynomial-time oracle machine M that given k and $i \in [n(k)]$* (in binary) *and oracle access to any $w \in [q(k)]^{n(k)}$ that is $\delta/2$-close to $C_k(x)$ returns $C_k(x)_i$; that is, $\Pr[D^w(k,i) = C_k(x)_i] \geq 2/3$.*

We stress that all these oracle machines work in time that is polynomial in the binary representation of k, which means that they run in time that is poly-logarithmic in k. The code asserted in Theorem E.3 is a (small modification of a) Reed-Muller code, for $r = m^2 \log m < q(k) = \text{poly}(r)$ and $[n(k)] \equiv \text{GF}(q(k))^m$ (see §E.1.2.4).[7]

[6]Thus, the running time of T is $\text{poly}(|k|) = \text{poly}(\log k)$.

[7]The modification is analogous to the one presented in Footnote 3: For a suitable choice of k points $\overline{\alpha}_1, ..., \overline{\alpha}_k \in \text{GF}(q(k))^m$, we map $v_1, ..., v_k$ to $(p(\overline{\alpha}_1), ..., p(\overline{\alpha}_n))$, where p is the unique m-variate polynomial of degree at most r that satisfies $p(\overline{\alpha}_i) = v_i$ for $i = 1, ..., k$.

The aforementioned oracle machines queries the oracle $w : [n(k)] \to \mathrm{GF}(q(k))$ at a non-constant number of locations. Specifically, self-correction for location $i \in \mathrm{GF}(q(k))^m$ is performed by selecting a random line (over $\mathrm{GF}(q(k))^m$) that passes through i, recovering the values assigned by w to all $q(k)$ points on this line, and performing univariate polynomial extrapolation (under mild noise). Local testability is easily reduced to self-correction, and (under the aforementioned modification) local decodability is a special case of self-correction.

Constant number of (binary) queries. The local testing and decoding algorithms asserted in Theorem E.3 make a polylogarithmic number of queries into the oracle. Furthermore, these queries (which refer to a non-binary code) are non-binary (i.e., they are each answered by a non-binary value). In contrast, the Hadamard code has local testing and decoding algorithms that use a *constant number of binary queries*. Can this be obtained with much shorter (binary) codewords? That is, redefining local testability and decodability as requiring a *constant number of queries*, we ask whether binary codes of significantly shorter block-length can be locally testable and decodable. For local testability the answer is definitely positive: one can construct such (locally testable and binary) codes with block-length that is nearly linear (i.e., linear up to polylogarithmic factors; see [36, 67]). For local decodability, the shortest known code has super-polynomial length (see [241]). In light of this state of affairs, we advocate natural relaxations of the local decodability task (e.g., the one studied in [35]).

The interested reader is referred to [93], which includes more details on locally testable and decodable codes as well as a wider perspective. (Note, however, that this survey was written prior to [67] and [241], which resolve two major open problems discussed in [93].)

E.1.4 A List Decoding Bound

A necessary condition for the feasibility of the list decoding task is that the list of codewords that are close to the given word is short. In this section we present an upper-bound on the length of such lists, noting that this bound has found several applications in complexity theory (and specifically to studies related to the contents of this book). In contrast, we do not present far more famous bounds (which typically refer to the relation among the main parameters of codes (i.e., k, n and d)), because they seem less relevant to the contents of this book.

We start with a general statement that refers to any alphabet $\Sigma \equiv [q]$, and later specialize it to the case that $q = 2$. Especially in the general case, it is natural and convenient to consider the agreement (rather than the distance) between sequences over $[q]$. Furthermore, it is natural to focus on agreement rate of at least $1/q$, and it is convenient to state the following result in terms of the "excessive agreement rate" (i.e., the excess beyond $1/q$).[8] Loosely speaking, the following result upper-bounds the number of codewords that have a (sufficient) large agreement rate with

[8]Indeed, we only consider codes with distance $d \leq (1 - 1/q) \cdot n$ (i.e., agreement rate of at least $1/q$) and words that are at distance at most d from the code. Note that a random sequence is expected to agree with any fixed sequence on a $1/q$ fraction of the locations.

any fixed sequence, where the upper-bound depends only on this agreement rate and the agreement rate between codewords (as well as on the alphabet size, but not on k and n).

Lemma E.4 (Part 2 of [105, Thm. 15]): *Let $C : [q]^k \to [q]^n$ be an arbitrary code of distance $d \leq n - (n/q)$, and let $\eta_{\rm C} \stackrel{\rm def}{=} (1-(d/n)) - (1/q) \geq 0$ denote the corresponding upper-bound on the excessive agreement rate between codewords. Suppose that $\eta \in (0,1)$ satisfies*

$$\eta > \sqrt{\left(1-\frac{1}{q}\right)\cdot \eta_{\rm C}} \qquad \text{(E.2)}$$

Then, for any $w \in [q]^n$, the number of codewords that agree with w on at least $((1/q)+\eta)\cdot n$ positions (i.e., are at distance at most $(1-((1/q)+\eta))\cdot n$ from w) *is upper-bounded by*

$$\frac{(1-(1/q))^2 - (1-(1/q))\cdot \eta_{\rm C}}{\eta^2 - (1-(1/q))\cdot \eta_{\rm C}} \qquad \text{(E.3)}$$

In the binary case (i.e., $q=2$), Eq. (E.2) requires $\eta > \sqrt{\eta_{\rm C}/2}$ and Eq. (E.3) yields the upper-bound $(1-2\eta_{\rm C})/(4\eta^2 - 2\eta_{\rm C})$. We highlight two specific cases:

1. At the end of §D.4.2.2, we refer to this bound (for the binary case) while setting $\eta_{\rm C} = (1/k)^2$ and $\eta = 1/k$. Indeed, in this case $(1-2\eta_{\rm C})/(4\eta^2-2\eta_{\rm C}) = O(k^2)$.

2. In the case of the Hadamard code, we have $\eta_{\rm C} = 0$. Thus, for every $w \in \{0,1\}^n$ and every $\eta > 0$, the number of codewords that are $(0.5-\eta)$-close to w is at most $1/4\eta^2$.

In the general case (and specifically for $q \gg 2$) it is useful to simplify Eq. (E.2) by $\eta > \min\{\sqrt{\eta_{\rm C}}, (1/q) + \sqrt{\eta_{\rm C} - (1/q)}\}$ and Eq. (E.3) by $\frac{1}{\eta^2 - \eta_{\rm C}}$.

E.2 Expander Graphs

In this section we review basic facts regarding expander graphs that are most relevant to the current book. For a wider perspective, the interested reader is referred to [124].

Loosely speaking, expander graphs are regular graphs of small degree that exhibit various properties of cliques.[9] In particular, we refer to properties such as the relative sizes of cuts in the graph (i.e., relative to the number of edges), and the rate at which a random walk converges to the uniform distribution (relative to the logarithm of the graph size to the base of its degree).

[9]Another useful intuition is that expander graphs exhibit various properties of random regular graphs of the same degree.

Some technicalities. Typical presentations of expander graphs refer to one of several variants. For example, in some sources, expanders are presented as bipartite graphs, whereas in others they are presented as ordinary graphs (and are in fact very far from being bipartite). We shall follow the latter convention. Furthermore, at times we implicitly consider an augmentation of these graphs where self-loops are added to each vertex. For simplicity, we also allow parallel edges.

We often talk of expander graphs while we actually mean an infinite collection of graphs such that each graph in this collection satisfies the same property (which is informally attributed to the collection). For example, when talking of a d-regular expander (graph) we actually refer to an infinite collection of graphs such that each of these graphs is d-regular. Typically, such a collection (or family) contains a single N-vertex graph for every $N \in \mathbb{S}$, where $\mathbb{S}$ is an infinite subset of $\mathbb{N}$. Throughout this section, we denote such a collection by $\{G_N\}_{N\in\mathbb{S}}$, with the understanding that G_N is a graph with N vertices and $\mathbb{S}$ is an infinite set of natural numbers.

E.2.1 Definitions and Properties

We consider two definitions of expander graphs, two different notions of explicit constructions, and two useful properties of expanders.

E.2.1.1 Two mathematical definitions

We start with two different definitions of expander graphs. These definitions are qualitatively equivalent and even quantitatively related. We start with an algebraic definition, which seems technical in nature but is actually the definition typically used in complexity theoretic applications, since it directly implies various "mixing properties" (see §E.2.1.3). We later present a very natural combinatorial definition (which is the source of the term "expander").

The algebraic definition (eigenvalue gap). Identifying graphs with their adjacency matrix, we consider the eigenvalues (and eigenvectors) of a graph (or rather of its adjacency matrix). Any d-regular graph $G = (V, E)$ has the uniform vector as an eigenvector corresponding to the eigenvalue d, and if G is connected and non-bipartite then the absolute values of all other eigenvalues are strictly smaller than d. The eigenvalue bound, denoted $\lambda(G) < d$, of such a graph G is defined as a tight upper-bound on the *absolute value* of all the other eigenvalues. (In fact, in this case it holds that $\lambda(G) < d - \Omega(1/d|V|^2)$.)[10] The algebraic definition of expanders refers to an infinite family of d-regular graphs and requires the existence of a *constant* eigenvalue bound that holds for all the graphs in the family.

Definition E.5 *An infinite family of d-regular graphs, $\{G_N\}_{N\in\mathbb{S}}$, where $\mathbb{S} \subseteq \mathbb{N}$,* satisfies the eigenvalue bound *β if for every $N \in \mathbb{S}$ it holds that $\lambda(G_N) \leq \beta$. In*

[10]This follows from the connection to the combinatorial definition (see Theorem E.7). Specifically, the square of this graph, denoted G^2, is $|V|^{-1}$-expanding and thus it holds that $\lambda(G)^2 = \lambda(G^2) < d^2 - \Omega(|V|^{-2})$.

*such a case, we say that $\{G_N\}_{N\in\mathbb{S}}$ is a family of (d,β)-*expanders*, and call $d-\beta$ its* eigenvalue gap.

It will be often convenient to consider relative (or normalized) versions of the foregoing quantities, obtained by division by d.

The combinatorial definition (expansion). Loosely speaking, expansion requires that any (not too big) set of vertices of the graph has a relatively large set of neighbors. Specifically, a graph $G=(V,E)$ is c-expanding if, for every set $S\subset V$ of cardinality at most $|V|/2$, it holds that

$$\Gamma_G(S) \stackrel{\rm def}{=} \{v : \exists u\in S \text{ s.t. } \{u,v\}\in E\} \tag{E.4}$$

has cardinality at least $(1+c)\cdot|S|$. Assuming the existence of self-loops on all vertices, the foregoing requirement is equivalent to requiring that $|\Gamma_G(S)\setminus S| \geq c\cdot|S|$. In this case, every connected graph $G=(V,E)$ is $(1/|V|)$-expanding.[11] The combinatorial definition of expanders refers to an infinite family of d-regular graphs and requires the existence of a *constant* expansion bound that holds for all the graphs in the family.

Definition E.6 *An infinite family of d-regular graphs, $\{G_N\}_{N\in\mathbb{S}}$* is c-expanding *if for every $N\in\mathbb{S}$ it holds that G_N is c-expanding.*

The two definitions of expander graphs are related (see [11, Sec. 9.2] or [124, Sec. 4.5]). Specifically, the "expansion bound" and the "eigenvalue bound" are related as follows.

Theorem E.7 *Let G be a d-regular graph having a self-loop on each vertex.*[12]

1. *The graph G is c-expanding for $c\geq(d-\lambda(G))/2d$.*
2. *If G is c-expanding then $d-\lambda(G)\geq c^2/(4+2c^2)$.*

Thus, any non-zero bound on the combinatorial expansion of a family of d-regular graphs yields a non-zero bound on its eigenvalue gap, and vice versa. Note, however, that the back-and-forth translation between these measures is not tight. We note that the applications presented in the main text (see, e.g., Section 8.5.3 and §9.3.2.3) refer to the algebraic definition, and that the loss incurred in Theorem E.7 is immaterial for them.

[11]In contrast, a bipartite graph $G=(V,E)$ is not expanding, because it always contains a set S of size at most $|V|/2$ such that $|\Gamma_G(S)|\leq|S|$ (although it may hold that $|\Gamma_G(S)\setminus S|\geq|S|$).

[12]Recall that in such a graph $G=(V,E)$ it holds that $\Gamma_G(S)\supseteq S$ for every $S\subseteq V$, and thus $|\Gamma_G(S)|=|\Gamma_G(S)\setminus S|+|S|$. Furthermore, in such a graph all eigenvalues are greater than or equal to $-d+1$, and thus if $d-\lambda(G)<1$ then this is due to a positive eigenvalue of G. These facts are used for bridging the gap between Theorem E.7 and the more standard versions (see, e.g., [11, Sec. 9.2]) that refer to variants of both definitions. Specifically, [11, Sec. 9.2] refers to $\Gamma_G^+(S)=\Gamma_G(S)\setminus S$ and $\lambda_2(G)$, where $\lambda_2(G)$ is the second largest eigenvalue of G, rather than referring to $\Gamma_G(S)$ and $\lambda(G)$. Note that, in general, $\Gamma_G(S)$ may be attained by the difference between the smallest eigenvalue of G (which may be negative) and $-d$.

Amplification. The "quality of expander graphs improves" by raising these graphs to any power $t > 1$ (i.e., raising their adjacency matrix to the t^{th} power), where this operation corresponds to replacing t-paths (in the original graphs) by edges (in the resulting graphs). Specifically, when considering the algebraic definition, it holds that $\lambda(G^t) = \lambda(G)^t$, but indeed the degree also gets raised to the power t. Still, the ratio $\lambda(G^t)/d^t$ deceases with t. An analogous phenomenon occurs also under the combinatorial definition, provided that some suitable modifications are applied. For example, if for every $S \subseteq V$ it holds that $|\Gamma_G(S)| \geq \min((1+c) \cdot |S|, |V|/2)$, then for every $S \subseteq V$ it holds that $|\Gamma_{G^t}(S)| \geq \min((1+c)^t \cdot |S|, |V|/2)$.

The optimal eigenvalue bound. For every d-regular graph $G = (V, E)$, it holds that $\lambda(G) \geq 2\gamma_G \cdot \sqrt{d-1}$, where $\gamma_G = 1 - O(1/\log_d |V|)$. Thus, for any infinite family of (d, λ)-expanders, it must holds that $\lambda \geq 2\sqrt{d-1}$.

E.2.1.2 Two levels of explicitness

Towards discussing various notions of explicit constructions of graphs, we need to fix a representation of such graphs. Specifically, throughout this section, when referring to an infinite family of graphs $\{G_N\}_{N \in \mathbb{S}}$, we shall assume that the vertex set of G_N equals $[N]$. Indeed, at times, we shall consider vertex sets having a different structure (e.g., $[m] \times [m]$ for some $m \in \mathbb{N}$), but in all these cases there exists a simple isomorphism of these sets to the canonical representation (i.e., there exists an efficiently computable and invertible mapping of the vertex set of G_N to $[N]$).

Recall that a mild notion of explicit constructiveness refers to the *complexity of constructing the entire object* (i.e., the graph). Applying this notion to our setting, we say that an infinite family of graphs $\{G_N\}_{N \in \mathbb{S}}$ is explicitly constructible if there exists a *polynomial-time algorithm that, on input* 1^N (where $N \in \mathbb{S}$), *outputs the list of the edges in the N-vertex graph G_N*. That is, the entire graph is constructed in time that is polynomial in its size (i.e., in poly(N)-time).

The foregoing (mild) level of explicitness suffices when the application requires holding the entire graph and/or when the running-time of the application is lower-bounded by the size of the graph. In contrast, other applications refer to a huge virtual graph (which is much bigger than their running time), and only require the computation of the neighborhood relation in such a graph. In this case, the following stronger level of explicitness is relevant.

A strongly explicit construction of an infinite family of (d-regular) graphs $\{G_N\}_{N \in \mathbb{S}}$ is a *polynomial-time algorithm that on input $N \in \mathbb{S}$* (in binary), *a vertex v in the N-vertex graph G_N* (i.e., $v \in [N]$), *and an index $i \in [d]$, returns the i^{th} neighbor of v*. That is, the "neighbor query" is answered in time that is polylogarithmic in the size of the graph. Needless to say, this strong level of explicitness implies the basic (mild) level.

An additional requirement, which is often forgotten but is very important, refers to the "tractability" of the set $\mathbb{S}$. Specifically, we require the existence of an *efficient algorithm that given any $n \in \mathbb{N}$ finds an $s \in \mathbb{S}$ such that $n \leq s < 2n$.*

Corresponding to the two foregoing levels of explicitness, "efficient" may mean either running in time $\text{poly}(n)$ or running in time $\text{poly}(\log n)$. The requirement that $n \le s < 2n$ suffices in most applications, but in some cases a smaller interval (e.g., $n \le s < n + \sqrt{n}$) is required, whereas in other cases a larger interval (e.g., $n \le s < \text{poly}(n)$) suffices.

Greater flexibility. In continuation to the foregoing paragraph, we comment that expanders can be combined in order to obtain expanders for a wider range of graph sizes. For example, given two d-regular c-expanding graphs, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ where $|V_1| \le |V_2|$ and $c \le 1$, we can obtain a $(d+1)$-regular $c/2$-expanding graph on $|V_1| + |V_2|$ vertices by connecting the two graphs using a perfect matching of V_1 and $|V_1|$ of the vertices of V_2 (and adding self-loops to the remaining vertices of V_2). More generally, combining the d-regular c-expanding graphs $G_1 = (V_1, E_1)$ through $G_t = (V_t, E_t)$, where $N' \stackrel{\text{def}}{=} \sum_{i=1}^{t-1} |V_i| \le |V_t|$, yields a $(d+1)$-regular $c/2$-expanding graph on $\sum_{i=1}^{t} |V_i|$ vertices (by using a perfect matching of $\cup_{i=1}^{t-1} V_i$ and N' of the vertices of V_t).

E.2.1.3 Two properties

The following two properties provide a quantitative interpretation to the statement that expanders approximate the complete graph (or behave approximately like a complete graph). When referring to (d, λ)-expanders, the deviation from the behavior of a complete graph is represented by an error term that is linear in λ/d.

The mixing lemma. Loosely speaking, the following (folklore) lemma asserts that in expander graphs (for which $\lambda \ll d$) the fraction of edges connecting two large sets of vertices approximately equals the product of the densities of these sets. This property is called *mixing*.

Lemma E.8 (Expander Mixing Lemma): *For every d-regular graph $G = (V, E)$ and for every two subsets $A, B \subseteq V$ it holds that*

$$\left| \frac{|(A \times B) \cap \vec{E}|}{|\vec{E}|} - \frac{|A|}{|V|} \cdot \frac{|B|}{|V|} \right| \le \frac{\lambda(G)\sqrt{|A| \cdot |B|}}{d \cdot |V|} \le \frac{\lambda(G)}{d} \qquad \text{(E.5)}$$

where $\vec{E}$ denotes the set of directed edges (i.e., vertex pairs) *that correspond to the undirected edges of G* (i.e., $\vec{E} = \{(u,v) : \{u,v\} \in E\}$ and $|\vec{E}| = d|V|$).

In particular, $|(A \times A) \cap \vec{E}| = (\rho(A) \cdot d \pm \lambda(G)) \cdot |A|$, where $\rho(A) = |A|/|V|$. It follows that $|(A \times (V \setminus A)) \cap \vec{E}| = ((1 - \rho(A)) \cdot d \pm \lambda(G)) \cdot |A|$.

Proof: Let $N \stackrel{\text{def}}{=} |V|$ and $\lambda \stackrel{\text{def}}{=} \lambda(G)$. For any subset of the vertices $S \subseteq V$, we denote its density in V by $\rho(S) \stackrel{\text{def}}{=} |S|/N$. Hence, Eq. (E.5) is restated as

$$\left| \frac{|(A \times B) \cap \vec{E}|}{d \cdot N} - \rho(A) \cdot \rho(B) \right| \le \frac{\lambda\sqrt{\rho(A) \cdot \rho(B)}}{d}.$$

We proceed by providing bounds on the value of $|(A \times B) \cap \vec{E}|$. To this end we let $\overline{a}$ denote the N-dimensional Boolean vector having 1 in the i^{th} component if and only if $i \in A$. The vector $\overline{b}$ is defined similarly. Denoting the adjacency matrix of the graph G by $M = (m_{i,j})$, we note that $|(A \times B) \cap \vec{E}|$ equals $\overline{a}^\top M \overline{b}$ (because $(i,j) \in (A \times B) \cap \vec{E}$ if and only if it holds that $i \in A$, $j \in B$ and $m_{i,j} = 1$). We consider the *orthogonal eigenvector basis*, $\overline{e_1}, ..., \overline{e_N}$, where $\overline{e_1} = (1, ..., 1)^\top$ and $\overline{e_i}^\top \overline{e_i} = N$ for each i, and write each vector as a linear combination of the vectors in this basis. Specifically, we denote by a_i the coefficient of $\overline{a}$ in the direction of $\overline{e_i}$; that is, $a_i = (\overline{a}^\top \overline{e_i})/N$ and $\overline{a} = \sum_i a_i \overline{e_i}$. Note that $a_1 = (\overline{a}^\top \overline{e_1})/N = |A|/N = \rho(A)$ and $\sum_{i=1}^N a_i^2 = (\overline{a}^\top \overline{a})/N = |A|/N = \rho(A)$. Similarly for $\overline{b}$. It now follows that

$$\begin{aligned} |(A \times B) \cap \vec{E}| &= \overline{a}^\top M \sum_{i=1}^N b_i \overline{e_i} \\ &= \sum_{i=1}^N b_i \lambda_i \cdot \overline{a}^\top \overline{e_i} \end{aligned}$$

where λ_i denotes the i^{th} eigenvalue of M. Note that $\lambda_1 = d$ and for every $i \geq 2$ it holds that $|\lambda_i| \leq \lambda$. Thus,

$$\begin{aligned} \frac{|(A \times B) \cap \vec{E}|}{dN} &= \sum_{i=1}^N \frac{b_i \lambda_i \cdot a_i}{d} \\ &= \rho(A)\rho(B) + \sum_{i=2}^N \frac{\lambda_i a_i b_i}{d} \\ &\in \left[\rho(A)\rho(B) \pm \frac{\lambda}{d} \cdot \sum_{i=2}^N a_i b_i \right] \end{aligned}$$

Using $\sum_{i=1}^N a_i^2 = \rho(A)$ and $\sum_{i=1}^N b_i^2 = \rho(B)$, and applying Cauchy-Schwartz Inequality, we bound $\sum_{i=2}^N a_i b_i$ by $\sqrt{\rho(A)\rho(B)}$. The lemma follows. ■

The random walk lemma. Loosely speaking, the first part of the following lemma asserts that, as far as remaining "trapped" in some subset of the vertex set is concerned, a random walk on an expander approximates a random walk on the complete graph.

Lemma E.9 (Expander Random Walk Lemma): *Let $G = ([N], E)$ be a d-regular graph, and consider walks on G that start from a uniformly chosen vertex and take $\ell - 1$ additional random steps, where in each such step we uniformly selects one out of the d edges incident at the current vertex and traverses it.*

Theorem 8.28 (restated): *Let W be a subset of $[N]$ and $\rho \stackrel{\text{def}}{=} |W|/N$. Then the probability that such a random walk stays in W is at most*

$$\rho \cdot \left(\rho + (1 - \rho) \cdot \frac{\lambda(G)}{d} \right)^{\ell - 1} \qquad \text{(E.6)}$$

Exercise 8.43 (restated): *For any* $W_0, ..., W_{\ell-1} \subseteq [N]$*, the probability that a random walk of length* ℓ *intersects* $W_0 \times W_1 \times \cdots \times W_{\ell-1}$ *is at most*

$$\sqrt{\rho_0} \cdot \prod_{i=1}^{\ell-1} \sqrt{\rho_i + (\lambda/d)^2}, \qquad \text{(E.7)}$$

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

The basic principle underlying Lemma E.9 was discovered by Ajtai, Komlos, and Szemerédi [4], who proved a bound as in Eq. (E.7). The better analysis yielding Theorem 8.28 is due to [135, Cor. 6.1]. A more general bound that refer to the probability of visiting W for a number of times that approximates $|W|/N$ is given in [120], which actually considers an even more general problem (i.e., obtaining Chernoff-type bounds for random variables that are generated by a walk on an expander).

Proof of Equation (E.7): The basic idea is viewing events occuring during the random walk as an evolution of a corresponding probability vector under suitable transformations. The transformations correspond to taking a random step in G 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, whereas the second transformation shrinks the component that is in the direction of the uniform distribution. Details follow.

Let A be a matrix representing the random walk on G (i.e., A is the adjacency matrix of G divided by d), and let $\hat{\lambda} \stackrel{\text{def}}{=} \lambda(G)/d$ (i.e., $\hat{\lambda}$ upper-bounds the absolute value of every eigenvalue of A except the first one). Note that the uniform distribution, represented by the vector $\overline{u} = (N^{-1}, ..., N^{-1})^\top$, is the eigenvector of A that is associated with the largest eigenvalue (which is 1). Let P_i be a 0-1 matrix that has 1-entries only on its diagonal such that entry (j, j) is set to 1 if and only if $j \in W_i$. Then, the probability that a random walk of length ℓ intersects $W_0 \times W_1 \times \cdots \times W_{\ell-1}$ is the sum of the entries of the vector

$$\overline{v} \stackrel{\text{def}}{=} P_{\ell-1}A \cdots P_2 A P_1 A P_0 \overline{u}. \qquad \text{(E.8)}$$

We are interested in upper-bounding $\|\overline{v}\|_1$, and use $\|\overline{v}\|_1 \leq \sqrt{N} \cdot \|\overline{v}\|$, where $\|\overline{z}\|_1$ and $\|\overline{z}\|$ denote the L_1-norm and L_2-norm of $\overline{z}$, respectively (e.g., $\|\overline{u}\|_1 = 1$ and $\|\overline{u}\| = N^{-1/2}$). The key observation is that the linear transformation $P_i A$ shrinks every vector.

Main Claim. For every $\overline{z}$, it holds that $\|P_i A \overline{z}\| \leq (\rho_i + \hat{\lambda}^2)^{1/2} \cdot \|\overline{z}\|$.

Proof. Intuitively, A shrinks the component of $\overline{z}$ that is orthogonal to $\overline{u}$, whereas P_i shrinks the component of $\overline{z}$ that is in the direction of $\overline{u}$. Specifically, we decompose $\overline{z} = \overline{z_1} + \overline{z_2}$ such that $\overline{z_1}$ is the projection of $\overline{z}$ on $\overline{u}$ and $\overline{z_2}$ is the component orthogonal to $\overline{u}$. Then, using the triangle inequality and other obvious facts (which imply $\|P_iA\overline{z_1}\| = \|P_i\overline{z_1}\|$ and $\|P_iA\overline{z_2}\| \leq \|A\overline{z_2}\|$), we have

$$\begin{aligned}\|P_iA\overline{z_1} + P_iA\overline{z_2}\| &\leq \|P_iA\overline{z_1}\| + \|P_iA\overline{z_2}\| \\ &\leq \|P_i\overline{z_1}\| + \|A\overline{z_2}\| \\ &\leq \sqrt{\rho_i} \cdot \|\overline{z_1}\| + \hat{\lambda} \cdot \|\overline{z_2}\|\end{aligned}$$

where the last inequality uses the fact that P_i shrinks any uniform vector by eliminating $1-\rho_i$ of its elements, whereas A shrinks the length of any eigenvector except $\overline{u}$ by a factor of at least $\hat{\lambda}$. Using the Cauchy-Schwartz inequality[13], we get

$$\begin{aligned}\|P_iA\overline{z}\| &\leq \sqrt{\rho_i + \hat{\lambda}^2} \cdot \sqrt{\|\overline{z_1}\|^2 + \|\overline{z_2}\|^2} \\ &= \sqrt{\rho_i + \hat{\lambda}^2} \cdot \|\overline{z}\|\end{aligned}$$

where the equality is due to the fact that $\overline{z_1}$ is orthogonal to $\overline{z_2}$. $\square$

Recalling Eq. (E.8) and using the Main Claim (and $\|\overline{v}\|_1 \leq \sqrt{N} \cdot \|\overline{v}\|$), we get

$$\begin{aligned}\|\overline{v}\|_1 &\leq \sqrt{N} \cdot \|P_{\ell-1}A \cdots P_2AP_1AP_0\overline{u}\| \\ &\leq \sqrt{N} \cdot \left(\prod_{i=1}^{\ell-1} \sqrt{\rho_i + \hat{\lambda}^2}\right) \cdot \|P_0\overline{u}\|.\end{aligned}$$

Finally, using $\|P_0\overline{u}\| = \sqrt{\rho_0 N \cdot (1/N)^2} = \sqrt{\rho_0/N}$, we establish Eq. (E.7). $\blacksquare$

Rapid mixing. A property related to Lemma E.9 is that a random walk starting at any vertex converges to the uniform distribution on the expander vertices after a logarithmic number of steps. Specifically, we claim that *starting at any distribution* $\overline{s}$ (including a distribution that assigns all weight to a single vertex) *after ℓ steps on a (d, λ)-expander $G = ([N], E)$ we reach a distribution that is $\sqrt{N} \cdot (\lambda/d)^\ell$-close to the uniform distribution over* $[N]$. Using notation as in the proof of Eq. (E.7), the claim asserts that $\|A^\ell\overline{s} - \overline{u}\|_1 \leq \sqrt{N} \cdot \hat{\lambda}^\ell$, which is meaningful only for $\ell > 0.5 \cdot \log_{1/\hat{\lambda}} N$. The claim is proved by recalling that $\|A^\ell\overline{s} - \overline{u}\|_1 \leq \sqrt{N} \cdot \|A^\ell\overline{s} - \overline{u}\|$ and using the fact that $\overline{s} - \overline{u}$ is orthogonal to $\overline{u}$ (because the former is a zero-sum vector). Thus, $\|A^\ell\overline{s} - \overline{u}\| = \|A^\ell(\overline{s} - \overline{u})\| \leq \hat{\lambda}^\ell\|\overline{s} - \overline{u}\|$ and using $\|\overline{s} - \overline{u}\| < 1$ the claim follows.

E.2.2 Constructions

Many explicit constructions of (d, λ)-expanders are known. The first such construction was presented in [164] (where $\lambda < d$ was not explicitly bounded), and an optimal construction (i.e., an optimal eigenvalue bound of $\lambda = 2\sqrt{d-1}$) was first

[13]That is, we get $\sqrt{\rho_i}\|z_1\| + \hat{\lambda}\|z_2\| \leq \sqrt{\rho_i + \hat{\lambda}^2} \cdot \sqrt{\|z_1\|^2 + \|z_2\|^2}$, by using $\sum_{i=1}^n a_i \cdot b_i \leq \left(\sum_{i=1}^n {a_i}^2\right)^{1/2} \cdot \left(\sum_{i=1}^n {b_i}^2\right)^{1/2}$, with $n = 2$, $a_1 = \sqrt{\rho_i}$, $b_1 = \|z_1\|$, etc.

provided in [160]. Most of these constructions are quite simple (see, e.g., §E.2.2.1), but their analysis is based on non-elementary results from various branches of mathematics. In contrast, the construction of Reingold, Vadhan, and Wigderson [191], presented in §E.2.2.2, is based on an iterative process, and its analysis is based on a relatively simple algebraic fact regarding the eigenvalues of matrices.

Before turning to these explicit constructions we note that it is relatively easy to prove the existence of 3-regular expanders, by using the Probabilistic Method (cf. [11]) and referring to the combinatorial definition of expansion.[14]

E.2.2.1 The Margulis–Gabber–Galil Expander

For every natural number m, consider the graph with vertex set $\mathbb{Z}_m \times \mathbb{Z}_m$ and the edge set in which every $\langle x, y\rangle \in \mathbb{Z}_m \times \mathbb{Z}_m$ is connected to the vertices $\langle x \pm y, y\rangle$, $\langle x \pm (y+1), y\rangle$, $\langle x, y \pm x\rangle$, and $\langle x, y \pm (x+1)\rangle$, where the arithmetic is modulo m. This yields an extremely simple 8-regular graph with an eigenvalue bound that is a constant $\lambda < 8$ (which is independent of m). Thus, we get:

Theorem E.10 *There exists a strongly explicit construction of a family of* $(8, 7.9999)$-*expanders for graph sizes* $\{m^2 : m \in \mathbb{N}\}$. *Furthermore, the neighbors of a vertex in these expanders can be computed in logarithmic-space.*[15]

An appealing property of Theorem E.10 is that, for every $n \in \mathbb{N}$, it directly yields expanders with vertex set $\{0,1\}^n$. This is obvious in case n is even, but can be easily achieved also for odd n (e.g., use two copies of the graph for $n-1$, and connect the two copies by the obvious perfect matching).

Theorem E.10 is due to Gabber and Galil [84], building on the basic approach suggested by Margulis [164]. We mention again that the (strongly explicit) (d, λ)-expanders of [160] achieve the optimal eigenvalue bound (i.e., $\lambda = 2\sqrt{d-1}$), but there are annoying restrictions on the degree d (i.e., $d-1$ should be a prime congruent to 1 modulo 4) and on the graph sizes for which this construction works.[16]

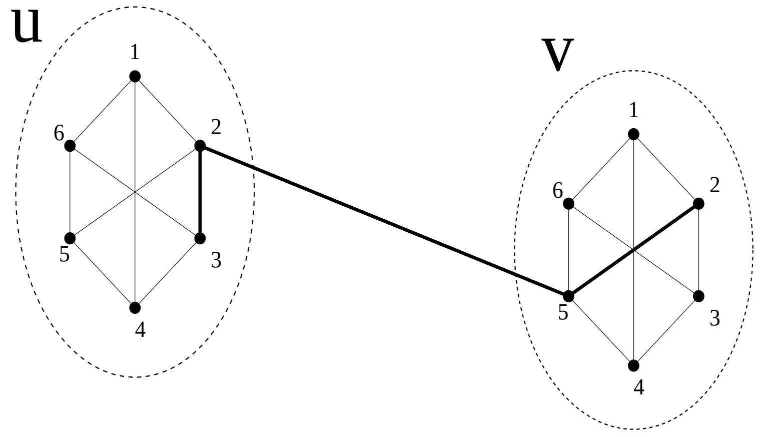
[14]This can be done by considering a 3-regular graph obtained by combining an N-cycle with a random matching of the first $N/2$ vertices and the remaining $N/2$ vertices. It is actually easier to prove the related statement that refers to the alternative definition of combinatorial expansion that refers to the relative size of $\Gamma_G^+(S) = \Gamma_G(S) \setminus S$ (rather than to the relative size of $\Gamma_G(S)$). In this case, for a sufficiently small $\varepsilon > 0$ and all sufficiently large N, a random 3-regular N-vertex graph is "ε-expanding" with overwhelmingly high probability. The proof proceeds by considering a (not necessarily simple) graph G obtained by combining three uniformly chosen perfect matchings of the elements of $[N]$. For every $S \subseteq [N]$ of size at most $N/2$ and for every set T of size $\varepsilon|S|$, we consider the probability that for a random perfect matching M it holds that $\Gamma_M^+(S) \subseteq T$. The argument is concluded by applying a union bound.

[15]In fact, for m that is a power of two (and under a suitable encoding of the vertices), the neighbors can be computed by a on-line algorithm that uses a constant amount of space. The same holds also for a variant in which each vertex $\langle x, y\rangle$ is connected to the vertices $\langle x \pm 2y, y\rangle$, $\langle x \pm (2y+1), y\rangle$, $\langle x, y \pm 2x\rangle$, and $\langle x, y \pm (2x+1)\rangle$. This variant yields a better known bound on λ, i.e., $\lambda \leq 5\sqrt{2} \approx 7.071$.

[16]The construction in [160] allows graph sizes of the form $(p^3 - p)/2$, where $p \equiv 1 \pmod 4$ is a prime such that $d-1$ is a quadratic residue modulo p. As stated in [8, Sec. 2], the construction can be extended to graph sizes of the form $(p^{3k} - p^{3k-2})/2$, for any $k \in \mathbb{N}$ and p as in the foregoing.

E.2.2.2 The Iterated Zig-Zag Construction

The starting point of the following construction is a very good expander G of *constant size*, which may be found by an exhaustive search. The construction of a large expander graph proceeds in iterations, where in the i^{th} iteration the current graph G_i and the fixed graph G are combined, resulting in a larger graph G_{i+1}. The combination step guarantees that the expansion property of G_{i+1} is at least as good as the expansion of G_i, while G_{i+1} maintains the degree of G_i and is a constant times larger than G_i. The process is initiated with $G_1 = G^2$ and terminates when we obtain a graph G_t of approximately the desired size (which requires a logarithmic number of iterations).



In this example G' is 6-regular and G is a 3-regular graph having six vertices. In the graph G' (not shown), the 2nd edge of vertex u is incident at v, as its 5th edge. The wide 3-segment line shows one of the corresponding edges of G'ⓩG, which connects the vertices $\langle u,3\rangle$ and $\langle v,2\rangle$.

Figure E.1: Detail of the zig-zag product of G' and G.

The Zig-Zag product. The heart of the combination step is a new type of "graph product" called *Zig-Zag product.* This operation is applicable to any pair of graphs $G = ([D], E)$ and $G' = ([N], E')$, provided that G' (which is typically larger than G) is D-regular. For simplicity, we assume that G is d-regular (where typically $d \ll D$). The Zig-Zag product of G' and G, denoted G'ⓩG, is defined as a graph with vertex set $[N] \times [D]$ and an edge set that includes an edge between $\langle u,i\rangle \in [N] \times [D]$ and $\langle v,j\rangle$ if and only if $\{i,k\}, \{\ell,j\} \in E$ and the k^{th} edge incident at u equals the ℓ^{th} edge incident at v. That is, $\langle u,i\rangle$ and $\langle v,j\rangle$ are connected in G'ⓩG if there exists a "three step sequence" consisting of a G-step from $\langle u,i\rangle$ to $\langle u,k\rangle$ (according to the edge $\{i,k\}$ of G), followed by a G'-step from $\langle u,k\rangle$ to $\langle v,\ell\rangle$

(according to the k^{th} edge of u in G' (which is the ℓ^{th} edge of v)), and a final G-step from $\langle v,\ell\rangle$ to $\langle v,j\rangle$ (according to the edge $\{\ell,j\}$ of G). See Figure E.1 as well as further formalization (which follows).

Teaching note: The following paragraph, which provides a formal description of the zig-zag product, can be ignored in first reading but is useful for more advanced discussion.

It will be convenient to represent graphs like G' by their edge-rotation function, denoted $R' : [N] \times [D] \to [N] \times [D]$, such that $R'(u,i) = (v,j)$ if $\{u,v\}$ is the i^{th} edge incident at u as well as the j^{th} edge incident at v. That is, R' rotates the pair (u,i), which represents one "side" of the edge $\{u,v\}$ (i.e., the side incident at u as its i^{th} edge), resulting in the pair (v,j), which represents the other side of the same edge (which is the j^{th} edge incident at v). For simplicity, we assume that the (constant-size) d-regular graph $G = ([D],E)$ is edge-colorable with d colors, which in turn yields a natural edge-rotation function (i.e., $R(i,\alpha) = (j,\alpha)$ if the edge $\{i,j\}$ is colored α). We will denote by $E_\alpha(i)$ the vertex reached from $i \in [D]$ by following the edge colored α (i.e., $E_\alpha(i) = j$ iff $R(i,\alpha) = (j,\alpha)$). The Zig-Zag product of G' and G, denoted $G'ⓩG$, is then defined as a graph with the vertex set $[N] \times [D]$ and the edge-rotation function

$$(\langle u,i\rangle, \langle \alpha,\beta\rangle) \mapsto (\langle v,j\rangle, \langle \beta,\alpha\rangle) \quad \text{if } R'(u,E_\alpha(i)) = (v,E_\beta(j)). \tag{E.9}$$

That is, edges are labeled by pairs over $[d]$, and the $\langle\alpha,\beta\rangle^{\text{th}}$ edge out of vertex $\langle u,i\rangle \in [N]\times[D]$ is incident at the vertex $\langle v,j\rangle$ (as its $\langle\beta,\alpha\rangle^{\text{th}}$ edge) if $R(u,E_\alpha(i)) = (v,E_\beta(j))$, where indeed $E_\beta(E_\beta(j)) = j$. Intuitively, based on $\langle\alpha,\beta\rangle$, we first take a G-step from $\langle u,i\rangle$ to $\langle u,E_\alpha(i)\rangle$, then viewing $\langle u,E_\alpha(i)\rangle \equiv (u,E_\alpha(i))$ as a side of an edge of G' we rotate it (i.e., we effectively take a G'-step) reaching $(v,j') \stackrel{\text{def}}{=} R'(u,E_\alpha(i))$, and finally we take a G-step from $\langle v,j'\rangle$ to $\langle v,E_\beta(j')\rangle$.

Clearly, the graph $G'ⓩG$ is d^2-regular and has $D \cdot N$ vertices. The key fact, proved in [191] (using techniques as in §E.2.1.3), is that the relative eigenvalue-value of the zig-zag product is upper-bounded by the sum of the relative eigenvalue-values of the two graphs; that is, $\bar{\lambda}(G'ⓩG) \le \bar{\lambda}(G')+\bar{\lambda}(G)$, where $\bar{\lambda}(\cdot)$ denotes the relative eigenvalue-bound of the relevant graph. The (qualitative) fact that $G'ⓩG$ is an expander if both G' and G are expanders is very intuitive (e.g., consider what happens if G' or G is a clique). Things are even more intuitive if one considers the (related) replacement product of G' and G, denoted $G'ⓡG$, *where there is an edge between $\langle u,i\rangle \in [N] \times [D]$ and $\langle v,j\rangle$ if and only if either $u = v$ and $\{i,j\} \in E$ or the i^{th} edge incident at u equals the j^{th} edge incident at v.*

The iterated construction. The iterated expander construction uses the aforementioned zig-zag product as well as graph squaring. Specifically, the construction starts[17] with the d^2-regular graph $G_1 = G^2 = ([D],E^2)$, where $D = d^4$ and $\bar{\lambda}(G) < 1/4$, and proceeds in iterations such that $G_{i+1} = G_i^2ⓩG$ for $i = 1,2,...,t-1$,

[17]Recall that, for a sufficiently large constant d, we first find a d-regular graph $G = ([d^4],E)$ satisfying $\bar{\lambda}(G) < 1/4$, by exhaustive search.

where t is logarithmic in the desired graph size. That is, in each iteration, the current graph is first squared and then composed with the fixed (d-regular D-vertex) graph G via the zig-zag product. This process maintains the following two invariants:

1. The graph G_i is d^2-regular and has D^i vertices.

   (The degree bound follows from the fact that a zig-zag product with a d-regular graph always yields a d^2-regular graph.)

2. The relative eigenvalue-bound of G_i is smaller than one half (i.e., $\bar{\lambda}(G_i) < 1/2$).

   (Here we use the fact that $\bar{\lambda}(G_{i-1}^2 \textcircled{z} G) \leq \bar{\lambda}(G_{i-1}^2) + \bar{\lambda}(G)$, which in turn equals $\bar{\lambda}(G_{i-1})^2 + \bar{\lambda}(G) < (1/2)^2 + (1/4)$. Note that graph squaring is used to reduce the relative eigenvalue of G_i before increasing it by zig-zag product with G.)

In order to show that we can actually construct G_i, we show that we can compute the edge-rotation function that correspond to its edge set. This boils down to showing that, given the edge-rotation function of G_{i-1}, we can compute the edge-rotation function of G_{i-1}^2 as well as of its zig-zag product with G. Note that this entire computation amounts to two recursive calls to computations regarding G_{i-1} (and two computations that correspond to the constant graph G). But since the recursion depth is logarithmic in the size of the final graph (i.e., $t = \log_D |\text{vertices}(G_t)|$), the total number of recursive calls is polynomial in the size of the final graph (and thus the entire computation is polynomial in the size of the final graph). This suffices for the minimal (i.e., "mild") notion of explicitness, but not for the strong one.

The strongly explicit version. To achieve a *strongly explicit construction*, we slightly modify the iterative construction. Rather than letting $G_{i+1} = G_i^2 \textcircled{z} G$, we let $G_{i+1} = (G_i \times G_i)^2 \textcircled{z} G$, where $G' \times G'$ denotes the *tensor product of* G' *with itself*; that is, if $G' = (V', E')$ then $G' \times G' = (V' \times V', E'')$, where

$$E'' = \{\{\langle u_1, u_2\rangle, \langle v_1, v_2\rangle\} : \{u_1, v_1\}, \{u_2, v_2\} \in E'\}$$

(i.e., $\langle u_1, u_2\rangle$ and $\langle v_1, v_2\rangle$ are connected in $G' \times G'$ if for $i = 1, 2$ it holds that u_i is connected to v_i in G'). The corresponding edge-rotation function is

$$R''(\langle u_1, u_2\rangle, \langle i_1, i_2\rangle) = (\langle v_1, v_2\rangle, \langle j_1, j_2\rangle),$$

where $R'(u_1, i_1) = (v_1, j_1)$ and $R'(u_2, i_2) = (v_2, j_2)$. We still use $G_1 = G^2$, where (as before) G is d-regular and $\bar{\lambda}(G) < 1/4$, but here G has $D = d^8$ vertices.[18] Using the fact that tensor product preserves the relative eigenvalue-bound while squaring the degree (and the number of vertices), we note that the modified iteration $G_{i+1} = (G_i \times G_i)^2 \textcircled{z} G$ yields a d^2-regular graph with $(D^{2^i - 1})^2 \cdot D = D^{2^{i+1}-1}$ vertices, and

[18] The reason for the change is that $(G_i \times G_i)^2$ will be d^8-regular, since G_i will be d^2-regular.

that $\bar{\lambda}(G_{i+1}) < 1/2$ (because $\bar{\lambda}((G_i \times G_i)^2 ⓩ G) \leq \bar{\lambda}(G_i)^2 + \bar{\lambda}(G)$). Computing the neighbor of a vertex in G_{i+1} boils down to a constant number of such computations regarding G_i, but due to the tensor product operation the depth of the recursion is only double-logarithmic in the size of the final graph (and hence logarithmic in the length of the description of vertices in this graph).

Digest. In the first construction, the zig-zag product was used both in order to increase the size of the graph and to reduce its degree. However, as indicated by the second construction (where the tensor product of graphs is the main vehicle for increasing the size of the graph), the primary effect of the zig-zag product is reducing the graph's degree, and the increase in the size of the graph is merely a side-effect.[19] In both cases, graph squaring is used in order to compensate for the modest increase in the relative eigenvalue-bound caused by the zig-zag product. In retrospect, the second construction is the "correct" one, because it decouples three different effects, and uses a natural operation to obtain each of them: Increasing the size of the graph is obtained by tensor product of graphs (which in turn increases the degree), the desired degree reduction is obtained by the zig-zag product (which in turn slightly increases the relative eigenvalue-bound), and graph squaring is used in order to reduce the relative eigenvalue-bound.

Stronger bound regarding the effect of the zig-zag product. In the foregoing description we relied on the fact, proved in [191], that the relative eigenvalue-bound of the zig-zag product is upper-bounded by the sum of the relative eigenvalue-bounds of the two graphs (i.e., $\bar{\lambda}(G' ⓩ G) \leq \bar{\lambda}(G') + \bar{\lambda}(G)$). Actually, a stronger upper-bound is proved in [191]: It holds that $\bar{\lambda}(G' ⓩ G) \leq f(\bar{\lambda}(G'), \bar{\lambda}(G))$, where

$$f(x,y) \stackrel{\rm def}{=} \frac{(1-y^2)\cdot x}{2} + \sqrt{\left(\frac{(1-y^2)\cdot x}{2}\right)^2 + y^2} \qquad \text{(E.10)}$$

Indeed, $f(x,y) \leq (1-y^2)\cdot x + y \leq x + y$. On the other hand, for $x \leq 1$, we have $f(x,y) \leq \frac{(1-y^2)\cdot x}{2} + \frac{1+y^2}{2} = 1 - \frac{(1-y^2)\cdot(1-x)}{2}$, which implies

$$\bar{\lambda}(G' ⓩ G) \;\leq\; 1 - \frac{(1-\bar{\lambda}(G)^2)\cdot(1-\bar{\lambda}(G'))}{2}. \qquad \text{(E.11)}$$

Thus, $1 - \bar{\lambda}(G' ⓩ G) \geq (1-\bar{\lambda}(G)^2)\cdot(1-\bar{\lambda}(G'))/2$, and it follows that the zig-zag product has a positive eigenvalue-gap if both graphs have positive eigenvalue-gaps (i.e., $\lambda(G' ⓩ G) < 1$ if both $\lambda(G) < 1$ and $\lambda(G') < 1$). Furthermore, if $\bar{\lambda}(G) < 1/\sqrt{3}$ then $1 - \bar{\lambda}(G' ⓩ G) > (1-\bar{\lambda}(G'))/3$. This fact plays an important role in the proof of Theorem 5.6.

[19]We mention that this side-effect may actually be undesired in some applications. For example, in Section 5.2.4 we would rather not have the graph grow in size, but we can tolerate the constant size blow-up (caused by zig-zag product with a constant-size graph).

Appendix F

Some Omitted Proofs

A word of a Gentleman is better than a proof,
but since you are not a Gentleman – please provide a proof.

Leonid A Levin (1986)

The proofs presented in this appendix were not included in the main text for a variety of reasons (e.g., they were deemed too technical and/or out-of-pace for the corresponding location). On the other hand, since our presentation of these proofs is sufficiently different from the original and/or standard presentation, we see a benefit in including these proofs in the current book.

Summary: This appendix contains proofs of the following results:

1. $\mathcal{PH}$ is reducible to $\#\mathcal{P}$ (and in fact to $\oplus\mathcal{P}$) via randomized Karp-reductions. The proof follows the underlying ideas of Toda's original proof, but the actual presentation is quite different.
2. For any integral function f that satisfies $f(n) \in \{2, ..., \text{poly}(n)\}$, it holds that $\mathcal{IP}(f) \subseteq \mathcal{AM}(O(f))$ and $\mathcal{AM}(O(f)) \subseteq \mathcal{AM}(f)$. The proofs differ from the original proofs (provided in [111] and [23], respectively) only in secondary details, but these details seem significant.

F.1 Proving that $\mathcal{PH}$ reduces to $\#\mathcal{P}$

Recall that Theorem 6.16 asserts that $\mathcal{PH}$ is Cook-reducible to $\#\mathcal{P}$ (via deterministic reductions). Here we prove a closely related result (also due to Toda [220]), which relaxes the requirement from the reduction (allowing it to be randomized) but uses an oracle to a seemingly weaker class. The latter class is denoted $\oplus\mathcal{P}$ and is the "modulo 2 analogue" of $\#\mathcal{P}$. Specifically, a Boolean function f is in $\oplus\mathcal{P}$ if there exists a function $g \in \#\mathcal{P}$ such that for every x it holds that

$f(x) = g(x) \bmod 2$. Equivalently, f is in $\oplus\mathcal{P}$ if there exists a search problem $R \in \mathcal{PC}$ such that $f(x) = |R(x)| \bmod 2$, where $R(x) = \{y : (x,y) \in R\}$. Thus, for any $R \in \mathcal{PC}$, the set $\oplus R \stackrel{\text{def}}{=} \{x : |R(x)| \equiv 1 \pmod 2\}$ is in $\oplus\mathcal{P}$. (The $\oplus$ symbol in the notation $\oplus\mathcal{P}$ actually represents parity, which is merely addition modulo 2. Indeed, a notation such as $\#_2\mathcal{P}$ would have been more appropriate.)

Theorem F.1 *Every set in $\mathcal{PH}$ is reducible to $\oplus\mathcal{P}$ via a probabilistic polynomial-time reduction. Furthermore, the reduction is via a many-to-one randomized mapping and it fails with negligible error probability.*

The proof follows the underlying ideas of the original proof [220], but the actual presentation is quite different. Alternative proofs of Theorem F.1 can be found in [136, 212].

Teaching note: It is quite easy to prove a non-uniform analogue of Theorem F.1, which asserts that AC0 circuits can be approximated by circuits consisting of an unbounded parity of conjunctions, where each conjunction has polylogarithmic fan-in. Turning this argument into a proof of Theorem F.1 requires a careful implementation as well as using transitions of the type presented in Exercise 3.8. Furthermore, such a presentation tends to obscure the conceptual steps that underly the argument.

Proof Outline: The proof uses three main ingredients. The first ingredient is the fact that $\mathcal{NP}$ is reducible to $\oplus\mathcal{P}$ via a probabilistic Karp-reduction, and that this reduction "relativizes" (i.e., reduces $\mathcal{NP}^A$ to $\oplus\mathcal{P}^A$ for any oracle A).[1] The second ingredient is the fact that error-reduction is available in the current context (of randomized reductions to $\oplus\mathcal{P}$), resulting in reductions that have exponentially vanishing error probability.[2] The third ingredient is the extension of the first ingredient to Σ_k, which relies on Proposition 3.9 as well as on the aforementioned error-reduction. These ingredients correspond to the three main steps of the proof, which are outlined next:

Step 1: Present a randomized Karp-reduction of $\mathcal{NP}$ to $\oplus\mathcal{P}$.

Step 2: Decrease the error probability of the foregoing Karp-reduction such that the error probability becomes exponentially vanishing. Such a low error probability is crucial as a starting point for the next step.

[1]Indeed, the "relativization" requirement presumes that both $\mathcal{NP}$ and $\oplus\mathcal{P}$ are each associated with a class of (standard) machines that generalizes to a class of corresponding oracle machines (see comment at Section 3.2.2). This presumption holds for both classes, by virtue of a (deterministic polynomial-time) machine that decide membership in the corresponding relation that belongs to $\mathcal{PC}$. Alternatively, one may use the fact that the aforementioned reduction is "highly structured" in the sense that for some polynomial-time computable predicate ψ this reduction maps x to $\langle x,s\rangle$ such that for every non-empty set $S_x \subseteq \{0,1\}^{p(|x|)}$ it holds that $\mathsf{Pr}_s[|\{y \in S_x : \psi(x,s,y)\}| \equiv 1 \pmod 2] > 1/3$.

[2]We comment that such an error-reduction is not available in the context of reductions to unique solution problems. This comment is made in view of the similarity between the reduction of $\mathcal{NP}$ to $\oplus\mathcal{P}$ and the reduction of $\mathcal{NP}$ to problems of unique solution.

Step 3: Prove that Σ_2 is randomly reducible to $\oplus\mathcal{P}$ by extending the reduction of Step 1 (while using Step 2). Intuitively, for any oracle A, the reduction of Step 1 offers a reduction of $\mathcal{NP}^A$ to $\oplus\mathcal{P}^A$, whereas a reduction of A to B having exponentially vanishing error probability allows reducing $\oplus\mathcal{P}^A$ to $\oplus\mathcal{P}^B$ (or, similarly, reduce $\mathcal{NP}^A$ to $\mathcal{NP}^B$). Observing that $\oplus\mathcal{P}^{\oplus\mathcal{P}} = \oplus\mathcal{P}$, we obtain a randomized Karp-reduction of Σ_2 (viewed as $\mathcal{NP}^{\mathcal{NP}}$) to $\oplus\mathcal{P}$.

When completing the third step, we shall have all the ingredients needed for the general case (of randomly reducing Σ_k to $\oplus\mathcal{P}$, for any $k \geq 2$). We shall finish the proof by sketching the extension of the case of Σ_2 (treated in Step 3) to the general case of Σ_k (for any $k \geq 2$). The actual extension is quite cumbersome, but the ideas are all present in the case of Σ_2. Furthermore, we believe that the case of Σ_2 is of significant interest per se.

Teaching note: The foregoing sketch of Step 3 suggests an abstract treatment that evolves around definitions such as $\mathcal{NP}^A$ and $\oplus\mathcal{P}^B$. We prefer a concrete presentation that performs Step 3 as an extension of Step 1 (while using Step 2). This is one reason for explicitly performing Step 1 (i.e., present a randomized Karp-reduction of $\mathcal{NP}$ to $\oplus\mathcal{P}$). We note that Step 1 (i.e., a reduction of $\mathcal{NP}$ to $\oplus\mathcal{P}$) follows immediately from the NP-hardness of deciding unique solution for some relations $R \in \mathcal{PC}$ (i.e., Theorem 6.29), because the promise problem $(\mathtt{US}_R, \overline{S}_R)$, where $\mathtt{US}_R = \{x : |R(x)| = 1\}$ and $\overline{S}_R = \{x : |R(x)| = 0\}$, is reducible to $\oplus R = \{x : |R(x)| \equiv 1 \pmod 2\}$ by the identity mapping. However, for the sake of self-containment and conceptual rightness, we present an alternative proof.

Step 1: a direct proof for the case of $\mathcal{NP}$. As in the proof of Theorem 6.29, we start with any $R \in \mathcal{PC}$ and our goal is reducing $S_R = \{x : |R(x)| \geq 1\}$ to $\oplus\mathcal{P}$ by a randomized Karp-reduction.[3] The standard way of obtaining such a reduction (e.g., in [136, 178, 212, 220]) consists of just using the reduction (to "unique solution") that was presented in the proof of Theorem 6.29, but we believe that this way is conceptually wrong. Let us explain.

Recall that the proof of Theorem 6.29 consists of implementing a randomized sieve that has the following property. For any $x \in S_R$, with noticeable probability, a single element of $R(x)$ passes the sieve (and this event can be detected by an oracle to a unique solution problem). Indeed, an adequate oracle in $\oplus\mathcal{P}$ correctly detects the case in which a single element of $R(x)$ passes the sieve. However, by definition, this oracle correctly detects the more general case in which any odd number of elements of $R(x)$ pass the sieve. Thus, insisting on a random sieve that allows the passing of a single element of $R(x)$ seems an over-kill (or at least is conceptually wrong). Instead, *we should just apply a less stringent random sieve that, with noticeable probability, allows the passing of an odd number of elements*

[3]As in Theorem 6.29, if any search problem in $\mathcal{PC}$ is reducible to R via a parsimonious reduction, then we can reduce S_R to $\oplus R$. Specifically, we shall show that S_R is randomly reducible to $\oplus R_2$, for some $R_2 \in \mathcal{PC}$, and a reduction of S_R to $\oplus R$ follows (by using the parsimonious reduction of R_2 to R).

of $R(x)$. The adequate tool for such a random sieve is a small-bias generator (see Section 8.5.2).

Indeed, we randomly reduce S_R to $\oplus\mathcal{P}$ by sieving potential solutions via a small-bias generator. Intuitively, we randomly map x to $\langle x,s\rangle$, where s is a random seed for such a generator, and y is considered a solution to the instance $\langle x,s\rangle$ if and only if $y \in R(x)$ and the y^{th} bit of $G(s)$ equals 1. (Indeed, if $|R(x)| \geq 1$ then, with probability approximately 1/2, the instance $\langle x,s\rangle$ has an odd number of solutions, whereas if $|R(x)| = 0$ then $\langle x,s\rangle$ has no solutions.) Specifically, we use a *strongly efficient* generator (see §8.5.2.1), denoted $G : \{0,1\}^k \to \{0,1\}^{\ell(k)}$, where $G(U_k)$ has bias at most 1/6 and $\ell(k) = \exp(\Omega(k))$. That is, given a seed $s \in \{0,1\}^k$ and index $i \in [\ell(k)]$, we can produce the i^{th} bit of $G(s)$, denoted $G(s,i)$, in polynomial-time. Assuming, without loss of generality, that $R(x) \subseteq \{0,1\}^{p(|x|)}$ for some polynomial p, we consider the relation

$$R_2 \stackrel{\text{def}}{=} \{(\langle x,s\rangle, y) : (x,y) \in R \wedge G(s,y) = 1\} \qquad \text{(F.1)}$$

where $y \in \{0,1\}^{p(|x|)} \equiv [2^{p(|x|)}]$ and $s \in \{0,1\}^{O(|y|)}$ such that $\ell(|s|) = 2^{|y|}$. In other words, $R_2(\langle x,s\rangle) = \{y : y \in R(x) \wedge G(s,y) = 1\}$. Then, for every $x \in S_R$, with probability at least 1/3, a uniformly selected $s \in \{0,1\}^{O(|y|)}$ satisfies $|R_2(\langle x,s\rangle)| \equiv 1 \pmod 2$, whereas for every $x \notin S_R$ and every $s \in \{0,1\}^{O(|y|)}$ it holds that $|R_2(\langle x,s\rangle)| = 0$. A key observation is that $R_2 \in \mathcal{PC}$ (and thus $\oplus R_2$ is in $\oplus\mathcal{P}$). Thus, deciding membership in S_R is randomly reducible to $\oplus R_2$ (by the many-to-one randomized mapping of x to $\langle x,s\rangle$, where s is uniformly selected in $\{0,1\}^{O(p(|x|))}$). Since the foregoing holds for any $R \in \mathcal{PC}$, it follows that $\mathcal{NP}$ is reducible to $\oplus\mathcal{P}$ via randomized Karp-reductions.

Dealing with co$\mathcal{NP}$**.** We may Cook-reduce co$\mathcal{NP}$ to $\mathcal{NP}$ and thus prove that co$\mathcal{NP}$ is randomly reducible to $\oplus\mathcal{P}$, but we wish to highlight the fact that a randomized Karp-reduction will also do. Starting with the reduction presented for the case of sets in $\mathcal{NP}$, we note that for $S \in \text{co}\mathcal{NP}$ (i.e., $S = \{x : R(x) = \emptyset\}$) we obtain a relation R_2 such that $x \in S$ is indicated by $|R_2(\langle x,\cdot\rangle)| \equiv 0 \pmod 2$. We wish to flip the parity such that $x \in S$ will be indicated by $|R_2(\langle x,\cdot\rangle)| \equiv 1 \pmod 2$, and this can be done by augmenting the relation R_2 with a single dummy solution per each x. For example, we may redefine $R_2(\langle x,s\rangle)$ as $\{0y : y \in R_2(\langle x,s\rangle)\} \cup \{10^{p(|x|)}\}$. Indeed, *we have just demonstrated and used the fact that* $\oplus\mathcal{P}$ *is closed under complementation.*

We note that dealing with the cases of $\mathcal{NP}$ and co$\mathcal{NP}$ is of interest only because we reduced these classes to $\oplus\mathcal{P}$ rather than to $\#\mathcal{P}$. In contrast, even a reduction of Σ_2 to $\#\mathcal{P}$ is of interest, and thus the reduction of Σ_2 to $\oplus\mathcal{P}$ (presented in Step 3) is interesting. This reduction relies heavily on the fact that error-reduction is applicable to the context of randomized Karp-reductions to $\oplus\mathcal{P}$.

Step 2: error reduction. An important observation, towards the core of the proof, is that it is possible to drastically decrease the (one-sided) error probability in randomized Karp-reductions to $\oplus\mathcal{P}$. Specifically, let R_2 be as in Eq. (F.1) and

t be any polynomial. Then, a binary relation $R_2^{(t)}$ that satisfies

$$|R_2^{(t)}(\langle x, s_1, ..., s_{t(|x|)}\rangle)| \;=\; 1 + \prod_{i=1}^{t(|x|)} (1 + |R_2(\langle x, s_i\rangle)|) \tag{F.2}$$

offers such an error-reduction, because $|R_2^{(t)}(\langle x, s_1, ..., s_{t(|x|)}\rangle)|$ is odd if and only if for some $i \in [t(|x|)]$ it holds that $|R_2(\langle x, s_i\rangle)|$ is odd. Thus,

$$\begin{aligned}&\mathsf{Pr}_{s_1,...,s_{t(|x|)}}[|R_2^{(t)}(\langle x, s_1, ..., s_{t(|x|)}\rangle)| \equiv 0 \pmod 2]\\ &\quad = \mathsf{Pr}_s[|R_2(\langle x, s\rangle)| \equiv 0 \pmod 2]^{t(|x|)}\end{aligned}$$

where $s, s_1, ..., s_{t(|x|)}$ are uniformly and independently distributed in $\{0,1\}^{O(p(|x|))}$ (and p is such that $R(x) \subseteq \{0,1\}^{p(|x|)}$). This means that the one-sided error probability of a randomized reduction of S_R to $\oplus R_2$ (which maps x to $\langle x, s\rangle$) can be drastically decreased by reducing S_R to $\oplus R_2^{(t)}$, where the reduction maps x to $\langle x, s_1, ..., s_{t(|x|)}\rangle$. Specifically, an error probability of ε (e.g., $\varepsilon = 2/3$) in the case that we desire an "odd outcome" (i.e., $x \in S_R$) is decreased to error probability ε^t, whereas the zero error probability in the case of a desired "even outcome" (i.e., $x \in \overline{S}_R$) is preserved.

A key question is whether $\oplus R_2^{(t)}$ is in $\oplus\mathcal{P}$; that is, whether $R_2^{(t)}$ (as postulated in Eq. (F.2)) can be implemented in $\mathcal{PC}$. The answer is positive, and this can be shown by using a Cartesian product construction (and adding some dummy solutions). For example, let $R_2^{(t)}(\langle x, s_1, ..., s_{t(|x|)}\rangle)$ consists of tuples $\langle \sigma_0, y_1, ..., y_{t(|x|)}\rangle$ such that either $\sigma_0 = 1$ and $y_1 = \cdots = y_{t(|x|)} = 0^{p(|x|)+1}$ or $\sigma_0 = 0$ and for every $i \in [t(|x|)]$ it holds that $y_i \in (\{0\} \times R_2(\langle x, s_i\rangle)) \cup \{10^{p(|x|)}\}$ (i.e., either $y_i = 10^{p(|x|)}$ or $y_i = 0y_i'$ and $y_i' \in R_2(\langle x, s_i\rangle)$).

We wish to stress that, when starting with R_2 as in Eq. (F.1), the forgoing process of error-reduction can be used for obtaining error probability that is upper-bounded by $\exp(-q(|x|))$ for any desired polynomial q. The importance of this comment will become clear shortly.

Step 3: the case of Σ_2. With the foregoing preliminaries, we are now ready to handle the case of $S \in \Sigma_2$. By Proposition 3.9, there exists a polynomial p and a set $S' \in \Pi_1 = \text{co}\mathcal{NP}$ such that $S = \{x : \exists y \in \{0,1\}^{p(|x|)} \text{ s.t. } (x,y) \in S'\}$. Using $S' \in \text{co}\mathcal{NP}$, we apply the forgoing reduction of S' to $\oplus\mathcal{P}$ as well as an adequate error-reduction that yields an upper-bound of $\varepsilon \cdot 2^{-p(|x|)}$ on the error probability, where $\varepsilon \leq 1/7$ is unspecified at this point. (For the case of Σ_2 the setting $\varepsilon = 1/7$ will do, but for the dealing with Σ_k we will need a much smaller value of $\varepsilon > 0$.) Thus, for an adequate polynomial t (i.e., $t(n+p(n)) = O(p(n)\log(1/\varepsilon))$), we obtain a relation $R_2^{(t)} \in \mathcal{PC}$ such that the following holds: *for every x and $y \in \{0,1\}^{p(|x|)}$, with probability at least $1 - \varepsilon \cdot 2^{-p(|x|)}$ over the random choice of $s' \in \{0,1\}^{\mathrm{poly}(|x|)}$, it holds that $x' \overset{\mathrm{def}}{=} (x,y) \in S'$ if and only if $|R_2^{(t)}(\langle x', s'\rangle)|$ is odd.*[4]

[4]Recall that $|s'| = t(|x'|) \cdot O(p'(|x'|))$, where $R'(x') \subseteq \{0,1\}^{p'(|x'|)}$ is the "witness-relation" corresponding to S' (i.e., $x' \in S'$ if and only if $R'(x') = \{0,1\}^{p'(|x'|)}$). Thus, $R_2(\langle x', s'\rangle) \subseteq$

Using a union bound (over all possible $y \in \{0,1\}^{p(|x|)}$), it follows that, *with probability at least* $1-\varepsilon$ *over the choice of* s'*, it holds that* $x \in S$ *if and only if there exists a* y *such that* $|R_2^{(t)}(\langle(x,y),s'\rangle)|$ *is odd.* Now, as in the treatment of $\mathcal{NP}$, we wish to reduce the latter "existential problem" to $\oplus\mathcal{P}$. That is, we wish to define a relation $R_3 \in \mathcal{PC}$ such that for a randomly selected s the value $|R_3(\langle x,s,s'\rangle)| \bmod 2$ provides an indication to whether or not $x \in S$ (by indicating whether or not there exists a y such that $|R_2^{(t)}(\langle(x,y),s'\rangle)|$ is odd). Analogously to Eq. (F.1), consider the binary relation

$$I_3 \stackrel{\text{def}}{=} \left\{(\langle x,s,s'\rangle,y) : |R_2^{(t)}(\langle(x,y),s'\rangle| \equiv 1 (\text{mod } 2) \ \wedge G(s,y)=1\right\}_. \tag{F.3}$$

In other words, $I_3(\langle x,s,s'\rangle) = \{y : |R_2^{(t)}(\langle(x,y),s'\rangle| \equiv 1(\text{mod } 2) \wedge G(s,y)=1\}$. Indeed, if $x \in S$ then, with probability at least $1-\varepsilon$ over the random choice of s' and probability at least $1/3$ over the random choice of s, it holds that $|I_3(\langle x,s,s'\rangle)|$ is odd, whereas for every $x \notin S$ and every choice of s it holds that $\mathsf{Pr}_{s'}[|I_3(\langle x,s,s'\rangle)| = 0] \geq 1-\varepsilon$.[5] Note that, for $\varepsilon \leq 1/7$, it follows that for every $x \in S$ we have $\mathsf{Pr}_{s,s'}[|I_3(\langle x,s,s'\rangle)| \equiv 1 \pmod 2] \geq (1-\varepsilon)/3 \geq 2/7$, whereas for every $x \notin S$ we have $\mathsf{Pr}_{s,s'}[|I_3(\langle x,s,s'\rangle)| \equiv 1 \pmod 2] \leq \varepsilon \leq 1/7$. Thus, $|I_3(\langle x,\cdot,\cdot\rangle)| \bmod 2$ provides a randomized indication to whether or not $x \in S$, but it is not clear whether I_3 is in $\mathcal{PC}$ (and in fact I_3 is likely not to be in $\mathcal{PC}$). The key observation is that there exists $R_3 \in \mathcal{PC}$ such that $\oplus R_3 = \oplus I_3$. Specifically, consider

$$R_3 \stackrel{\text{def}}{=} \left\{(\langle x,s,s'\rangle,\langle y,z\rangle) : (\langle(x,y),s'\rangle,z) \in R_2^{(t)} \wedge G(s,y)=1\right\}_, \tag{F.4}$$

where $\langle y,z\rangle \in \{0,1\}^{p(|x|)} \times \{0,1\}^{\text{poly}(|x|)}$. (That is, $\langle y,z\rangle$ is in $R_3(\langle x,s,s'\rangle)$ if $(\langle(x,y),s'\rangle,z) \in R_2^{(t)}$ and $G(s,y)=1$.) Clearly $R_3 \in \mathcal{PC}$, and so it is left to show that $|R_3(\langle x,s,s'\rangle)| \equiv |I_3(\langle x,s,s'\rangle)| \pmod 2$. The claim follows by letting $\chi_{y,z}$ (resp., ξ_y) indicate the event $(\langle(x,y),s'\rangle,z) \in R_2^{(t)}$ (resp., the event $G(s,y)=1$), noting that

$$\begin{aligned} |R_3(\langle x,s,s'\rangle)| \bmod 2 &\equiv \oplus_{y,z}(\chi_{y,z} \wedge \xi_y) \\ |I_3(\langle x,s,s'\rangle)| \bmod 2 &\equiv \oplus_y((\oplus_z \chi_{y,z}) \wedge \xi_y) \end{aligned}$$

$\{0,1\}^{p'(|x'|)+1}$ and $R_2^{(t)}(\langle x',s'\rangle)$ is a subset of $\{0,1\}^{1+t(|x'|)\cdot(p'(|x'|)+2)}$. Note that (since we started with $S' \in \text{co}\mathcal{NP}$) the error probability occurs on no-instances of S', whereas yes-instances are always accepted. However, to simplify the exposition, we allow possible errors also on yes-instances of S'. This does not matter because we will anyhow have an error probability on yes-instances of S (see Footnote 5).

[5]In continuation to Footnote 4, we note that actually, if $x \in S$ then there exists a y such that $(x,y) \in S'$ and consequently for every choice of s' it holds that $|R_2^{(t)}(\langle(x,y),s'\rangle)|$ is odd (because the reduction from $S' \in \text{co}\mathcal{NP}$ to $\oplus\mathcal{P}$ has zero error on yes-instances). Thus, for every $x \in S$ and s', with probability at least $1/3$ over the random choice of s, it holds that $|I_3(\langle x,s,s'\rangle)|$ is odd (because the reduction from $S \in \mathcal{NP}^{S'}$ to $\oplus\mathcal{P}^{S'}$ has non-zero error on yes-instances). On the other hand, if $x \notin S$ then $\mathsf{Pr}_{s'}[(\forall y)\,|R_2^{(t)}(\langle(x,y),s'\rangle)| \equiv 0 \pmod 2] \geq 1-\varepsilon$ (because for every y it holds that $(x,y) \notin S'$ and the reduction from $\text{co}\mathcal{NP}$ to $\oplus\mathcal{P}$ has non-zero error on no-instances). Thus, for every $x \notin S$ and s, it holds that $\mathsf{Pr}_{s'}[|I_3(\langle x,s,s'\rangle)| = 0] \geq 1-\varepsilon$ (because the reduction from $S \in \mathcal{NP}^{S'}$ to $\oplus\mathcal{P}^{S'}$ has zero error on no-instances). To sum-up, the combined reduction has two-sided error, because each of the two reductions introduces an error in a different direction.

and using the equivalence of the two corresponding Boolean expressions. Thus, S is randomly Karp-reducible to $\oplus R_3 \in \oplus\mathcal{P}$ (by the many-to-one randomized mapping of x to $\langle x, s, s'\rangle$, where (s, s') is uniformly selected in $\{0,1\}^{O(p(|x|))} \times \{0,1\}^{\mathrm{poly}(|x|)}$). Since this holds for any $S \in \Sigma_2$, we conclude that Σ_2 is randomly Karp-reducible to $\oplus\mathcal{P}$.

Again, error-reduction may be applied to this reduction (of Σ_2 to $\oplus\mathcal{P}$) such that the resulting reduction can be used for dealing with Σ_3 (viewed as $\mathcal{NP}^{\Sigma_2}$). A technical difficulty arises since the foregoing reduction has two-sided error probability, where one type (or "side") of error is due to the error in the reduction of $S' \in \mathrm{co}\mathcal{NP}$ to $\oplus R_2^{(t)}$ (which occurs on no-instances of S') and the second type (or "side") of error is due to the (new) reduction of S to $\oplus R_3$ (and occurs on the yes-instances of S). However, the error probability in the first reduction is (or can be made) very small and thus can be ignored when applying error-reduction to the second reduction. See following comments.

The general case. First note that, as in the case of $\mathrm{co}\mathcal{NP}$, we can obtain a similar reduction (to $\oplus\mathcal{P}$) for sets in $\Pi_2 = \mathrm{co}\Sigma_2$. It remains to extend the treatment of Σ_2 to Σ_k, for every $k \geq 2$. Indeed, we show how to reduce Σ_k to $\oplus\mathcal{P}$ by using a reduction of Σ_{k-1} (or rather Π_{k-1}) to $\oplus\mathcal{P}$. Specifically, $S \in \Sigma_k$ is treated by considering a polynomial p and a set $S' \in \Pi_{k-1}$ such that $S = \{x : \exists y \in \{0,1\}^{p(|x|)} \text{ s.t. } (x,y) \in S'\}$. Relying on the treatment of Π_{k-1}, we use a relation $R_k^{(t_k)}$ such that, with overwhelmingly high probability over the choice of s', the value $|R_k^{(t_k)}(\langle(x,y),s'\rangle)| \bmod 2$ indicates whether or not $(x,y) \in S'$. Using the ideas underlies the treatment of $\mathcal{NP}$ (and Σ_2) we check whether there exists $y \in \{0,1\}^{p(|x|)}$ such that $|R_k^{(t_k)}(\langle(x,y),s'\rangle)| \equiv 1 \pmod 2$. This yields a relation R_{k+1} such that for random s, s' the value $|R_{k+1}(\langle x, s, s'\rangle)| \bmod 2$ indicates whether or not $x \in S$. Finally, we apply error reduction, while ignoring the probability that s' is bad, and obtain the desired relation $R_{k+1}^{(t_{k+1})}$.

We comment that the foregoing inductive process should be implemented with some care. Specifically, if we wish to upper-bound the error probability in the reduction (of S) to $\oplus R_{k+1}^{(t_{k+1})}$ by ε_{k+1}, then the error probability in the reduction (of S') to $\oplus R_k^{(t_k)}$ should be upper-bounded by $\varepsilon_k \leq \varepsilon_{k+1} \cdot 2^{-p(|x|)}$ (and t_k should be set accordingly). Thus, the proof that $\mathcal{PH}$ is randomly reducible to $\oplus\mathcal{P}$ actually proceed "top down" (at least partially); that is, starting with an arbitrary $S \in \Sigma_k$, we first determine the auxiliary sets (as per Proposition 3.9) as well as the error-bounds that should be proved for the reductions of these sets (which reside in lower levels of $\mathcal{PH}$), and only then we establish the existence of such reductions. Indeed, this latter (and main) step is done "bottom up" using the reduction (to $\oplus\mathcal{P}$) of the set in the i^{th} level when reducing (to $\oplus\mathcal{P}$) the set in the $i+1^{\text{st}}$ level.

F.2 Proving that $\mathcal{IP}(f) \subseteq \mathcal{AM}(O(f)) \subseteq \mathcal{AM}(f)$

Using the notations presented in §9.1.4.3, we restate two results mentioned there.

Theorem F.2 (round-efficient emulation of $\mathcal{IP}$ by $\mathcal{AM}$): *Let $f : \mathbb{N} \to \mathbb{N}$ be a polynomially bounded function. Then $\mathcal{IP}(f) \subseteq \mathcal{AM}(f+3)$.*

We comment that, in light of the following linear speed-up in round-complexity for $\mathcal{AM}$, it suffices to establish $\mathcal{IP}(f) \subseteq \mathcal{AM}(O(f))$.

Theorem F.3 (linear speed-up for $\mathcal{AM}$): *Let $f : \mathbb{N} \to \mathbb{N}$ be a polynomially bounded function. Then $\mathcal{AM}(2f) \subseteq \mathcal{AM}(f+1)$.*

Combining these two theorems, we obtain a linear speed-up for $\mathcal{IP}$; that is, for any polynomially bounded $f : \mathbb{N} \to (\mathbb{N} \setminus \{1\})$, it holds that $\mathcal{IP}(O(f)) \subseteq \mathcal{AM}(f) \subseteq \mathcal{IP}(f)$. In this appendix we prove both theorems.

Note: The proof of Theorem F.2 relies on the fact that, for every f, error-reduction is possible for $\mathcal{IP}(f)$. Specifically, error-reduction can be obtained via *parallel* repetitions (see [90, Apdx. C.1]). We mention that error-reduction (in the context of $\mathcal{AM}(f)$) is implicit also in the proof of Theorem F.3 (and is explicit in the original proof of [23]).

F.2.1 Emulating general interactive proofs by AM-games

In this section we prove Theorem F.2. Our proof differs from the original proof of Goldwasser and Sipser [111] only in the conceptualization and implementation of the iterative emulation process.

F.2.1.1 Overview

Our aim is to transform a general interactive proof system (P, V) into a public-coin interactive proof system for the same set. Suppose, without loss of generality, that P constitutes an optimal prover with respect to V (i.e., P maximizes the acceptance probability of V on any input). Then, for any yes-instance x, the set A_x of coin sequences that make V accept when interacting with this optimal prover contains all possible outcomes, whereas for a no-instance x (of equal length) the set A_x is significantly smaller. The basic idea is having a public-coin system in which, on common input x, the prover proves to the verifier that the said set A_x is big. Such a proof system can be constructed using ideas as in the case of approximate counting (see the proof of Theorem 6.27), while replacing the NP-oracle with a prover that is required to prove the correctness of its answers. Implementing this idea requires taking a closer look at the set of coin sequences that make V accept an input.

A very restricted case. Let us first demonstrate the implementation of the foregoing approach by considering a restricted type of two-message interactive proof systems. Recall that in a two-message interactive proof system the verifier, denoted V, sends a single message (based on the common input and its internal coin tosses) to which the prover, denoted P, responds with a single message and then V decides whether to accept or reject the input. We further restrict our attention by assuming that *each possible message of V is equally likely and that the number of possible V-messages is easy to determine from the input.* Thus, on input x, the verifier V tosses $\ell = \ell(|x|)$ coins and sends one out of $N = N(x)$ possible messages. Note that if x is a yes-instance then for each possible V-message there exists a P-response that is accepted by the $2^\ell/N$ corresponding coin sequences of V (i.e., the coin sequences that lead V to send this V-message). On the other hand, if x is a no-instance then, in expectation, for a uniformly selected V-message, the optimal P-response is accepted by a significantly smaller number of corresponding coin sequences. We now show how such an interactive proof system can be emulated by a *public-coin system.*

In the *public-coin system*, on input x, the prover will attempt to prove that for each possible V-message (in the original system) there exists a response (by the original prover) that is accepted by $2^\ell/N$ corresponding coin sequences of V. Recall that $N = N(x)$ and $\ell = \ell(|x|)$ are easily determined by both parties, and so if the foregoing claim holds then x must be a yes-instance. The new interaction itself proceeds as follows: First, the verifier selects uniformly a coin sequence for V, denoted r, and sends it to the prover. The coin sequence r determines a V-message, denoted α. Next, the prover sends back an adequate P-message, denoted β, and *interactively proves to the verifier* that β would have been accepted by $2^\ell/N$ possible coin sequences of V that correspond to the V-message α (i.e., β should be accepted not only by r but rather by the $2^\ell/N$ coin sequences of V that correspond to the V-message α). The latter interactive proof follows the idea of the proof of Theorem 6.27: The verifier applies a random sieve that lets only a $(2^\ell/N)^{-1}$ fraction of the elements pass, and the prover shows that some adequate sequence of V-coins has passed this sieve (by merely presenting such a sequence).[6] We stress that the foregoing interaction (and in particular the random sieve) can be implemented in the public-coin model.

Waiving one restriction. Next, we waive the restriction that the number of possible V-messages is easy to determine from the input, but still assume that all possible V-messages are equally likely. In this case, the prover should provide the number N of possible V-messages and should prove that indeed there exist at least N possible V-messages (and that, as in the prior case, for each V-message there exists a P-response that is accepted by $2^\ell/N$ corresponding coin sequences of V). That is, the prover should prove that for at least N possible V-messages there exists a P-response that is accepted by $2^\ell/N$ corresponding coin sequences of V. This calls for a double (or rather nested) application of the aforementioned "lower-bound" protocol. That is, first the parties apply a random sieve to the set of possible V-messages such that only a N^{-1} fraction of these messages pass, and next the parties apply a random sieve to the set coin sequences that fit a passing V-message such that only a $(2^\ell/N)^{-1}$ fraction of these sequences pass.

[6] Indeed, the verifier can easily check whether a coin sequence r' passes the sieve as well as fits the initial message α and would have made V accept when the prover responds with β (i.e., V would have accepted the input, on coins r', when receiving the prover message β).

The general case of $\mathcal{IP}(2)$. Treating general two-message interactive proofs requires waiving also the restriction that all possible V-messages are equally likely. In this case, the prover may cluster the V-messages into few (say ℓ) clusters such that the messages in each cluster are sent (by V) with roughly the same probability (say, up to a factor of two). Then, focusing on the cluster having the largest probability weight, the prover can proceed as in the previous case (i.e., send i and claim that there are $2^\ell/\ell$ possible V-messages that are each supported by 2^i coin sequences). This has a potential of cutting the probabilistic gap between yes-instances and no-instances by a factor related to the number of clusters times the approximation level within clusters (e.g., a factor of $O(\ell)$)[7], but this loss is negligible in comparison to the initial gap (which can be obtained via error-reduction).

Dealing with all levels of $\mathcal{IP}$. So far, we only dealt with two-message systems (i.e., $\mathcal{IP}(2)$). We shall see that the general case of $\mathcal{IP}(f)$ can be dealt by recursion (or rather by iterations), where each level of recursion (resp., each iteration) is analogous to the (general) case of $\mathcal{IP}(2)$. Recall that our treatment of the case of $\mathcal{IP}(2)$ boils down to selecting a random V-message, α, and having the prover send a P-response, β, and prove that β is acceptable by many V-coins. In other words, the prover should prove that in the conditional probability space defined by a V-message α, the original verifier V accepts with high probability. In the general case (of $\mathcal{IP}(f)$), the latter claim refers to the probability of accepting in the residual interaction, which consists of $f-2$ messages, and thus the very same protocol can be applied iteratively (until we get to the last message, which is dealt as in the case of $\mathcal{IP}(2)$). The only problem is that, in the residual interactions, it may not be easy for the verifier to select a random V-message (as done in the *very restricted case*). However, as already done when *waiving the first restriction*, the the verifier can be assisted by the prover, while making sure that it is not being fooled by the prover. This process is made explicit in §F.2.1.2, where we define an adequate notion of a "random selection" protocol (which need to be implemented in the public-coin model). For simplicity, we may consider the problem of uniformly selecting a sequence of coins in the corresponding (residual) probability space, because such a sequence determines the desired random V-message.

F.2.1.2 Random selection

Various types of "random selection" protocols have appeared in the literature (see, e.g., [227, Sec. 6.4]). The common theme in these protocols is that they allow for a probabilistic polynomial-time player (called the *verifier*) to sample a set, denoted $S \subset \{0,1\}^\ell$, while being assisted by a second player (called the *prover*) that is powerful but not trustworthy. These nicknames fit the common conventions regarding interactive proofs and are further justified by the typical applications of such protocols as subroutines within an interactive proof system (where indeed the

[7]The loss is due to the fact that the distribution of (probability) weights may not be identical on all instances. For example, in one case (e.g., of some yes-instance) all clusters may have equal weight, and thus a corresponding factor is lost, while in another case (e.g., of some no-instance) all the probability mass may be concentrated in a single cluster.

first party is played by the higher-level verifier while the second party is played by the higher-level prover). The various types of random selection protocols differ by what is known about the set S and what is required from the protocol.

Here we will assume that the verifier is given a parameter N, which is supposed to equal $|S|$, and the performance guarantee of the protocol will be meaningful only for sets of size at most N. We seek a constant-round (preferably two-message) public-coin protocol (for this setting) such that the following two conditions hold, with respect to a security parameter $\varepsilon \geq 1/\text{poly}(\ell)$.

1. If both players follow the protocol and $N = |S|$ then the verifier's output is ε-close to the uniform distribution over S. Furthermore, the verifier always outputs an element of S.

2. For any set $S' \subseteq \{0,1\}^\ell$ if the verifier follows the protocol then, no matter how the prover behaves, the verifier's output resides in S' with probability at most $\text{poly}(\ell/\varepsilon) \cdot (|S'|/N)$.

Indeed, the second property is meaningful only for sets S' having size that is (significantly) smaller than N. We shall be using such a protocol while setting ε to be a constant (say, $\varepsilon = 1/2$).

A three-message public-coin protocol that satisfies the foregoing properties can be obtained by using the ideas that underly Construction 6.32. Specifically, we set $m = \max(0, \log_2 N - O(\log \ell/\varepsilon))$ in order to guarantee that if $|S| = N$ then, with overwhelmingly high probability, each of the 2^m cells defined by a uniformly selected hashing function contains $(1 \pm \varepsilon) \cdot |S|/2^m$ elements of S. In the protocol, *the prover arbitrarily selects a good hashing function* (i.e., one defining such a good partition of S) *and sends it to the verifier, which answers with a uniformly selected cell, to which the prover responds with a uniformly selected element of S that resides in this cell.*[8]

We stress that the foregoing protocol is indeed in the public-coin model, and comment that the fact that it uses three messages rather than two will have a minor effect on our application (see §F.2.1.3). Indeed, this protocol satisfies the two foregoing properties. In particular, the second property follows because for every possible hashing function, the fraction of cells containing an element of S' is at most $|S'|/2^m$, which is upper-bounded by $\text{poly}(\ell/\varepsilon) \cdot |S'|/N$.

[8]We mention that the foregoing protocol is but one out of several possible implementations of the ideas that underly Construction 6.32. Firstly, note that an alternative implementation may designate the task of selecting a hashing function to the verifier, who may do so by selecting a function at random. Although this seems more natural, it actually offers no advantage with respect to the "soundness-like" property (i.e., the second property). Furthermore, in this case, it may happen (rarely) that the hashing function selected by the verifier is not good, and consequently the furthermore clause of the first property (i.e., requiring that the output always resides in S) is not satisfied. Secondly, recall that in the foregoing protocol the last step consists of the prover selecting a random element of S that resides in the selected (by the verifier) cell. An alternative implementation may replace this step by two steps such that first the prover sends a list of $(1-\varepsilon) \cdot N/2^m$ elements (of S) that resides in the said cell, and then the verifier outputs a uniformly selected element of this list. This alternative yields an improvement in the "soundness-like" property (i.e., the verifier's output resides in S' with probability at most $(|S'|/N) + \varepsilon$), but requires an additional message (which we prefer to avoid, although this not that crucial).

F.2.1.3 The iterated partition protocol

Using the random selection protocol of §F.2.1.2, we now present a public-coin emulation of an arbitrary interactive proof system, (P,V). We start with some notations.

Fixing any input x to (P,V), we denote by $t = t(|x|)$ the number of pairs of messages exchanged in the corresponding interaction, while assuming that the verifier takes the first move in (P,V).[9] We denote by $\ell = \ell(|x|)$ the number of coins tossed by V, and assume that $\ell > t$. Recall that we assume that P is an optimal prover (with respect to V), and that (without loss of generality) P is deterministic. Let us denote by $\langle P, V(r)\rangle(x)$ the full transcript of the interaction of P and V on input x, when V uses coins r; that is, $\langle P, V(r)\rangle(x) = (\alpha_1, \beta_1, ..., \alpha_t, \beta_t, \sigma)$ if $\sigma = V(x, r, \beta_1, ..., \beta_t) \in \{0,1\}$ is V's final verdict and for every $i = 1, ..., t$ it holds that $\alpha_i = V(x, r, \beta_1, ..., \beta_{i-1})$ and $\beta_i = P(x, \alpha_1, ..., \alpha_i)$. For any partial transcript ending with a P-message, $\gamma = (\alpha_1, \beta_1, ..., \alpha_{i-1}, \beta_{i-1})$, we denote by $\mathrm{ACC}_x(\gamma)$ the set of coin sequences that are consistent with the partial transcript γ and lead V to accept x when interacting with P; that is, $r \in \mathrm{ACC}_x(\gamma)$ if and only if for some $\gamma' \in \{0,1\}^{2(t-i)\cdot \mathrm{poly}(|x|)}$ it holds that $\langle P, V(r)\rangle(x) = (\alpha_1, \beta_1, ..., \alpha_{i-1}, \beta_{i-1}, \gamma', 1)$. The same notation is also used for a partial transcript ending with a V-message; that is, $r \in \mathrm{ACC}_x(\alpha_1, \beta_1, ..., \alpha_i)$ if and only if $\langle P, V(r)\rangle(x) = (\alpha_1, \beta_1, ..., \alpha_i, \gamma', 1)$ for some γ'.

Motivation. By suitable error reduction, we may assume that (P,V) has soundness error $\mu = \mu(|x|)$ that is smaller than $\mathrm{poly}(\ell)^{-t}$. Thus, for any yes-instance x it holds that $|\mathrm{ACC}_x(\lambda)| = 2^\ell$, whereas for any no-instance x it holds that $|\mathrm{ACC}_x(\lambda)| \leq \mu \cdot 2^\ell$. Indeed, the gap between the initial set sizes is huge, and we can maintain a gap between the sizes of the corresponding residual sets (i.e., $\mathrm{ACC}_x(\alpha_1, \beta_1, ..., \alpha_i)$) provided that we lose at most a factor of $\mathrm{poly}(\ell)$ per each round. The key observations is that, for any partial transcript $\gamma = (\alpha_1, \beta_1, ..., \alpha_{i-1}, \beta_{i-1})$, it holds that

$$|\mathrm{ACC}_x(\gamma)| = \sum_\alpha |\mathrm{ACC}_x(\gamma, \alpha)|, \tag{F.5}$$

whereas $|\mathrm{ACC}_x(\gamma, \alpha)| = \max_\beta\{|\mathrm{ACC}_x(\gamma, \alpha, \beta)|\}$. Clearly, we can prove that $|\mathrm{ACC}_x(\gamma, \alpha)|$ is big by providing an adequate β and proving that $|\mathrm{ACC}_x(\gamma, \alpha, \beta)|$ is big. Likewise, proving that $|\mathrm{ACC}_x(\gamma)|$ is big reduces to proving that the sum $\sum_\alpha |\mathrm{ACC}_x(\gamma, \alpha)|$ is big. The problem is that this sum may contain exponentially many terms, and so we cannot even afford reading the value of each of these terms.[10] As hinted in §F.2.1.1, we may cluster these terms into ℓ clusters, such that the j^{th} cluster contains sets of cardinality approximately 2^j (i.e., α's such that $2^j \leq |\mathrm{ACC}_x(\gamma, \alpha)| < 2^{j+1}$). One of these clusters must account for a $1/2\ell$ fraction of the claimed size of $|\mathrm{ACC}_x(\gamma)|$,

[9] We note if the prover takes the first move in (P,V) then its first message can be emulated with no cost (in the number of rounds).

[10] Furthermore, we cannot afford verifying more than a single claim regarding the value of one of these terms, because examining at least two values per round will yield an exponential blow-up (i.e., time complexity that is exponential in the number of rounds).

and so we focus on this cluster; that is, the prover we construct will identify a suitable j (i.e., such that there are at least $|\mathrm{ACC}_x(\gamma)|/2\ell$ elements in the sets of the j^{th} cluster), and prove that there are at least $N = |\mathrm{ACC}_x(\gamma)|/(2\ell \cdot 2^{j+1})$ sets (i.e., $\mathrm{ACC}_x(\gamma,\alpha)$'s) each of size at least 2^j. Note that this establishes that $|\mathrm{ACC}_x(\gamma)|$ is bigger than $N \cdot 2^j = |\mathrm{ACC}_x(\gamma)|/O(\ell)$, which means that we lost a factor of $O(\ell)$ of the size of $\mathrm{ACC}_x(\gamma)$. But as stated previously, we may afford such a lost.

Before we turn to the actual protocol, let us discuss the method of proving that that there are at least N sets (i.e., $\mathrm{ACC}_x(\gamma,\alpha)$'s) each of size at least 2^j. This claim is proved by employing the random selection protocol (while setting the size parameter to N) with the goal of selecting such a set (or rather its index α). If indeed N such sets exists then the first property of the protocol guarantees that such a set is always chosen, and we will proceed to the next iteration with this set, which has size at least 2^j (and so we should be able to establish a corresponding lower-bound there). Thus, entering the current iteration with a valid claim, we proceed to the next iteration with a new valid claim. On the other hand, suppose that $|\mathrm{ACC}_x(\gamma)| \ll N \cdot 2^j$. Then, the second property of the protocol implies[11] that, with probability at least $1-(1/3t)$, the selected α is such that $|\mathrm{ACC}_x(\gamma,\alpha)| < \mathrm{poly}(\ell)\cdot|\mathrm{ACC}_x(\gamma)|/N \ll 2^j$, whereas at the next iteration we will need to prove that the selected set has size at least 2^j. Thus, entering the current iteration with a false claim that is wrong by a factor $F \gg \mathrm{poly}(\ell)$, with probability at least $1-(1/3t)$, we proceed to the next iteration with a false claim that is wrong by a factor of at least $F/\mathrm{poly}(\ell)$.

We note that, although the foregoing motivational discussion refers to proving lower-bounds on various set sizes, the actual implementation refers to randomly selecting elements in such sets. If the sets are smaller than claimed, the selected elements are likely to reside outside these sets, which will be eventually detected.

Construction F.4 (the actual protocol). *On common input x, the $2t$-message interaction of P and V is* "quasi-emulated" *in t iterations, where $t = t(|x|)$. The i^{th} iteration starts with a partial transcript $\gamma_{i-1} = (\alpha_1,\beta_1,...,\alpha_{i-1},\beta_{i-1})$ and a claimed bound M_{i-1}, where in the first iteration γ_0 is the empty sequence and $M_0 = 2^\ell$. The i^{th} iteration proceeds as follows.*

1. *The prover determines an index j such that the cluster $C_j = \{\alpha : 2^j \leq |\mathrm{ACC}_x(\gamma_{i-1},\alpha)| < 2^{j+1}\}$ has size at least $N \overset{\text{def}}{=} M_{i-1}/(2^{j+2}\ell)$, and sends j to the verifier. Note that if $|\mathrm{ACC}_x(\gamma_{i-1})| \geq M_{i-1}$ then such a j exists.*

2. *The prover invokes the random selection protocol with size parameter N in order to select $\alpha \in C_j$, where for simplicity we assume that $C_j \subseteq \{0,1\}^\ell$. Recall that this public-coin protocol involves three messages with the first and last message being sent by the prover. Let us denote the outcome of this protocol by α_i.*

[11]For a loss factor $L = \mathrm{poly}(\ell)$, consider the set $S' = \{\alpha : |\mathrm{ACC}_x(\gamma,\alpha)| \geq L \cdot |\mathrm{ACC}_x(\gamma)|/N\}$. Then $|S'| \leq N/L$, and it follows that an element in S' is selected with probability at most $\mathrm{poly}(\ell)/L$, which is upper-bounded by $1/3t$ when using a suitable choice of L.

3. *The prover determines β_i such that $\mathrm{ACC}_x(\gamma_{i-1},\alpha_i,\beta_i) = \mathrm{ACC}_x(\gamma_{i-1},\alpha_i)$ and sends β_i to the verifier.*

   Towards the next iteration $M_i \leftarrow 2^j$ and $\gamma_i = (\alpha_1,\beta_1,...,\alpha_i,\beta_i) \equiv (\gamma_{i-1},\alpha_i,\beta_i)$.

After the last iteration,[12] *the prover invokes the random selection protocol with size parameter $N = M_t$ in order to select $r \in \mathrm{ACC}_x(\alpha_1,\beta_1,...,\alpha_t,\beta_t)$. Upon obtaining this r, the verifier accepts if and only if $V(x,r,\beta_1,...,\beta_t) = 1$ and for every $i = 1,...,t$ it holds that $\alpha_i = V(x,r,\beta_1,...,\beta_{i-1})$, where the α_i's and β_i's are as determined in the foregoing iterations.*

Note that the three steps of each iteration involve a single message by the (public-coin) verifier, and thus the foregoing protocol can be implemented using $2t+3$ messages.

Clearly, if x is a yes-instance then the prover can make the verifier accept with probability one (because an adequately large cluster exists at each iteration, and the random selection protocol guarantees that the selected α_i will reside in this cluster).[13] On the other hand, if x is a no-instance then by using the low soundness error of (P,V) we can establish the soundness of Construction F.4. This is proved in the *following claim, which refers to a polynomial p that is sufficiently large.*

Proposition F.5 *Suppose that $|\mathrm{ACC}_x(\lambda)| < \delta^{t+1}\cdot 2^\ell$, where $\delta = 1/p(\ell)$. Then, the verifier of Construction F.4 accepts x with probability smaller than $1/2$.*

Proof Sketch: We first prove that, for every $i = 1,...,t$, if $|\mathrm{ACC}_x(\gamma_{i-1})| < \delta^{t+1-(i-1)}\cdot M_{i-1}$ then, with probability at least $1-(1/3t)$, it holds that $|\mathrm{ACC}_x(\gamma_i)| < \delta^{t+1-i}\cdot M_i$. Fixing any i, let j be the value selected by the prover in Step 1 of iteration i, and define $S' = \{\alpha : |\mathrm{ACC}_x(\gamma_{i-1},\alpha)| \geq \delta^{t+1-i}\cdot 2^j\}$. Then

$$|S'|\cdot\delta^{t+1-i}2^j \leq |\mathrm{ACC}_x(\gamma_{i-1})| < \delta^{t+1-(i-1)}\cdot M_{i-1},$$

where the second inequality represents the claim's hypothesis. Letting $N = M_{i-1}/(2^{j+2}\ell)$ (as in Step 1 of this iteration), it follows that $|S'| < 4\ell\delta\cdot N$. By the second property of the random selection protocol (invoked in Step 2 of this iteration with size parameter N), it follows that

$$\mathsf{Pr}[\alpha_i \in S'] \leq \mathrm{poly}(\ell)\cdot\frac{|S'|}{N} \leq \mathrm{poly}(\ell)\cdot\delta,$$

which is smaller than $1/3t$ (provided that the polynomial p that determines $\delta = 1/p(\ell)$ is sufficiently large). Thus, with probability at least $1-(1/3t)$, it holds that $|\mathrm{ACC}_x(\gamma_{i-1},\alpha_i)| < \delta^{t+1-i}\cdot 2^j$. The claim regarding $|\mathrm{ACC}_x(\gamma_i)|$ follows by recalling that $M_i = 2^j$ (in Step 3) and that for every β it holds that $|\mathrm{ACC}_x(\gamma_{i-1},\alpha_i,\beta)| \leq |\mathrm{ACC}_x(\gamma_{i-1},\alpha_i)|$.

[12]Alternatively, we may modify (P,V) by adding a last V-message in which V sends its internal coin tosses (i.e., r). In this case, the additional invocation of the random selection protocol occurs as a special case of handling the added $t+1^{\mathrm{st}}$ iteration.

[13]Thus, at the last invocation of the random selection protocol, the verifier always obtains $r \in \mathrm{ACC}_x(\gamma_t)$ and accepts.

Using the hypothesis $|\mathrm{ACC}_x(\gamma_0)| < \delta^{t+1} \cdot M_0$ and the foregoing claim, it follows that, with probability at least 2/3, the execution of the aforementioned t iterations yields values γ_t and M_t such that $|\mathrm{ACC}_x(\gamma_t)| < \delta \cdot M_t$. In this case, the last invocation of the random selection protocol (invoked with size parameter M_t) produces an element of $\mathrm{ACC}_x(\gamma_t)$ with probability at most $\mathrm{poly}(\ell) \cdot \delta < 1/6$, and otherwise the verifier rejects (because the conditions that the verifier checks regarding the output r of the random selection protocol are logically equivalent to $r \in \mathrm{ACC}_x(\gamma_t)$). The proposition follows. $\square$

F.2.2 Linear speed-up for $\mathcal{AM}$

In this section we prove Theorem F.3. Our proof differs from the original proof of Babai and Moran [23] in the way we analyze the basic switch (of MA to AM).

We adopt the standard terminology of public-coin (a.k.a Arthur-Merlin) interactive proof systems, where the verifier is called Arthur and the prover is called Merlin. More importantly, we view the *execution* of such a proof system, on any fixed common input x, as a (full-information) game (indexed by x) between an honest Arthur and powerful Merlin. These parties alternate in taking moves such that Arthur takes random moves and Merlin takes optimal moves with respect to a fixed (polynomial-time computable) predicate v_x that is *evaluated on the full transcript of the game's execution.* We stress that (in contrast to general interactive proof systems), each of Arthur's moves is uniformly distributed in a set of possible values that is predetermined independently of prior moves (e.g., the set $\{0,1\}^{\ell(|x|)}$). The value of the game is defined as the expected value of an execution of the game, where the expectation is taken over Arthur's moves (and Merlin's moves are assumed to be optimal).

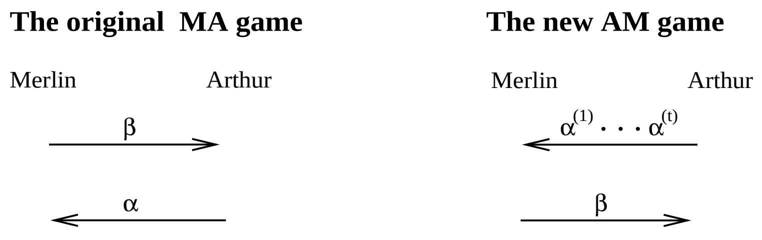
We shall assume, without loss of generality, that all messages of Arthur are of the same length, denoted $\ell = \ell(|x|)$. Similarly, each of Merlin's messages is of length $m = m(|x|)$.

Recall that $\mathcal{AM} = \mathcal{AM}(2)$ denotes a two-message system in which Arthur moves first and does not toss coins after receiving Merlin's answer, whereas $\mathcal{MA} = \mathcal{AM}(1)$ denotes a one-message system in which Merlin sends a single message and Arthur tosses additional coins after receiving this message. Thus, both $\mathcal{AM}$ and $\mathcal{MA}$ are viewed as two-move games, and differ in the order in which the two parties take these moves. As we shall shortly see (in §F.2.2.1), the "MA order" can be emulated by the "AM order" (i.e., $\mathcal{MA} \subseteq \mathcal{AM}$). This fact will be the basis of the "round speed-up" transformation (presented in §F.2.2.2).

F.2.2.1 The basic switch (from MA to AM)

The basic idea is transforming an MA-game (i.e., a two-move game in which Merlin moves first and Arthur follows) into an AM-game (in which Arthur moves first and Merlin follows). In the original game (on input x), first Merlin sends a message $\beta \in \{0,1\}^m$, then Arthur responds with a random $\alpha \in \{0,1\}^\ell$, and Arthur's verdict (i.e., the value of this execution of the game) is given by $v_x(\beta, \alpha) \in \{0,1\}$. In the new game (see Figure F.1), the order of these moves will be switched, but to limit

Merlin's potential gain from the switch we require it to provide a single answer that should "fit" several random messages of Arthur. That is, for a parameter t to be specified, first Arthur send a random sequence $(\alpha^{(1)},...,\alpha^{(t)}) \in \{0,1\}^{t\cdot\ell}$, then Merlin responds with a string $\beta \in \{0,1\}^m$, and Arthur accepts if and only if for every $i \in \{1,...t\}$ it holds that $v_x(\beta,\alpha^{(i)}) = 1$ (i.e., the value of this transcript of the new game is defined as $\prod_{i=1}^t v_x(\beta,\alpha^{(i)})$). Intuitively, Merlin gets the advantage of choosing its move after seeing Arthur's move(s), but Merlin's choice must fit the t choices of Arthur's move, which leaves Merlin with little gain (if t is sufficiently large).



The value of the transcript (β,α) of the original MA-game is given by $v_x(\beta,\alpha)$, whereas the value of the transcript $((\alpha^{(1)},...,\alpha^{(t)}),\beta)$ of the new AM-game is given by $\prod_{i=1}^t v_x(\beta,\alpha^{(i)})$.

Figure F.1: The transformation of an MA-game into an AM-game.

Recall that the value, v'_x, of the transcript $(\overline{\alpha},\beta)$ of the new game, where $\overline{\alpha} = (\alpha^{(1)},...,\alpha^{(t)})$, is defined as $\prod_{i=1}^t v_x(\beta,\alpha^{(i)})$. Thus, the value of the new game is defined as

$$\mathsf{E}_{\overline{\alpha}}\left[\max_{\beta}\left\{\prod_{i=1}^{t} v_x(\beta,\alpha^{(i)})\right\}\right], \tag{F.6}$$

which is upper-bounded by

$$\mathsf{E}_{\overline{\alpha}}\left[\max_{\beta}\left\{\frac{1}{t}\sum_{i=1}^{t} v_x(\beta,\alpha^{(i)})\right\}\right]. \tag{F.7}$$

Note that the upper-bound provided in Eq. (F.7) is tight in the case that the value of the original MA-game equals one (i.e., if x is a yes-instance), and that in this case the value of the new game is one (because in this case there exists a move β such that $v_x(\beta,\alpha) = 1$ holds for every α). However, the interesting case, where Merlin may gain something by the switch, is when the value of the original MA-game is strictly smaller than one (i.e., when x is a no-instance). The main observation is that, for a suitable choice of t, *it is highly improbable that Merlin's gain from the switch is significant.*

Recall that in the original MA-game Merlin selects β obliviously of Arthur's choice of α, and thus Merlin's "profit" (i.e., the value of the game) is represented by $\max_\beta\{\mathsf{E}_\alpha(v_x(\beta,\alpha))\}$. In the new AM-game, Merlin selects β based on the sequence $\overline{\alpha}$ chosen by Arthur, and we have upper-bounded its "profit" (in the new AM-game) by Eq. (F.7). Merlin's gain from the switch is thus the excess profit (of the new AM-game as compared to the original MA-game). We upper-bound the probability that *Merlin's gain from the switch exceeds a parameter*, denoted δ, as follows.

$$\begin{aligned} p_{x,\delta} &\stackrel{\text{def}}{=} \mathsf{Pr}_{(\alpha^{(1)},\dots,\alpha^{(t)})}\left[\max_{\beta}\left\{\frac{1}{t}\cdot\sum_{i=1}^{t} v_x(\beta,\alpha^{(i)})\right\} > \max_{\beta}\{\mathsf{E}_\alpha(v_x(\beta,\alpha))\}+\delta\right] \\ &\leq \mathsf{Pr}_{(\alpha^{(1)},\dots,\alpha^{(t)})}\left[\exists\beta\in\{0,1\}^m \text{ s.t. } \left|\frac{1}{t}\cdot\sum_{i=1}^{t} v_x(\beta,\alpha^{(i)}) - \mathsf{E}_\alpha(v_x(\beta,\alpha))\right| > \delta\right] \\ &\leq 2^m\cdot\exp(-\Omega(\delta^2\cdot t)), \end{aligned}$$

where the last inequality is due to combining the Union Bound with the Chernoff Bound. Denoting by $V_x = \max_\beta\{\mathsf{E}_\alpha(v_x(\beta,\alpha))\}$ the value of the original game, we upper-bound Eq. (F.7) by $p_{x,\delta}+V_x+\delta$. Using $t = O((m+k)/\delta^2)$ we have $p_{x,\delta}\leq 2^{-k}$, and thus

$$V'_x \stackrel{\text{def}}{=} \mathsf{E}_{\overline{\alpha}}\left[\max_{\beta}\left\{\frac{1}{t}\sum_{i=1}^{t} v_x(\beta,\alpha^{(i)})\right\}\right] \leq \max_{\beta}\{\mathsf{E}_\alpha(v_x(\beta,\alpha))\}+\delta+2^{-k}. \quad \text{(F.8)}$$

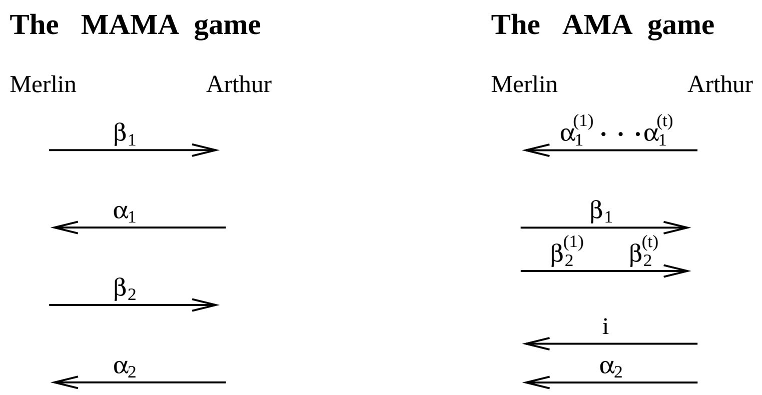
Needless to say, Eq. (F.7) is lower-bounded by V_x (since Merlin may just use the optimal move of the MA-game). In particular, using $\delta = 2^{-k} = 1/8$ and assuming that $V_x \leq 1/4$, we obtain $V'_x < 1/2$. Thus, starting from an MA proof system for some set, we obtain an AM proof system for the same set; that is, we just proved that $\mathcal{MA}\subseteq\mathcal{AM}$.

Extension. We note that the foregoing transformation as well as its analysis does not refer to the fact that $v_x(\beta,\alpha)$ is efficiently computable from (β,α). Furthermore, the analysis remain valid for arbitrary $v_x(\cdot,\cdot)\in[0,1]$, because for any $v_1,\dots,v_t\in[0,1]$ it holds that $\prod_{i=1}^t v_i \leq (\prod_{i=1}^t v_i)^{1/t} \leq \sum_{i=1}^t v_i/t$. Thus, we may apply the foregoing transformation to any two consecutive Merlin-Arthur moves in any public-coin interactive proof, provided that all the subsequent moves are performed in t copies, where each copy corresponds to a different $\alpha^{(i)}$ used in the switch. That is, if the j^{th} move is by Merlin then we can switch the players in the j and $j+1$ moves, by letting Arthur take the j^{th} move, sending $(\alpha^{(1)},\dots,\alpha^{(t)})$, followed by Merlin's move, answering β. Subsequent moves will be played in t copies such that the i^{th} copy corresponds to the moves $\alpha^{(i)}$ and β. The value of the new game may increase by at most $2^{-k}+\delta < 1/4$, and so we obtain an "equivalent" game with the two steps switched. Schematically, acting on the middle MA (indicated in bold font), we can replace $[\text{AM}]^{j_1}\text{A}\mathbf{MA}[\text{MA}]^{j_2}$ by $[\text{AM}]^{j_1}\text{A}\mathbf{AM}[\text{MA}]^{j_2}$, which in turn allows the collapse of two consecutive A-moves (and two consecutive M-moves if $j_2\geq 1$). In particular (using only the case $j_1 = 0$), we get $\text{A}[\text{MA}]^{j+1} = \text{A}[\text{MA}]^{j} = \cdots = \text{AMA} = \text{AM}$. Thus, for any constant f, we get $\mathcal{AM}(f) = \mathcal{AM}(2)$.

We stress that the foregoing switching process can be applied only a constant number of times, because each time we apply the switch the length of messages increases by a factor of $t = \Omega(m)$. Thus, a different approach is required to deal with a non-constant number of messages (i.e., unbounded function f).

F.2.2.2 The augmented switch (from $[MAMA]^j$ to $[AMA]^jA$)

Sequential applications of the "MA-to-AM switch" allows for reducing the number of rounds by any additive constant. However, each time this switch is applied, all subsequent moves are performed t times (in parallel). That is, the "MA-to-AM switch" splits the rest of the game to t independent copies, and thus this switch cannot be performed more than a constant number of times. Fortunately, Eq. (F.7) suggests a way of shrinking the game back to a single copy: just have Arthur select $i \in [t]$ uniformly and have the parties continue with the i^{th} copy.[14] In order to avoid introducing an Arthur-Merlin alternation, the extra move of Arthur is postpone to after the following move of Merlin (see Figure F.2). Schematically (indicating the action by bold font), we replace **MA**MA by **AM**MA**A**=**AMA** (rather than replacing **MA**MA by **AMA**MA and obtaining no reduction in the number of move-alternations).



The value of the transcript $(\beta_1, \alpha_1, \beta_2, \alpha_2)$ of the original MAMA-game is given by $v_x(\beta_1, \alpha_1, \beta_2, \alpha_2)$, whereas the value of the transcript $((\alpha_1^{(1)}, ..., \alpha_1^{(t)}), (\beta_1, \beta_2^{(1)}, ..., \beta_2^{(t)}), (i, \alpha_2))$ of the new AMA-game is given by $v_x(\beta_1, \alpha_1^{(i)}, \beta_2^{(i)}, \alpha_2)$.

Figure F.2: The transformation of MAMA into AMA.

The value of game obtained via the aforementioned augmented switch is given by Eq. (F.7), which can be written as

$$\mathsf{E}_{\alpha^{(1)},\dots,\alpha^{(t)}}[\max_{\beta}\{\mathsf{E}_{i\in[t]}(v_x(\beta, \alpha^{(i)}))\}],$$

[14]Indeed, the relaxed form of Eq. (F.7) plays a crucial role here (in contrast to Eq. (F.6)).

which in turn is upper-bounded (in Eq. (F.8)) by $\max_\beta\{\mathsf{E}_\alpha(v_x(\beta,\alpha))\} + \delta + 2^{-k}$. As in §F.2.2.1, the argument applies to any two consecutive Merlin-Arthur moves in any public-coin interactive proof. Recall that in order to avoid the introduction of an extra Arthur move, we actually postpone the last move of Arthur to after the next move of Merlin. Thus, we may apply the augmented switch to the first two moves in any block of four consecutive moves that start with a Merlin move, transforming the schematic sequence MAMA into AMMAA=AMA (see Figure F.2). The key point is that *the moves that take place after the said block*, remain intact. Hence, we may apply the augmented "MA-to-AM switch" (which is actually an "MAMA-to-AMA switch") concurrently to disjoint segments of the game. Schematically, we can replace $[\mathrm{MAMA}]^j$ by $[\mathrm{AMA}]^j = A[\mathrm{MA}]^j$. Note that Merlin's gain from each such switch is upper-bounded by $\delta + 2^{-k}$, but selecting $t = \widetilde{O}(f(|x|)^2 \cdot m(|x|)) = \mathrm{poly}(|x|)$ allows to upper-bound the total gain by a constant (using, say, $\delta = 2^{-k} = 1/8f(|x|)$). We thus obtain $\mathcal{AM}(4f) \subseteq \mathcal{AM}(2f+1)$, and Theorem F.3 follows.

Appendix G

Some Computational Problems

Although we view specific (natural) computational problems as secondary to (natural) complexity classes, we do use the former for clarification and illustration of the latter. This appendix provides definitions of such computational problems, grouped according to the type of objects to which they refer (e.g., graphs, Boolean formula, etc.).

We start by addressing the central issue of the representation of the various objects that are referred to in the aforementioned computational problems. The general principle is that elements of all sets are "compactly" represented as binary strings (without much redundancy). For example, the elements of a finite set S (e.g., the set of vertices in a graph or the set of variables appearing in a Boolean formula) will be represented as binary strings of length $\log_2 |S|$.

G.1 Graphs

Graph theory has long become recognized as one of the more useful mathematical subjects for the computer science student to master. The approach which is natural in computer science is the algorithmic one; our interest is not so much in existence proofs or enumeration techniques, as it is in finding efficient algorithms for solving relevant problems, or alternatively showing evidence that no such algorithms exist. Although algorithmic graph theory was started by Euler, if not earlier, its development in the last ten years has been dramatic and revolutionary.

Shimon Even, Graph Algorithms [71]

A simple graph $G=(V,E)$ consists of a *finite* set of vertices V and a finite set of edges E, where each edge is an *unordered pair* of vertices; that is, $E \subseteq \{\{u,v\}$:

$u,v\in V \wedge u\neq v\}$. This formalism does not allow self-loops and parallel edges, which are allowed in general (i.e., non-simple) graphs, where E is a multi-set that may contain (in addition to two-element subsets of V also) singletons (i.e., self-loops). The vertex u is called an end-point of the edge $\{u,v\}$, and the edge $\{u,v\}$ is said to be incident at v. In such a case we say that u and v are adjacent in the graph, and that u is a neighbor of v. The degree of a vertex in G is defined as the number of edges that are incident at this vertex.

We will consider various sub-structures of graphs, the simplest one being paths. A path in a graph $G=(V,E)$ is a sequence of vertices $(v_0,...,v_\ell)$ such that for every $i\in[\ell]\overset{\text{def}}{=}\{1,...,\ell\}$ it holds that v_{i-1} and v_i are adjacent in G. Such a path is said to have length ℓ. A simple path is a path in which each vertex appears at most once, which implies that the longest possible simple path in G has length $|V|-1$. The graph is called connected if there exists a path between each pair of vertices in it.

A cycle is a path in which the last vertex equals the first one (i.e., $v_\ell = v_0$). The cycle $(v_0,...,v_\ell)$ is called simple if $\ell > 2$ and $|\{v_0,...,v_\ell\}| = \ell$ (i.e., if $v_i = v_j$ then $i \equiv j \pmod{\ell}$, and the cycle (u,v,u) is not considered simple). A graph is called acyclic (or a forest) if it has no simple cycles, and if it is also connected then it is called a tree. Note that $G=(V,E)$ is a tree if and only if it is connected and $|E| = |V|-1$, and that there is a unique simple path between each pair of vertices in a tree.

A subgraph of the graph $G=(V,E)$ is any graph $G'=(V',E')$ satisfying $V'\subseteq V$ and $E'\subseteq E$. Note that a simple cycle in G is a connected subgraph of G in which each vertex has degree exactly two. An induced subgraph of the graph $G=(V,E)$ is any subgraph $G'=(V',E')$ that contain all edges of E that are contained in V'. In such a case, we say that G' is the subgraph induced by V'.

Directed graphs. We will also consider (simple) directed graphs (a.k.a digraphs), where edges are *ordered pairs* of vertices. In this case the set of edges is a subset of $V\times V\setminus\{(v,v) : v\in V\}$, and the edges (u,v) and (v,u) are called anti-parallel. General (i.e., non-simple) directed graphs are defined analogously. The edge (u,v) is viewed as going from u to v, and thus is called an outgoing edge of u (resp., incoming edge of v). The out-degree (resp., in-degree) of a vertex is the number of its outgoing edges (resp., incoming edges). Directed paths and the related objects are defined analogously; for example, $v_0,...,v_\ell$ is a directed path if for every $i\in[\ell]$ it holds that (v_{i-1},v_i) is a directed edge (which is directed from v_{i-1} to v_i). It is common to consider also a pair of anti-parallel edges as a simple directed cycle.

A directed acyclic graph (DAG) is a digraph that has no directed cycles. Every DAG has at least one vertex having out-degree (resp., in-degree) zero, called a sink (resp., a source). A simple directed acyclic graph $G=(V,E)$ is called an inward (resp., outward) directed tree if $|E| = |V|-1$ and there exists a unique vertex, called the root, having out-degree (resp., in-degree) zero. Note that each vertex in an inward (resp., outward) directed tree can reach the root (resp., is reachable from the root) by a unique directed path.[1]

[1]Note that in any DAG, there is a directed path from each vertex v to some sink (resp., from

Representation. Graphs are commonly represented by their adjacency matrix and/or their incidence lists. The adjacency matrix of a simple graph $G=(V,E)$ is a $|V|$-by-$|V|$ Boolean matrix in which the (i,j)-th entry equals 1 if and only if i and j are adjacent in G. The incidence list representation of G consists of $|V|$ sequences such that the i^{th} sequence is an ordered list of the set of edges incident at vertex i.

Computational problems. Simple computational problems regarding graphs include determining whether a given graph is connected (and/or acyclic) and finding shortest paths in a given graph. Another simple problem is determining whether a given graph is bipartite, where a graph $G=(V,E)$ is bipartite (or 2-colorable) if there exists a 2-coloring of its vertices that does not assign neighboring vertices the same color. All these problems are easily solvable by BFS.

Moving to more complicated tasks that are still solvable in polynomial-time, we mention the problem of finding a perfect matching (or a maximum matching) in a given graph, where a matching is a subgraph in which all vertices have degree 1, a perfect matching is a matching that contains all the graph's vertices, and a maximum matching is a matching of maximum cardinality (among all matching of the said graph).

Turning to seemingly hard problems, we mention that the problem of determining whether a given graph is 3-colorable (i.e., G3C) is NP-complete. A few additional NP-complete problems follow.

- A Hamiltonian path (resp., Hamiltonian cycle) in the graph $G=(V,E)$ is a *simple* path (resp., cycle) that passes through all the vertices of G. Such a path (resp., cycle) has length $|V|-1$ (resp., $|V|$). The problem is to determine whether a given graph contains a Hamiltonian path (resp., cycle).

- An independent set (resp., clique) of the graph $G=(V,E)$ is a set of vertices $V' \subseteq V$ such that the subgraph induced by V' contains no edges (resp., contains all possible edges). The problem is to determine whether a given graph has an independent set (resp., a clique) of a given size.

  A vertex cover of the graph $G=(V,E)$ is a set of vertices $V' \subseteq V$ such that each edge in E has at least one end-point in V'. Note that V' is a vertex cover of G if and only if $V \setminus V'$ is an independent set of V.

A natural computational problem which is believed to be neither in $\mathcal{P}$ nor NP-complete is the graph isomorphism problem. The input consists of two graphs, $G_1=(V_1,E_1)$ and $G_2=(V_2,E_2)$, and the question is whether there exist a 1-1 and onto mapping $\phi : V_1 \to V_2$ such that $\{u,v\}$ is in E_1 if and only if $\{\phi(u),\phi(v)\}$ is in E_2. (Such a mapping is called an isomorphism.)

some source to each vertex v). In an inward (resp., outward) directed tree this sink (resp., source) must be unique. The condition $|E| = |V| - 1$ enforces the uniqueness of these paths, because (combined with the reachability condition) it implies that the underlying graph (obtained by disregarding the orientation of the edges) is a tree.

G.2 Boolean Formulae

In §1.2.4.3, Boolean formulae are defined as a special case of Boolean circuits (§1.2.4.1). Here we take the more traditional approach, and define Boolean formulae as structured sequences over an alphabet consisting of variable names and various connectives. It is most convenient to define Boolean formulae recursively as follows:

- A variable is a Boolean formula.
- If $\phi_1, ..., \phi_t$ are Boolean formulae and ψ is a t-ary Boolean operation then $\psi(\phi_1, ..., \phi_t)$ is a Boolean formula.

Typically, we consider three Boolean operations: the unary operation of negation (denoted `neg` or $\neg$), and the (bounded or unbounded) conjunction and disjunction (denoted $\wedge$ and $\vee$, respectively). Furthermore, the convention is to shorthand $\neg(\phi)$ by $\neg\phi$, and to write $(\wedge_{i=1}^{t}\phi_i)$ or $(\phi_1 \wedge \cdots \wedge \phi_t)$ instead of $\wedge(\phi_1, ..., \phi_t)$, and similarly for $\vee$.

Two important special cases of Boolean formulae are CNF and DNF formulae. A CNF formula is a conjunction of disjunctions of variables and/or their negation; that is, $\wedge_{i=1}^{t}\phi_i$ is a CNF if each ϕ_i has the form $(\vee_{j=1}^{t_i}\phi_{i,j})$, where each $\phi_{i,j}$ is either a variable or a negation of a variable (and is called a literal). If for every i it holds that $t_i \leq 3$ then we say that the formula is a 3CNF. Similarly, DNF formulae are defined as disjunctions of conjunctions of literals.

The value of a Boolean formula under a truth assignment to its variables is defined recursively along its structure. For example, $\wedge_{i=1}^{t}\phi_i$ has the value `true` under an assignment τ if and only if every ϕ_i has the value `true` under τ. We say that a formula ϕ is satisfiable if there exists a truth assignment τ to its variables such that the value of ϕ under τ is `true`.

The set of satisfiable CNF (resp., 3CNF) formulae is denoted `SAT` (resp., `3SAT`), and the problem of deciding membership in it is NP-complete. The set of tautologies (i.e., formula that have the value `true` under any assignment) is coNP-complete, even when restricted to 3DNF formulae.

Quantified Boolean Formulae. In contrast to the foregoing that refers to unquantified Boolean formulae, a quantified Boolean formula is a formula augmented with quantifiers that refer to each variable appearing in it. That is, if ϕ is a formula in the Boolean variables $x_1, ..., x_n$ and $Q_1, ..., Q_n$ are Boolean quantifiers (i.e., each Q_i is either $\exists$ or $\forall$) then $Q_1 x_1 \cdots Q_n x_n \, \phi(x_1, ..., x_n)$ is a quantified Boolean formula. A k-alternating quantified Boolean formula is a quantified Boolean formula with up to k alternating sequences of existential and universal quantifiers, starting with an existential quantifier. For example, $\exists x_1 \exists x_2 \forall x_3 \phi(x_1, x_2, x_3)$ is a 2-alternating quantified Boolean formula. (We say that a quantified Boolean formula is satisfiable if it evaluates to `true`.)

The set of satisfiable k-alternating quantified Boolean formulae is denoted `kQBF` and is Σ_k-complete, whereas the set of all satisfiable quantified Boolean formulae is denoted `QBF` and is $\mathcal{PSPACE}$-complete.

The foregoing definition refers to the canonical form of quantified Boolean formulae, in which all the quantifiers appear at the leftmost side of the formula. A more general definition allows each variable to be quantified at an arbitrary place to the left of its leftmost occurrence in the formula (e.g., $(\forall x_1)(\exists x_2)(x_1 = x_2) \wedge (\exists x_3)(x_3 = x_1))$. Note that such generalized formulae (used in the proof of Theorems 5.15 and 9.4) can be transformed to the canonical form by "pulling" all quantifiers to the left of the formula (e.g., $\forall x_1 \exists x_2 \exists x_3 ((x_1 = x_2) \wedge (x_3 = x_1))$).

G.3 Finite Fields, Polynomials and Vector Spaces

Various algebraic objects, computational problems and techniques play an important role in complexity theory. The most dominant such objects are finite fields as well as vector spaces and polynomials over such fields.

Finite Fields. We denote by $\mathrm{GF}(q)$ the finite field of q elements and note that q may be either a prime or a prime power. In the first case, $\mathrm{GF}(q)$ is viewed as consisting of the elements $\{0, ..., q-1\}$ with addition and multiplication being defined modulo q. Indeed, $\mathrm{GF}(2)$ is an important special case. In the case that $q = p^e$, where p is a prime and $e > 1$, the standard representation of $\mathrm{GF}(p^e)$ refers to an irreducible polynomial of degree e over $\mathrm{GF}(p)$. Specifically, if f is an irreducible polynomial of degree e over $\mathrm{GF}(p)$ then $\mathrm{GF}(p^e)$ can be represented as the set of polynomials of degree at most $e-1$ over $\mathrm{GF}(p)$ with addition and multiplication defined modulo the polynomial f.

We mention that finding representations of large finite fields is a non-trivial computational problem, where in both cases we seek an *efficient* algorithm that finds a representation (i.e., either a large prime or an irreducible polynomial) in time that is polynomial in the length of the representation. In the case of a field of prime cardinality, this calls for generating a prime number of adequate size, which can be done efficiently by a randomized algorithm (while a corresponding deterministic algorithm is not known). In the case of $\mathrm{GF}(p^e)$, where p is a prime and $e > 1$, we need to find an irreducible polynomial of degree e over $\mathrm{GF}(p)$. Again, this task is efficiently solvable by a randomized algorithm (see [24]), but a corresponding deterministic algorithm is not known for the general case (i.e., for arbitrary prime p and $e > 1$). Fortunately, for $e = 2 \cdot 3^{e'}$ (with e' being an integer), the polynomial $x^e + x^{e/2} + 1$ is irreducible over $\mathrm{GF}(2)$, which means that finding a representation of $\mathrm{GF}(2^e)$ is easy in this case. Thus, *there exists a strongly explicit construction of an infinite family of finite fields* (i.e., $\{\mathrm{GF}(2^e)\}_{e \in \mathbb{L}}$, where $\mathbb{L} = \{2 \cdot 3^{e'} : \varepsilon' \in \mathbb{N}\}$).

Polynomials and Vector Spaces. The set of degree $d-1$ polynomials over a finite field F (of cardinality at least d) forms a d-dimensional vector space over F (e.g., consider the basis $\{1, x, ..., x^{d-1}\}$). Indeed, the standard representation of this vector space refers to the basis $1, x, ..., x^{d-1}$, and (when referring to this basis) the polynomial $\sum_{i=0}^{d-1} c_i x^i$ is represented as the vector $(c_0, c_1, ..., c_{d-1})$. An alternative basis is obtained by considering the evaluation at d distinct points $\alpha_1, ..., \alpha_d \in F$; that is, the degree $d-1$ polynomial p is represented by the sequence of values $(p(\alpha_1), ..., p(\alpha_d))$. Needless to say, moving between such representations (i.e., representations with respect to different bases) amounts to applying an adequate linear transformation; that is, for $p(x) = \sum_{i=0}^{d-1} c_i x^i$, we have

$$\begin{pmatrix} p(\alpha_1) \\ p(\alpha_2) \\ \vdots \\ p(\alpha_d) \end{pmatrix} = \begin{pmatrix} 1 & \alpha_1 & \cdots & \alpha_1^{d-1} \\ 1 & \alpha_2 & \cdots & \alpha_2^{d-1} \\ \vdots & \vdots & \cdots & \vdots \\ 1 & \alpha_d & \cdots & \alpha_d^{d-1} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{d-1} \end{pmatrix} \tag{G.1}$$

where the (full rank) matrix in Eq. (G.1) is called a Vandermonde matrix. The foregoing transformation (or rather its inverse) is closely related to the task of polynomial interpolation (i.e., given the values of a degree $d-1$ polynomial at d points, find the polynomial itself).

G.4 The Determinant and the Permanent

Recall that the permanent of an n-by-n matrix $M = (a_{i,j})$ is defined as the sum $\sum_\pi \prod_{i=1}^n a_{i,\pi(j)}$ taken over all permutations π of the set $\{1, ..., n\}$. This is related to the definition of the determinant in which the same sum is used except that some elements are negated; that is, the determinant of $M = (a_{i,j})$ is defined as $\sum_\pi (-1)^{\sigma(\pi)} \prod_{i=1}^n a_{i,\pi(j)}$, where $\sigma(\pi) = 1$ if π is an even permutation (i.e., can be expressed by an even number of transpositions) and $\sigma(\pi) = -1$ otherwise.

The corresponding computational problems (i.e., computing the determinant or permanent of a given matrix) seem to have vastly different complexities. The determinant can be computed in polynomial-time; moreover, it can be computed in uniform $\mathcal{NC}^2$. In contrast, computing the permanent is $\#\mathcal{P}$-complete, even in the special case of matrices with entries in $\{0, 1\}$ (see Theorem 6.20).

G.5 Primes and Composite Numbers

A prime is a natural number that is not divisible by any natural number other than itself and 1. A natural number that is not a prime is called composite, and its prime factorization is the set of primes that divide it; that is, if $N = \prod_{i=1}^t P_i^{e_i}$, where the P_i's are distinct primes (greater than 1) and $e_i \geq 1$, then $\{P_i : i = 1, ..., t\}$ is the prime factorization of N. (If $t = 1$ then N is a prime power.)

Two famous computational problems, identified by Gauss as fundamental ones, are testing primality (i.e., given a natural number, determine whether it is prime or composite) and factoring composite integers (i.e., given a composite number, find its prime factorization). Needless to say, in both cases, the input is presented in binary representation. Although testing primality is reducible to integer factorization, the problems seem to have different complexities: While testing primality is in $\mathcal{P}$ (see [3] (and §6.1.2.2 showing that the problem is in $\mathcal{BPP}$)), *it is conjectured that* *factoring composite integers is intractable.* In fact, many popular candidates for various cryptographic systems are based on this conjecture.

Extracting modular square roots. Two related computational problems are extracting (modular) square roots with respect to prime and composite moduli. Specifically, a quadratic residue modulo a prime P is an integer s such that there exists an integer r satisfying $s \equiv r^2 \pmod{P}$. The corresponding search problem (i.e., given such P and s, find r) can be solved in probabilistic polynomial-time (see Exercise 6.16). The corresponding problem for composite moduli is computationally equivalent to factoring (see [183]); furthermore, extracting square roots modulo N is easily reducible to factoring N, and factoring N is randomly reducible to extracting square roots modulo N (even in a typical-case sense). We mention that even the problem of *deciding whether or not a given integer has a modular square root modulo a given composite* is conjectured to be hard (but is not known to be computationally equivalent to factoring).

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, J. Bruck, J. Naor, M. Naor and R. Roth. Construction of Asymptotically Good, Low-Rate Error-Correcting Codes through Pseudo-Random Graphs. *IEEE Transactions on Information Theory*, Vol. 38, pages 509–516, 1992.

[9] 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*, pages 251–260, 2006.

[10] 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, pages 289–304, 1992. Preliminary version in *31st FOCS*, 1990.

[11] N. Alon and J.H. Spencer. *The Probabilistic Method.* John Wiley & Sons, Inc., 1992. Second edition, 2000.

[12] 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.

[13] S. Arora. Approximation schemes for NP-hard geometric optimization problems: A survey. *Math. Programming*, Vol. 97, pages 43–69, July 2003.

[14] S. Arora abd B. Barak. *Complexity Theory: A Modern Approach.* Cambridge University Press, to appear.

[15] 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.

[16] 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.

[17] H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics.* McGraw-Hill, 1998.

[18] L. Babai. Trading Group Theory for Randomness. In *17th ACM Symposium on the Theory of Computing*, pages 421–429, 1985.

[19] L. Babai. Random oracles separate PSPACE from the Polynomial-Time Hierarchy. *Information Processing Letters*, Vol. 26, pages 51–53, 1987.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] E. Bach and J. Shallit. *Algorithmic Number Theory* (Volume I: Efficient Algorithms). MIT Press, 1996.

[25] B. Barak. Non-Black-Box Techniques in Crypptography. PhD Thesis, Weizmann Institute of Science, 2004.

[26] W. Baur and V. Strassen. The Complexity of Partial Derivatives. *Theor. Comput. Sci.* 22, pages 317–330, 1983.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] A. Ben-Dor and S. Halevi. In *2nd Israel Symp. on Theory of Computing and Systems*, IEEE Computer Society Press, pages 108-117, 1993.

[32] 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

[33] 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.

[34] 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.

[35] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan. Robust PCPs of Proximity, Shorter PCPs, and Applications to Coding. *SIAM Journal on Computing*, Vol. 36 (4), 2006, pages 889–974. Extended abstract in *36th STOC*, 2004.

[36] E. Ben-Sasson and M. Sudan. Simple PCPs with Poly-log Rate and Query Complexity. In *37th ACM Symposium on the Theory of Computing*, pages 266–275, 2005.

[37] 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.

[38] M. Blum. A Machine-Independent Theory of the Complexity of Recursive Functions. *Journal of the ACM*, Vol. 14 (2), pages 290–305, 1967.

[39] M. Blum and S. Kannan. Designing Programs that Check their Work. In *21st ACM Symposium on the Theory of Computing*, pages 86–97, 1989.

[40] 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.

[41] 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.

[42] 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.

[43] A. Bogdanov and L. Trevisan. On worst-case to average-case reductions for NP problems. *SIAM Journal on Computing*, Vol. 36 (4), 2006, pages 1119–1159. Extended abstract in *44th FOCS*, 2003.

[44] A. Bogdanov and L. Trevisan. Average-case complexity. *Foundations and Trends in Theoretical Computer Science*, Vol. 2:1, 2006.

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

[46] 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.

[47] A. Borodin. Computational Complexity and the Existence of Complexity Gaps. *Journal of the ACM*, Vol. 19 (1), pages 158–174, 1972.

[48] A. Borodin. On Relating Time and Space to Size and Depth. *SIAM Journal on Computing*, Vol. 6 (4), pages 733–744, 1977.

[49] 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.

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

[51] G.J. Chaitin. On the Length of Programs for Computing Finite Binary Sequences. *Journal of the ACM*, Vol. 13, pages 547–570, 1966.

[52] A.K. Chandra, D.C. Kozen and L.J. Stockmeyer. Alternation. *Journal of the ACM*, Vol. 28, pages 114–133, 1981.

[53] 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.

[54] 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.

[55] A. Church. An Unsolvable Problem of Elementary Number Theory. *Amer. J. of Math.*, Vol. 58, pages 345–363, 1936.

[56] N. Creignou, S. Khanna, and M. Sudan. *Complexity Classifications of Boolean Constraint Satisfaction Problems. SIAM Monographs on Discrete Mathematics and Applications*, 2001.

[57] A. Cobham. The Intristic Computational Difficulty of Functions. In *Proc. 1964 International Congress for Logic Methodology and Philosophy of Science*, pages 24–30, 1964.

[58] S.A. Cook. The Complexity of Theorem Proving Procedures. In *3rd ACM Symposium on the Theory of Computing*, pages 151–158, 1971.

[59] S.A. Cook. An Overview of Computational Complexity. Turing Award Lecture. *CACM*, Vol. 26 (6), pages 401–408, 1983.

[60] S.A. Cook. A Taxonomy of Problems with Fast Parallel Algorithms. *Information and Control*, Vol. 64, pages 2–22, 1985.

[61] S.A. Cook and R.A. Reckhow. The Relative Efficiency of Propositional Proof Systems. *J. of Symbolic Logic*, Vol. 44 (1), pages 36–50, 1979.

[62] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9, pages 251–280, 1990.

[63] T.M. Cover and G.A. Thomas. *Elements of Information Theory.* John Wiley & Sons, Inc., New-York, 1991.

[64] P. Crescenzi and V. Kann. A compendium of NP Optimization problems. Available at `http://www.nada.kth.se/~viggo/wwwcompendium/`

[65] R.A. DeMillo and R.J. Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, Vol. 7 (4), pages 193–195, June 1978.

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

[67] I. Dinur. The PCP Theorem by Gap Amplification. In *38th ACM Symposium on the Theory of Computing*, pages 241–250, 2006.

[68] I. Dinur and O. Reingold. Assignment-testers: Towards a combinatorial proof of the PCP-Theorem. *SIAM Journal on Computing*, Vol. 36 (4), 2006, pages 975–1024. Extended abstract in *45th FOCS*, 2004.

[69] I. Dinur and S. Safra. The importance of being biased. In *34th ACM Symposium on the Theory of Computing*, pages 33–42, 2002.

[70] J. Edmonds. Paths, Trees, and Flowers. *Canad. J. Math.*, Vol. 17, pages 449–467, 1965.

[71] S. Even. *Graph Algorithms.* Computer Science Press, 1979.

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

[73] U. Feige, S. Goldwasser, L. Lovász and S. Safra. On the Complexity of Approximating the Maximum Size of a Clique. Unpublished manuscript, 1990.

[74] 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.

[75] 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. Preliminary version in *31st FOCS*, 1990.

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

[77] 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.

[78] G.D. Forney. *Concatenated Codes.* MIT Press, Cambridge, MA 1966.

[79] 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.

[80] 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.

[81] S. Fortune. A Note on Sparse Complete Sets. *SIAM Journal on Computing*, Vol. 8, pages 431–433, 1979.

[82] 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.

[83] 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.

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

[85] 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.

[86] J. von zur Gathen. Algebraic Complexity Theory. *Ann. Rev. Comput. Sci.*, Vol. 3, pages 317–347, 1988.

[87] O. Goldreich. *Foundation of Cryptography – Class Notes*. Computer Science Dept., Technion, Israel, Spring 1989. Superseded by [91, 92].

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

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

[90] O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*. Algorithms and Combinatorics series (Vol. 17), Springer, 1999.

[91] O. Goldreich. *Foundation of Cryptography: Basic Tools*. Cambridge University Press, 2001.

[92] O. Goldreich. *Foundation of Cryptography: Basic Applications*. Cambridge University Press, 2004.

[93] O. Goldreich. Short Locally Testable Codes and Proofs (Survey). *ECCC*, TR05-014, 2005.

[94] O. Goldreich. On Promise Problems (a survey in memory of Shimon Even [1935-2004]). *ECCC*, TR05-018, 2005.

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

[96] 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.

[97] 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. Extended abstract in *37th FOCS*, 1996.

[98] 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.

[99] 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.

[100] 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.

[101] 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.

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

[103] O. Goldreich and D. Ron. Property testing in bounded degree graphs. *Algorithmica*, pages 302–343, 2002. Extended abstract in *29th STOC*, 1997.

[104] O. Goldreich and D. Ron. A sublinear bipartite tester for bounded degree graphs. *Combinatorica*, Vol. 19 (3), pages 335–373, 1999. Extended abstract in *30th STOC*, 1998.

[105] 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.

[106] O. Goldreich, S. Vadhan and A. Wigderson. On interactive proofs with a laconic provers. *Computational Complexity*, Vol. 11, pages 1–53, 2002.

[107] O. Goldreich and A. Wigderson. Computational Complexity. In *The Princeton Companion to Mathematics*, to appear.

[108] 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.

[109] 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.

[110] 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.

[111] 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.

[112] S.W. Golomb. *Shift Register Sequences.* Holden-Day, 1967. (Aegean Park Press, revised edition, 1982.)

[113] V. Guruswami, C. Umans, and S. Vadhan. Extractors and condensers from univariate polynomials. *ECCC*, TR06-134, 2006.

[114] J. Hartmanis and R.E. Stearns. On the Computational Complexity of of Algorithms. *Transactions of the AMS*, Vol. 117, pages 285–306, 1965.

[115] 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*, 1986.

[116] 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).

[117] J. Håstad. Getting optimal in-approximability results. *Journal of the ACM*, Vol. 48, pages 798–859, 2001. Extended abstract in *29th STOC*, 1997.

[118] 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).

[119] J. Håstad and S. Khot. Query efficient PCPs with pefect completeness. In *42nd IEEE Symposium on Foundations of Computer Science*, pages 610–619, 2001.

[120] A. Healy. Randomness-Efficient Sampling within NC1. *Computational Complexity*, to appear. Preliminary version in *10th RANDOM*, 2006.

[121] A. Healy, S. Vadhan and E. Viola. Using nondeterminism to amplify hardness. *SIAM Journal on Computing*, Vol. 35 (4), pages 903–931, 2006.

[122] D. Hochbaum (ed.). *Approximation Algorithms for NP-Hard Problems.* PWS, 1996.

[123] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[124] S. Hoory, N. Linial, and A. Wigderson. *Expander Graphs and their Applications. Bull. AMS*, Vol. 43 (4), pages 439–561, 2006.

[125] N. Immerman. Nondeterministic Space is Closed Under Complementation. *SIAM Journal on Computing*, Vol. 17, pages 760–778, 1988.

[126] R. Impagliazzo. Hard-core Distributions for Somewhat Hard Problems. In *36th IEEE Symposium on Foundations of Computer Science*, pages 538–545, 1995.

[127] 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.

[128] 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.

[129] 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.

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

[131] 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.

[132] 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.

[133] B. Juba and M. Sudan. Towards Universal Semantic Communication. Manuscript, February 2007. Available from `http://theory.csail.mit.edu/~madhu/papers/juba.pdf`

[134] V. Kabanets and R. Impagliazzo. Derandomizing Polynomial Identity Tests Means Proving Circuit Lower Bounds. *Computational Complexity*, Vol. 13, pages 1-46, 2004. Preliminary version in *35th STOC*, 2003.

[135] N. Kahale. Eigenvalues and Expansion of Regular Graphs. *Journal of the ACM*, Vol. 42 (5), pages 1091–1106, September 1995.

[136] 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.

[137] 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.

[138] 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.

[139] 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.

[140] R.M. Karp and V. Ramachandran. Parallel Algorithms for Shared-Memory Machines. In *Handbook of Theoretical Computer Science, Vol A: Algorithms and Complexity*, 1990.

[141] 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.

[142] M.J. Kearns and U.V. Vazirani. *An introduction to Computational Learning Theory.* MIT Press, 1994.

[143] 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.

[144] V.M. Khrapchenko. A method of determining lower bounds for the complexity of Pi-schemes. In *Matematicheskie Zametki*, Vol. 10 (1), pages 83–92, 1971 (in Russian). English translation in *Mathematical Notes of the Academy of Sciences of the USSR*, Vol. 10 (1), pages 474–479, 1971.

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

[146] D.E. Knuth. *The Art of Computer Programming*, Vol. 2 (*Seminumerical Algorithms*). Addison-Wesley Publishing Company, Inc., 1969 (first edition) and 1981 (second edition).

[147] A. Kolmogorov. Three Approaches to the Concept of "The Amount Of Information". *Probl. of Inform. Transm.*, Vol. 1/1, 1965.

[148] E. Kushilevitz and N. Nisan. *Communication Complexity.* Cambridge University Press, 1996.

[149] R.E. Ladner. On the Structure of Polynomial Time Reducibility. *Journal of the ACM*, Vol. 22, 1975, pages 155–171.

[150] C. Lautemann. BPP and the Polynomial Hierarchy. *Information Processing Letters*, Vol. 17, pages 215–217, 1983.

[151] F.T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[152] L.A. Levin. Universal Search Problems. *Problemy Peredaci Informacii 9*, pages 115–116, 1973 (in Russian). English translation in *Problems of Information Transmission 9*, pages 265–266.

[153] L.A. Levin. Randomness Conservation Inequalities: Information and Independence in Mathematical Theories. *Information and Control*, Vol. 61, pages 15–37, 1984.

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

[155] L.A. Levin. Fundamentals of Computing. *SIGACT News*, Education Forum, special 100th issue, Vol. 27 (3), pages 89–110, 1996.

[156] M. Li and P. Vitanyi. *An Introduction to Kolmogorov Complexity and its Applications.* Springer Verlag, August 1993.

[157] R.J. Lipton. New directions in testing. *Distributed Computing and Cryptography*, J. Feigenbaum and M. Merritt (ed.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematics Society, Vol. 2, pages 191–202, 1991.

[158] N. Livne. All Natural NPC Problems Have Average-Case Complete Versions. *ECCC*, TR06-122, 2006.

[159] 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.

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

[161] M. Luby and A. Wigderson. Pairwise Independence and Derandomization. *Foundations and Trends in Theoretical Computer Science*, Vol. 1:4, 2005. Preliminary version: TR-95-035, ICSI, Berkeley, 1995.

[162] 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.

[163] F. MacWilliams and N. Sloane. *The theory of error-correcting codes.* North-Holland, 1981.

[164] G.A. Margulis. Explicit Construction of Concentrators. *Prob. Per. Infor.*, Vol. 9 (4), pages 71–80, 1973 (in Russian). English translation in *Problems of Infor. Trans.*, pages 325–332, 1975.

[165] S. Micali. Computationally Sound Proofs. *SIAM Journal on Computing*, Vol. 30 (4), pages 1253–1298, 2000. Preliminary version in *35th FOCS*, 1994.

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

[167] P.B. Miltersen and N.V. Vinodchandran. Derandomizing Arthur-Merlin Games using Hitting Sets. *Computational Complexity*, Vol. 14 (3), pages 256–279, 2005. Preliminary version in *40th FOCS*, 1999.

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

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

[170] J. Naor and M. Naor. Small-bias Probability Spaces: Efficient Constructions and Applications. *SIAM Journal on Computing*, Vol 22, 1993, pages 838–856. Preliminary version in *22nd STOC*, 1990.

[171] 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.

[172] M. Nguyen, S.J. Ong, S. Vadhan. Statistical Zero-Knowledge Arguments for NP from Any One-Way Function. In *47th IEEE Symposium on Foundations of Computer Science*, pages 3-14, 2006.

[173] N. Nisan. Pseudorandom bits for constant depth circuits. *Combinatorica*, Vol. 11 (1), pages 63–70, 1991.

[174] N. Nisan. Pseudorandom Generators for Space Bounded Computation. *Combinatorica*, Vol. 12 (4), pages 449–461, 1992. Preliminary version in *22nd STOC*, 1990.

[175] N. Nisan. $\mathcal{RL} \subseteq \mathcal{SC}$. *Computational Complexity*, Vol. 4, pages 1-11, 1994. Preliminary version in *24th STOC*, 1992.

[176] N. Nisan and A. Wigderson. Hardness vs Randomness. *Journal of Computer and System Science*, Vol. 49, No. 2, pages 149–167, 1994. Preliminary version in *29th FOCS*, 1988.

[177] N. Nisan and D. Zuckerman. Randomness is Linear in Space. *Journal of Computer and System Science*, Vol. 52 (1), pages 43–52, 1996. Preliminary version in *25th STOC*, 1993.

[178] C.H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.

[179] C.H. Papadimitriou and M. Yannakakis. Optimization, Approximation, and Complexity Classes. In *20th ACM Symposium on the Theory of Computing*, pages 229–234, 1988.

[180] N. Pippenger and M.J. Fischer. Relations among complexity measures. *Journal of the ACM*, Vol. 26 (2), pages 361–381, 1979.

[181] E. Post. A Variant of a Recursively Unsolvable Problem. *Bull. AMS*, Vol. 52, pages 264–268, 1946.

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

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

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

[185] R. Raz. A Parallel Repetition Theorem. *SIAM Journal on Computing*, Vol. 27 (3), pages 763–803, 1998. Extended abstract in *27th STOC*, 1995.

[186] 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.

[187] 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 (in Russian). English translation in *Soviet Math. Doklady*, 31, pages 354–357, 1985.

[188] 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 (in Russian). English translation in *Mathematical Notes of the Academy of Sci. of the USSR*, Vol. 41 (4), pages 333–338, 1987.

[189] A.R. Razborov and S. Rudich. Natural Proofs. *Journal of Computer and System Science*, Vol. 55 (1), pages 24–35, 1997. Preliminary version in *26th STOC*, 1994.

[190] O. Reingold. Undirected ST-Connectivity in Log-Space. In *37th ACM Symposium on the Theory of Computing*, pages 376–385, 2005.

[191] 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.

[192] H.G. Rice. Classes of Recursively Enumerable Sets and their Decision Problems. *Trans. AMS*, Vol. 89, pages 25–59, 1953.

[193] 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.

[194] 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.)

[195] 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.

[196] M. Saks and S. Zhou. $\mathrm{BP_H SPACE}(S) \subseteq \mathrm{DSPACE}(S^{3/2})$. *Journal of Computer and System Science*, Vol. 58 (2), pages 376–403, 1999. Preliminary version in *36th FOCS*, 1995.

[197] W.J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Science*, Vol. 4 (2), pages 177-192, 1970.

[198] A. Selman. On the structure of NP. *Notices Amer. Math. Soc.*, Vol. 21 (6), page 310, 1974.

[199] J.T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, Vol. 27 (4), pages 701–717, October 1980.

[200] R. Shaltiel. Recent Developments in Explicit Constructions of Extractors. In *Current Trends in Theoretical Computer Science: The Challenge of the New Century, Vol 1: Algorithms and Complexity*, World scietific, 2004. (Editors: G. Paun, G. Rozenberg and A. Salomaa.) Preliminary version in *Bulletin of the EATCS 77*, pages 67–95, 2002.

[201] 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.

[202] C.E. Shannon. A Symbolic Analysis of Relay and Switching Circuits. *Trans. American Institute of Electrical Engineers*, Vol. 57, pages 713–723, 1938.

[203] C.E. Shannon. A mathematical theory of communication. *Bell Sys. Tech. Jour.*, Vol. 27, pages 623–656, 1948.

[204] C.E. Shannon. Communication Theory of Secrecy Systems. *Bell Sys. Tech. Jour.*, Vol. 28, pages 656–715, 1949.

[205] A. Shamir. IP = PSPACE. *Journal of the ACM*, Vol. 39, No. 4, pages 869–877, 1992. Preliminary version in *31st FOCS*, 1990.

[206] A. Shpilka. Lower Bounds for Matrix Product. *SIAM Journal on Computing*, pages 1185-1200, 2003.

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

[208] M. Sipser. *Introduction to the Theory of Computation.* PWS Publishing Company, 1997.

[209] 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.

[210] R.J. Solomonoff. A Formal Theory of Inductive Inference. *Information and Control*, Vol. 7/1, pages 1–22, 1964.

[211] 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.

[212] 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`.

[213] L.J. Stockmeyer. The Polynomial-Time Hierarchy. *Theoretical Computer Science*, Vol. 3, pages 1–22, 1977.

[214] L. Stockmeyer. The Complexity of Approximate Counting. In *15th ACM Symposium on the Theory of Computing*, pages 118–126, 1983.

[215] 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.

[216] M. Sudan. Decoding of Reed Solomon codes beyond the error-correction bound. *Journal of Complexity*, Vol. 13 (1), pages 180–193, 1997.

[217] M. Sudan. Algorithmic introduction to coding theory. Lecture notes, Available from `http://theory.csail.mit.edu/~madhu/FT01/`, 2001.

[218] 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.

[219] R. Szelepcsenyi. A Method of Forced Enumeration for Nondeterministic Automata. *Acta Informatica*, Vol. 26, pages 279–284, 1988.

[220] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, Vol. 20 (5), pages 865–877, 1991.

[221] 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.

[222] L. Trevisan. Extractors and Pseudorandom Generators. *Journal of the ACM*, Vol. 48 (4), pages 860–879, 2001. Preliminary version in *31st STOC*, 1999.

[223] L. Trevisan. On uniform amplification of hardness in NP. In *37th ACM Symposium on the Theory of Computing*, pages 31–38, 2005.

[224] 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.

[225] 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.

[226] C. Umans. Pseudo-random generators for all hardness. *Journal of Computer and System Science*, Vol. 67 (2), pages 419–440, 2003.

[227] 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`.

[228] S. Vadhan. An Unconditional Study of Computational Zero Knowledge. *SIAM Journal on Computing*, Vol. 36 (4), 2006, pages 1160–1214. Extended abstract in *45th FOCS*, 2004.

[229] S. Vadhan. *Lecture Notes for CS 225: Pseudorandomness*, Spring 2007. Available from `http://www.eecs.harvard.edu/~salil`.

[230] L.G. Valiant. The Complexity of Computing the Permanent. *Theoretical Computer Science*, Vol. 8, pages 189–201, 1979.

[231] L.G. Valiant. A theory of the learnable. *CACM*, Vol. 27/11, pages 1134–1142, 1984.

[232] 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.

[233] 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.

[234] J. von Neumann, Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen*, 100, pages 295–320, 1928.

[235] I. Wegener. *The Complexity of Boolean Functions.* Wiley-Teubner, 1987.

[236] I. Wegener. *Branching Programs and Binary Decision Diagrams – Theory and Applications.* SIAM Monographs on Discrete Mathematics and Applications, 2000.

[237] A. Wigderson. The amazing power of pairwise independence. In *26th ACM Symposium on the Theory of Computing*, pages 645–647, 1994.

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

[239] A.C. Yao. Separating the Polynomial-Time Hierarchy by Oracles. In *26th IEEE Symposium on Foundations of Computer Science*, pages 1-10, 1985.

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

[241] S. Yekhanin. New Locally Decodable Codes and Private Information Retrieval Schemes. *ECCC*, TR06-127, 2006.

[242] R.E. Zippel. Probabilistic algorithms for sparse polynomials. In the *Proceedings of EUROSAM '79: International Symposium on Symbolic and Algebraic Manipulation*, E. Ng (Ed.), Lecture Notes in Computer Science (Vol. 72), pages 216–226, Springer, 1979.

[243] D. Zuckerman. Linear-Degree Extractors and the Inapproximability of Max-Clique and Chromatic Number. In *38th ACM Symposium on the Theory of Computing*, 2006, pages 681–690.

Index