On the Impact of Cryptography on Complexity Theory*

Oded Goldreich
Department of Computer Science
Weizmann Institute of Science
Rehovot, Israel.
oded.goldreich@weizmann.ac.il

May 5, 2019

Abstract

We trace three major directions of research in complexity theory to their origins in the foundations of cryptography. Specifically, we refer to the theory of pseudorandomness (including the various incarnations of this concept), to the study of various forms of probabilistic proof system (including interactive proofs, zero-knowledge proofs, and probabilistically checkable proofs), and to the finer study of reductions (including random self-reducibility, worst-case to average-case reductions, average-case preserving reductions, and black-box reductions).

Contents

1 The story **1**
1.1 Pseudorandomness . . . 1
1.2 Probabilistic proof systems . . . 3
1.3 Finer study of reductions . . . 5

2 Pseudorandomness: A wide computational perspective **6**
2.1 The general paradigm . . . 8
2.1.1 Three fundamental aspects . . . 9
2.1.2 Some instantiations of the general paradigm . . . 10
2.2 General-Purpose Pseudorandom Generators . . . 12
2.2.1 The Archetypical Application . . . 12
2.2.2 Pseudorandom Functions . . . 14
2.2.3 The Intellectual Contents of Pseudorandom Generators . . . 14

3 Probabilistic Proof Systems: A bird's eye view **15**
3.1 Interactive Proof Systems . . . 16
3.2 Zero-Knowledge Proof Systems . . . 17
3.3 Probabilistically Checkable Proof systems . . . 17
3.4 Doubly-efficient interactive proof systems . . . 18

References **18**

*To appear as Chapter 18 in an ACM book celebrating the work of Goldwasser and Micali.

1 The story

In this essay we discuss the impact that research in the foundations of cryptography has had on developments in complexity theory. In particular, we trace three major research directions in complexity theory to their origins in the foundations of cryptography. These directions are:

1. the theory of pseudorandomness, including the various incarnations of this concept;
2. the study of various forms of probabilistic proof system, including interactive proofs, zero-knowledge proofs, and probabilistically checkable proofs;
3. the finer study of reductions, including random self-reducibility, worst-case to average-case reductions, average-case preserving reductions, and black-box reductions.

In the following subsections, we shall tell the story of how these complexity theoretic studies have emerged from the study of the foundations of cryptography.

In contrast, in Sections 2 and 3, we shall further discuss two of these three (complexity theoretic) endeavors while ignoring their cryptographic origins. In Section 2, we offer a wide perspective on the notion pseudorandom generators, viewing it as general paradigm that includes the general-purpose pseudorandom generator studied in cryptography as a specific (archetypical) incarnation. In Section 3, we shall offer a bird's eye view on the aforementioned types of probabilistic proof systems.

1.1 Pseudorandomness

The notion of a pseudorandom generator has first emerged in practice, where such candidate generators were used for various sampling tasks. In that context, it was natural to require that the sequences produced by these generators pass various statistical tests (as reviewed at great length by Knuth [38]). Given the *ad hoc* nature of the choice of the statistical tests, such an approach fails to yield a robust notion of pseudorandom generators. The inadequacy of this approach is most striking in the cryptographic setting, where the adversary is likely to launch attacks that are not captured by natural statistical tests.

The potential applications of "cryptographically secure" pseudorandom generators in cryptography (e.g., for the construction of a (private-key) stream cipher), led Blum and Micali to propose such a notion and a candidate construction of it [10]. By their definition, a pseudorandom generator is an *efficient* deterministic algorithm that *stretches* a short random seed into a long sequence that is *unpredictable* by any feasible observer; that is, no feasible algorithm can predict the next bit in the sequence, when given the previous bits, with success probability that is non-negligibly higher than half (which is obtained by just tossing a coin). We stress that, under this definition, the potential predictor may be stronger than the generator (as long as it is feasible); this reflects the default cryptographic principle by which the adversary may be more powerful than the honest user (i.e., may be willing to invest more resources than are required for proper use of the system that it attacks).

Having other applications in mind, Yao observed that the unpredictability requirement is equivalent to requiring that the output of the generator be *computationally indistinguishable* from a truly random sequence [56], where the notion of computational indistinguishability is exactly the one put forward by Goldwasser and Micali [28]. Recall that Goldwasser and Micali suggested this notion as a pivot of their definition of secure encryptions, while arguing that indistinguishable distributions

are equivalent for all practical purposes. Specifically, by their definition, an encryption scheme is secure if the encryptions of any two messages (of the same length) are computationally indistinguishable. Again, the cryptographic origin of this definition mandates that, in the context of pseudorandom generators, the potential distinguisher may be stronger than the generator.

The foregoing notion of a pseudorandom generator implies that *any efficient randomized algorithm maintains its performance when its internal coin tosses are substituted by a sequence generated by a pseudorandom generator*. The fact that these pseudorandom generators can be used in all efficient applications, including applications that are run for more time than the generator itself, identifies them as *general purpose* constructs, and hence we call them *general-purpose pseudorandom generators*. We mention that such pseudorandom generators exist if and only if one-way functions exist [32].

General-purpose pseudorandom generators are actually the archetypical incarnation of a general paradigm. In general, pseudorandom generators are efficient deterministic procedures that stretch short random seeds into longer "pseudorandom" sequences. Thus, a generic formulation of pseudorandom generators consists of specifying three fundamental aspects – the *stretch measure* of the generators; the class of distinguishers that the generators are supposed to fool (i.e., the algorithms with respect to which the *computational indistinguishability* requirement should hold); and the resources that the generators are allowed to use (i.e., their own *computational complexity*). Other incarnations of this general paradigm are telegraphically reviewed next.

One notable example is provided by pseudorandom generators that suffice for the derandomization of randomized complexity classes such as $\mathcal{BPP}$, which is the application envisioned by Yao [56]. In such applications, after replacing the original random-tape by the output of a generator, one considers a deterministic algorithm that scans all possible seeds of the generator (and invokes the generator on each possible seed). Hence, as observed by Nisan and Wigderson [46], in such applications, one may allow the generator to run in time that is exponential in its seed length, which is typically much larger than the running time of the distinguishers that one needs to fool. We call such pseudorandom generators *canonical derandomizers*, and note that they can be constructed under seemingly weaker intractability assumption than those required for the construction of general-purpose pseudorandom generators [46, 36].

Another famous incarnation of the notion of pseudorandom generators consists of generators that fool bounded-space machines. Such generators can be constructed without relying on any intractability assumption, and their seed length and space complexity is only moderately higher than the space complexity of the algorithms that they fool [44, 47]. Other incarnations of the paradigm refer to passing very restricted tests such as local tests (yielding limited independence generators) or linear tests (yielding small bias generators). We call such pseudorandom generators *special purpose*, and note that such generators of exponential stretch can be constructed unconditionally (see [20, Sec. 8.5]).

To summarize: The theory of pseudorandomness provides a fresh view at the *question of randomness*, which has puzzled thinkers for ages. This theory postulates that a distribution is random (or rather pseudorandom) if it cannot be told apart from the uniform distribution by any efficient procedure. The paradigm, originally associating efficient procedures with polynomial-time algorithms, has been applied also with respect to a variety of limited classes of such distinguishing procedures. Thus, (pseudo)randomness is not an inherent property of an object, but is rather subjective to the observer. At the extreme, this approach says that the question of whether the world is deterministic or allows for some free choice (which may be viewed as sources of randomness)

is irrelevant. *What matters is how the world looks to us and to various computationally bounded devices.* That is, if some phenomenon looks random, then we may just treat it as if it were random.

Hence, the theory of pseudorandomness is pivoted at the notion of computational indistinguishability, which in turn was put forward by Goldwasser and Micali, in the context of defining secure encryption schemes [28]. The archetypical incarnation of this theory, yielding the notion of *general-purpose pseudorandom generator*, was derived from the cryptographic setting considered by Blum and Micali [10], but other incarnations were proposed as well. The latter were either directly or indirectly inspired by the archetypical case.

In Section 2 we provide a wide perspective on the theory of pseudorandomness, but refrain from reproducing definitions and results that appear in [19, Sec. 3]. Our focus in Section 2 will be on aspects that are not covered in [19, Sec. 3]. A more detailed treatment of the subject can be found in [20, Chap. 8].

1.2 Probabilistic proof systems

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

The notion of a verification procedure presupposes the notion of computation[1], and furthermore the notion of efficient computation. This implicit dependency is made explicit in the definition of NP-proof systems (giving rise to the class $\mathcal{NP}$), where efficient computation is associated with deterministic polynomial-time algorithms. However, we can gain a lot if we are willing to take a somewhat non-traditional step and allow *probabilistic* verification procedures. In particular:

- Randomized and interactive verification procedures, giving rise to *interactive proof systems*, seem much more powerful than their deterministic counterparts (see Section 3.1).
- Such randomized procedures allow the introduction of *zero-knowledge proofs*, which are of great conceptual and practical interest (see Section 3.2).
- NP-proofs can be efficiently transformed into a (redundant) form (called a *probabilistically checkable proof*) that offers a trade-off between the number of bit-locations examined in the NP-proof and the confidence in its validity (see Section 3.3).

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

Like in the case of pseudorandom generators, the story of probabilistic proof systems originates in cryptography. It begins with Goldwasser, Micali and Rackoff who sought a general setting for their novel notion of zero-knowledge [29], which was aimed to capture cryptographic protocols that preserve the secrecy of the inputs of their users. The choice fell on proof systems – as capturing a fundamental activity that takes place in a cryptographic protocol. Motivated by the desire to

[1] This may explain the historical fact that notions of computation were first *rigorously formulated* in the context of logic.

formulate the most general type of "proofs" that may be used within cryptographic protocols, they introduced the notion of an *interactive proof system* [29]. Although the main thrust of their paper is the introduction of a special type of interactive proofs (i.e., ones that are *zero-knowledge*), the possibility that interactive proof systems may be more powerful from NP-proof system has been pointed out in [29].

Independently of [29][2], Babai suggested a different formulation of interactive proofs, which he called *Arthur-Merlin Games* [4]. Syntactically, Arthur-Merlin Games are a restricted form of interactive proof systems, yet it was subsequently shown that these restricted systems are as powerful as the general ones [30]. Babai's motivation was to place a group-theoretic problem, previously placed in $\mathcal{NP}$ under some group-theoretic assumptions, "as close to $\mathcal{NP}$ as possible" without using any assumptions. Interestingly, Babai underestimated the expressive power of interactive proof systems, conjecturing that the class of sets possessing such proof systems (even with an unbounded number of message-exchange rounds) is "very close" to $\mathcal{NP}$.

The first evidence of the surprising power of interactive proofs was given by Goldreich, Micali, and Wigderson, who presented an interactive proof system for Graph Non-Isomorphism [24], a set not known to be in $\mathcal{NP}$. More importantly, their paper has demonstrated the generality and wide applicability of zero-knowledge proofs. Assuming the existence of one-way function, it was shown how to construct zero-knowledge interactive proofs for any set in $\mathcal{NP}$. This result has had a dramatic impact on the design of cryptographic protocols (cf., [25]). In addition, this result has called attention to the then-new notion of interactive proof systems (since zero-knowledge NP-proofs could exist only in a trivial sense [26]).

A generalization of interactive proofs to *multi-prover interactive proofs* was suggested by Ben-Or, Goldwasser, Kilian and Wigderson [8]. Again, the main motivation came from zero-knowledge aspects; specifically, introducing multi-prover zero-knowledge proofs for $\mathcal{NP}$ without relying on intractability assumptions. Yet, the complexity theoretic prospects of the new class, denoted $\mathcal{MIP}$, have not been ignored. A more appealing, to our taste, formulation of the class $\mathcal{MIP}$ has been presented in [16]. The latter formulation exactly coincides with the formulation now known as *probabilistically checkable proofs* (i.e., $\mathcal{PCP}$).

The cryptographic lens was responsible for yet another development regarding interactive proof system. Motivated by the desire to construct schemes for delgating computation in a relaiable manner, Goldwasser, Kalai, and Rothblum [27] introduced the notion of *doubly-efficient* interactive proof systems. In such proof systems, originally termed "interactive proofs for muggles" (where "muggles" are non-magicians in the Harry Potter lingo), the prover should be relatively efficient and the verifier should be super-efficient. Specifically, in the context of delegation schemes, the prover should run in time that is polynomially related to the complexity of the delegated computation, whereas the verifier should be much faster than the latter complexity.

Hence, each of the aforementioned four types of probabilistic proof systems was originally proposed in order to address some cryptographic concern. More generally, these works (esp., the first one [29]) introduced the idea that a proof system may be probabilistic, and that the resulting probabilistic proof systems yield very meaningful notions that have many practical benefits. We also mention that the cryptographic lens motivated the definition of computationally sound proof systems (a.k.a. argument systems) [12].[3]

[2] Although both [29] and [4] appeared in the same conference (i.e., *17th STOC*, 1985), early versions of [29] have existed in 1982, and were rejected three times from major conferences (i.e., *FOCS83*, *STOC84*, and *FOCS84*).

[3] Furthermore, a cryptographic primitive (i.e., collision resistant hash functions) was combined with PCP systems

In Section 3 we provide a very brief introduction to the aforementioned types of probabilistic proof systems. A detailed treatment of the basic definitions and results can be found in [20, Chap. 9], whereas a primer on doubly-efficient interactive proof systems appeared as [21].

1.3 Finer study of reductions

The notions of random self-reducibility, worst-case to average-case reductions, average-case preserving reductions, and black-box reductions emerged naturally from the study of the foundations of cryptography. In this subsection, we briefly trace their emergence.

Random self-reducibility. Although random self-reducibility was used as an algorithmic tool in the design of "index calculus" algorithms [1, 42, 48] for solving the Discrete Logarithm Problem, its first emergence as a tool for establishing hardness occured in the work of Goldwasser and Micali [28]: Specifically, they identified random self-reducibility as the King's road to establishing worst-case to average-case reductions, and this road was taken by many subsequent works, most notably by [6]. Loosely speaking, if solving a problem on any instance x can be reduced to solving the same problem on m random $|x|$-bit long instances, which need *not* be independently distributed, then the worst-case hardness of the problem implies that it is hard to solve on at least an $1/3m$ fraction of the domain.[4]

Worst-case to average-case reductions. Goldwasser and Micali [28] introduced the aforementioned reduction in order to base the security of their proposed encryption scheme on a seemingly reliable (worst-case) intractability assumption. Their encryption scheme consists of encrypting a bit σ by a random element of $\mathbb{Z}_N$ having a Jacobi symbol 1 and quadratic character σ, where N is the product of two primes that are each congruent to 3 mod 4. Recall that under their robust definition of security, which was introduced in [28], security was interpreted as the indistinguishably of an encryption of 0 from an encryption of 1. Hence, proving security of their scheme required showing that it is infeasible to distinguish a quadratic residue mod M from a quadratic non-residue of Jacobi symbol 1 mod M. Indeed, Goldwasser and Micali showed that if the Quadratic Residuosity problem was hard on the worst-case, then the foregoing distinguishing task is infeasible. This was shown by reducing the Quadratic Residuosity problem to the distinguishing task, which is equivalent to predicting the quadratic character of random numbers that have Jacobi symbol 1; that is, by showing a worst-case to average-case reduction.

We warn that the foregoing complexity measures are not purely worst-case or average-case, since they refer to a fixed parameter, which in the foregoing cases in the composite moduli N. In contrast, subsequent complexity theoretic studies of worst-case to average-case reductions do refer to such pure notions (see, e.g., [11]). In any case, we stress that it was realized from the very beginning of the study of the foundations of cryptography (i.e., from [28])[5] that cryptographic applications have to be secure in an average-case sense, and so basing their security of a worst-case intractability assumption (such as $\mathcal{P} \neq \mathcal{NP}$) requires a worst-case to average-case reduction.

---

to yield argument systems with extremely efficient verification procedures [37].

[4] The counter-positive asserts that an efficient algorithm that solves the problem correctly on on at least a $1-(1/3m)$ fraction of the domain, yields an efficient algorithm that solve the problem correctly on each instance with probability at least $2/3$.

[5] Some researchers realized this point before [28]. For example, in the late 1970s, Shimon Even realized that NP-hardness of the problem of breaking an encryption scheme does not guarantee its security.

Average-case preserving reductions. Relations between different cryptographic primitives are typically proved by reductions that preserve average-case hardness. This thread was also pioneered by Goldwasser and Micali, who showed that a (secure) bit-encryption scheme implies a (secure) full-fledged encryption scheme [28]. Shortly after, Blum and Micali showed that the average-case hardness of DLP implies a "hard-core predicate" (of the modular exponentiation function), which in turn implies a pseudorandom generator [10]. (The argument was generalized by Yao [56].) All these results are proved by a reduction that preserves average-case hardness in an adequate sense. Specifically, the reductions transform a violation of the average-case hardness of the claimed primitive to the violation of the average-case hardness of the given primitive.

A related ("point-wise") notion of preserving average-case hardness is pivotal to Levin's theory of average-case complexity, which was suggested a couple of years later [39].[6]

Yao's result by which weakly one-way functions imply (strong) one-way functions [56] (see exposition in [20, Sec. 7.1.2]) heralded a line of research known as "hardness amplification" (see, e.g., [34]), which is too rich to review here. Still, the "take home message" is that it all started in cryptography.

Black-box reductions. All traditional reductions used in complexity theory (e.g., for establishing NP-hardness) are black-box.[7] In fact, the definition of a Cook-reduction refers to an abstract oracle that provides answers to queries regarding the target problem (see, e.g., [20, Sec. 2.2]), and the notion of a Karp-reduction is a special case. Although some early expositions of the notion of NP-completeness entertained the possibility that a set $S \in \mathcal{NP}$ may be "NP-complete" if *it holds that* $S \in \mathcal{P}$ *implies* $\mathcal{NP} = \mathcal{P}$, the standard notion of NP-completeness calls for a reduction. Yet, the possibility of showing hardness without presenting a (black-box) reduction re-emerged in the study of the foundations of cryptography.

It began with the work of Impagliazzo and Rudich [35], who essentially showed that the security of a public-key encryption scheme cannot be reduced to the existence of one-way permutations via a black-box reduction. This result was taken as indication to the impossibility of constructing public-key encryption schemes based on one-way functions. Similarly, the fact that protocols of a certain type cannot be demonstrated to be zero-knowledge using a black-box simulator [23], was taken as indication to the non-existence of such zero-knowledge protocols. The latter belief was refuted by Barak [7] a decade later, and the interpretation of the host of black-box separation results that followed [35] is a controversial topic. For a careful examination of the relevant issues, the interested reader is directed to [51].

We mention that a natural notion in the context of zero-knowledge is one of a universal simulator, which obtains the code of the verifier (which is simulates) as an auxiliary input. Such a simulator (used by Barak [7] and subsequent works in cryptography) corresponds to the notion of a "white box" reduction, which is often considered in complexity theory (e.g., in the context of derandomization; see [33], which explicitly discusses the distinction between black-box and white-box reductions as well as the possibility of non-constructive proofs (of implications)).

2 Pseudorandomness: A wide computational perspective

Indistinguishable things are identical.[8]

[6] The interested reader may prefer the expositions provided in [17] and [20, Sec. 10.2.1].

[7] Indeed, this follows the notion of Turing-reduction used in computability theory (see, e.g., [20, Sec. 1.2.3.6]).

G.W. Leibniz (1646–1714)

The second half of this century has witnessed the development of three theories of randomness, a notion which has been puzzling thinkers for ages. The first theory (cf., [13]), initiated by Shannon, is rooted in probability theory and is focused at distributions that are not perfectly random (i.e., are not uniform over a set of strings of adequate length). Shannon's Information Theory characterizes perfect randomness as the extreme case in which the *information contents* is maximized (i.e., the strings contain no redundancy at all). Thus, perfect randomness is associated with a unique distribution: the uniform one. In particular, by definition, one cannot (deterministically) generate such perfect random strings from shorter random seeds.

The second theory (cf., [40]), initiated by Solomonov, Kolmogorov, and Chaitin, is rooted in computability theory and specifically in the notion of a universal language (equiv., universal machine or computing device). It measures the complexity of objects in terms of the shortest program (for a fixed universal machine) that generates the object. Like Shannon's theory, Kolmogorov Complexity is quantitative and perfect random objects appear as an extreme case. However, in this approach one may say that a single object, rather than a distribution over objects, is perfectly random. Still, Kolmogorov's approach is inherently intractable (i.e., Kolmogorov Complexity is uncomputable), and – by definition – one cannot (deterministically) generate strings of high Kolmogorov Complexity from short random seeds.

The third theory, initiated by Blum, Goldwasser, Micali, and Yao [28, 10, 56], is rooted in the notion of *efficient computation* and is the focus of this section. This approach is explicitly aimed at providing a notion of randomness that allows for an efficient generation of random strings from shorter random seeds. The heart of this approach is the suggestion to view objects as equal if they cannot be told apart by any efficient procedure. Consequently, a distribution that cannot be efficiently distinguished from the uniform distribution will be considered as being random (or rather called pseudorandom). Thus, randomness is not an "inherent" property of objects (or distributions) but is rather relative to an observer (and its computational abilities). To demonstrate this approach, let us consider the following mental experiment.

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

[8] This is Leibniz's *Principle of Identity of Indiscernibles*. Leibniz admits that counterexamples to this principle are conceivable but will not occur in real life because God is much too benevolent.

> In the fourth alternative, Bob's recording equipment is directly connected to a *powerful computer* programmed to solve the motion equations and output a prediction. It is conceivable that in such a case Bob can substantially improve his guess of the outcome of the coin.

We conclude that the randomness of an event is relative to the information and computing resources at our disposal. Thus, a natural concept of pseudorandomness arises: a distribution is *pseudorandom* if no efficient procedure can distinguish it from the uniform distribution, where efficient procedures are associated with (probabilistic) polynomial-time algorithms. This notion of pseudorandomness is indeed the most fundamental one, yet weaker notions of pseudorandomness arise as well – they refer to indistinguishability by weaker procedures such as space-bounded algorithms, constant-depth circuits, etc.[9]

2.1 The general paradigm

The foregoing discussion has focused at one aspect of the pseudorandomness question – the resources or type of the observer (or potential distinguisher). Another important aspect is whether such pseudorandom sequences can be generated from much shorter ones, and at what cost (or complexity). A natural approach requires the generation process to be efficient, and furthermore to be fixed before the specific observer is determined. Coupled with the aforementioned strong notion of pseudorandomness, this yields the archetypical notion of pseudorandom generators – those operating in (fixed) polynomial-time and producing sequences that are indistinguishable from uniform ones by *any* polynomial-time observer. In particular, this means that the distinguisher is allowed more resources than the generator. Such (general-purpose) pseudorandom generators (discussed in Section 2.2) allow to decrease the randomness complexity of *any efficient application*, and are thus of great relevance to randomized algorithms and cryptography. The term *general-purpose* is meant to emphasize the fact that the same generator is good for all efficient applications, including those that consume more resources than the generator itself.

Although general-purpose pseudorandom generators are very appealing, there are important reasons for considering also the opposite relation between the complexities of the generation and distinguishing tasks; that is, allowing the pseudorandom generator to use more resources (e.g., time or space) than the observer it tries to fool. This alternative is natural in the context of derandomization (i.e., converting randomized algorithms to deterministic ones), where the crucial step is replacing the random input of an algorithm by a pseudorandom input, which in turn can be generated based on a much shorter random seed. In particular, when derandomizing a probabilistic polynomial-time algorithm, the observer (to be fooled by the generator) is a fixed algorithm. In this case employing a more complex generator merely means that the complexity of the derived deterministic algorithm is dominated by the complexity of the generator (rather than by the complexity of the original randomized algorithm). Needless to say, allowing the generator to use more resources than the observer that it tries to fool makes the task of designing pseudorandom

[9] We mention two perspectives on pseudorandomness that are somewhat different than the one presented in this section. Vadhan's treatment [54] emphasizes the *connections* between a variety of fundamental "pseudorandom objects" that seem very different in nature. The *Pseudorandomness program of the Simons Institute* (run in Jan–May 2017) emphasizes that *pseudorandomness and structure are complementing opposites*. In both cases (esp., in the second), the computational aspect is somewhat de-emphasized. The word "computational" was inserted in the title of this section in order to re-emphasize this aspect.

generators potentially easier, and enables derandomization results that are not known when using general-purpose pseudorandom generators.

We note that the goal of all types of pseudorandom generators is to allow the generation of "sufficiently random" sequences based on much shorter random seeds; that is, such generators are actually deterministic algorithms that *stretch* their input seeds into much longer pseudoramndom sequences. Our focus is on pseudorandom generators that have significant stretch, since they offer significant saving in the randomness complexity of various applications (and, in some cases, eliminating randomness altogether). Saving on randomness is valuable because many applications are severely limited in their ability to generate or obtain truly random bits. Furthermore, typically, generating truly random bits is significantly more expensive than standard computation steps. Thus, randomness is a computational resource that should be considered on top of time complexity (analogously to the consideration of space complexity).

2.1.1 Three fundamental aspects

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

Stretch function: A necessary requirement from any notion of a pseudorandom generator is that the generator is a *deterministic algorithm* that stretches short strings, called seeds, into longer output sequences.[10] Specifically, this algorithm stretches k-bit long seeds into $\ell(k)$-bit long outputs, where $\ell(k) > k$. The function $\ell : \mathbb{N} \to \mathbb{N}$ is called the stretch measure (or stretch function) of the generator. In some settings (e.g., in the case of general-purpose pseudorandom generators), the stretch measure can be amplified.

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

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

where U_n denotes the uniform distribution over $\{0,1\}^n$, and the probability is taken over U_k (resp., $U_{\ell(k)}$) as well as over the coin tosses of algorithm D in case it is probabilistic. The reader may think of such a distinguisher, D, as of an observer that tries to tell whether the "tested string" is a random output of the generator (i.e., distributed as $G(U_k)$) or is a truly random string (i.e., distributed

[10] Indeed, the seed represents the randomness that is used in the generation of the output sequences; that is, the randomized generation process is decoupled into a deterministic algorithm and a random seed. This decoupling facilitates the study of such processes.

as $U_{\ell(k)}$). The condition in Eq. (1) requires that D cannot make a meaningful decision; that is, ignoring a negligible difference (represented by $f(k)$), D's verdict is the same in both cases.[11] The archetypical choice is that $\mathcal{D}$ is the set of all probabilistic polynomial-time algorithms, and $\mathcal{F}$ is the set of all functions that are the reciprocal of some positive polynomial.

Complexity of Generation: This aspect refers to the complexity of the generator itself, when viewed as an algorithm. The archetypical choice is that the generator has to work in polynomial-time (i.e., make a number of steps that is polynomial in the length of its input – the seed). Other choices will be discussed as well. We note that placing no computational requirements on the generator (or, alternatively, imposing very mild requirements such as upper-bounding the running-time by a double-exponential function), yields "generators" that can fool any subexponential-size circuit family.[12]

2.1.2 Some instantiations of the general paradigm

Two important instantiations of the notion of pseudorandom generators relate to polynomial-time distinguishers.

General-purpose pseudorandom generators. This incarnation corresponds to the case that the generator itself runs in polynomial-time and is required to withstand *any probabilistic polynomial-time distinguisher*, including distinguishers that run for more time than the generator (i.e., Eq. (1) holds for all polynomial-time D's and $F = \{1/p : p \in \mathrm{POLY}\}$). Thus, the same generator may be used safely in any efficient application.

This notion is treated in [19, Sec. 3], and we shall further discuss it in Section 2.2. Recall that in this case, any pseudorandom generator (of any stretch function, including the minimal $\ell(k) = k+1$), implies a pseudorandom generator of any desired (polynomial) stretch function [20, Sec. 8.2.4].

Canonical derandomizers. In contrast, pseudorandom generators intended for derandomization may run more time than the distinguisher, which is viewed as a fixed circuit having size that is upper-bounded by a fixed polynomial (say, the quadratic polynomial n^2). Specifically, a canonical derandomizer is an exponential-time deterministic algorithm that stretches its k-bit long random seed to an $\ell(k)$-bit long sequence that fools any quadratic (in ℓ) size circuits (i.e., Eq. (1) holds for any circuit D of size $\ell(n)^2$ and $F = \{1/6\}$).

Note that a canonical derandomizer of exponential stretch implies that $\mathcal{BPP} = \mathcal{P}$. To see this, consider an arbitrary probabilistic polynomial-time algorithm, denoted A, that decides $S \in \mathcal{BPP}$, and denote is running time by p. Letting G denote the canonical derandomizer, and $\ell(k) = \exp(\Omega(k))$ denote its stretch, we obtain an algorithm A_G that, on input x, uniformly selects $s \in \{0,1\}^k$, where $k = O(\log |x|)$ such that $\ell(k) = p(|x|)$, and invokes A on input x and randomness $G(s)$. By the current incarnation of Eq. (1), it follows that, for every x, we have $|\Pr[A(x) = 1] - \Pr[A_G(x) = 1]| < 1/6$, since otherwise we obtain a $o(\ell(k)^2)$-size circuit that distinguishes

[11] The class of threshold functions $\mathcal{F}$ should be viewed as determining the class of noticeable probabilities (as a function of k). Thus, we require certain functions (i.e., those presented at the l.h.s of Eq. (1)) to be smaller than any noticeable function *on all but finitely many integers*. We call the former functions negligible. Note that a function may be neither noticeable nor negligible (e.g., it may be smaller than any noticeable function on infinitely many values and yet larger than some noticeable function on infinitely many other values).

[12] This fact can be proved via the probabilistic method; see [20, Exer. 8.1].

$U_{\ell(k)}$ from $G(U_k)$. Finally, by trying all possible random-tapes of A_G, we obtain a deterministic polynomial-time algorithm that decides S (i.e., this algorithm accepts x if and only if the majority of the possible random-tapes lead $A_G(x)$ to accept (i.e., iff $\Pr[A_G(x)=1] > 1/2$)).[13]

Note that if f is computable in exponential time but is hard to approximate (or predict), on the average, by circuits of smaller exponential size (with advantage proportional to their size), then $G(s) = (s, f(s))$ constitutes a canonical derandomizer (of minimal stretch). Interestingly, canonical derandomizers of exponential stretch can also be obtained in this case [46], by applying f to an exponential number of $\Omega(k)$-bit long substrings of the k-bit long seed that have relatively small pairwise intersections.[14] For further details on canonical derandomizers, the interested reader is referred to [20, Sec. 8.3].

We now turn to a few additional instantiations of the notion of pseudorandom generators. These instantiations refer to more limited classes of distinguishers such as log-space machines, local computations, and linear computations. In the known constructions for each of these cases, each bit in the output of the generator can be computed in time that is polynomial in the seed length.

Fooling space-bounded distinguishers. Here the distinguishers are space-bounded machines that have unidirectional access to the input they examine; actually, we may consider (non-uniform) OBDDs of bounded width.[15] The two main constructions known are at the extremes the relation between the distinguishers' time and space complexities (i.e., the OBDDs' length and width), where in both cases the generator itself has linear space complexity.

1. Using a seed of length $k = O(\log s)^2$, one can fool 2^s-width OBDDs that read $\exp(s)$ many bits (i.e., $\ell(k) = \exp(\sqrt{k})$) [44].

2. Using a seed of length $k = O(\log s)$, one can fool 2^s-width OBDDs that read $\mathrm{poly}(s)$ many bits (i.e., $\ell(k) = \mathrm{poly}(k)$) [47].

In the first result one should think of s as being logarithmic in the length of the output sequence (i.e., $s = O(\log \ell(k))$), whereas in the second result one should think of s as being a $O(1)$-root of the length of the output sequence (i.e., $s = \ell(k)^{1/O(1)}$). The specific construction of the first generator allows for derandomizing the class $\mathcal{BPL}$ in polylogarithmic space and polynomial time [45]. For further details on space-bounded pseudorandom generators, the interested reader is referred to [20, Sec. 8.4].

Fooling local distinguishers. Here we consider distinguishers that inspect a constant number, denoted t, of bits in the sequence output by the generator, where these bit locations are not *a priori* known. Random sequences that perfectly fool such distinguishers are called t-wise independent (since each sequence of t bits in them is uniformly distributed in $\{0,1\}^t$). Constructions of t-wise independence generators can achieve stretch $\ell(k) = 2^{k/t}$, and this result extends to sequences over $\Sigma = \{0,1\}^{k/t}$; for details, see [20, Sec. 8.5.1].

[13] Recall that for every x, either $\Pr[A(x)=1] \geq 2/3$ or $\Pr[A(x)=1] \leq 1/3$.

[14] We mention that the construction of [46] has also been applied in other settings. One case, which predated [46], is that of constant-depth circuits [43]. Another case is information theoretic; this case led to a breakthrough in the study of randomness extractors [53]. In both these cases, the hard function f can be proved to exist without relying on any intractability assumptions.

[15] Ordered binary decision diagrams (OBDD) are branching programs that reads bits of the input in a predetermined order. Their width correspond to an exponential function of the space bound.

Fooling linear distinguishers. Here we consider distinguishers that inspect a linear combination (over GF(2)) of bits in the sequence output by the generator, where the linear combination is not *a priori* known. Random sequences that fool such distinguishers with a probability gap of ϵ are called ϵ-biased. Constructions of ϵ-biased generators can achieve stretch $\ell(k) = \epsilon \cdot \exp(\Omega(k))$; for details, see [20, Sec. 8.5.2].

Fooling hitting tests distinguishers. Lastly, we consider distinguishers that inspect sequences over $\Sigma = \{0,1\}^b$. Each such distinguisher is associated with a target set $T \subseteq \Sigma$ of density at least half, and accepts the sequence if at least one of its elements hit the set T. A generator $G : \{0,1\}^k \to \Sigma^{\ell'(k)}$ is said to pass such a test if the probability that its output is not accepted (i.e., each element in $G(U_k)$ misses T) is at most $\exp(-\Omega(\ell'(k)))$. Such generators can be constructed for $\ell'(k) = \Omega(k - b)$; for details, see [20, Sec. 8.5.3].

2.2 General-Purpose Pseudorandom Generators

Randomness is playing an increasingly important role in computation: It is frequently used in the design of sequential, parallel and distributed algorithms, and it is of course central to cryptography. Whereas it is convenient to design such algorithms making free use of randomness, it is also desirable to minimize the usage of randomness in real implementations. Thus, general-purpose pseudorandom generators (as defined in [19, Sec. 3]) are a key ingredient in an "algorithmic tool-box" – they provide an automatic compiler of programs written with free usage of randomness into programs that make an economical use of randomness.

2.2.1 The Archetypical Application

Recall that "pseudo-random number generators" appeared with the first computers, and have been used ever since for generating random choices (or samples) for various applications. However, typical implementations use generators that are not pseudorandom according to our definition. Instead, at best, these generators are shown to pass *some* ad-hoc statistical test (cf., [38]). We warn that the fact that a "pseudo-random number generator" passes some statistical tests, does not mean that it will pass a new test and that it will be good for a future (untested) application. Needless to say, the approach of subjecting the generator to some ad-hoc tests fails to provide general results of the form "for *all* practical purposes using the output of the generator is as good as using truly unbiased coin tosses." In contrast, the approach encompassed in the definition of general-purpose pseudorandom generators aims at such generality, and in fact is tailored to obtain it: The notion of computational indistinguishability, which underlines this definition, covers all possible efficient applications and guarantees that for all of them pseudorandom sequences are as good as truly random ones. Indeed, any efficient randomized algorithm maintains its performance when its internal coin tosses are substituted by a sequence generated by a (general purpose) pseudorandom generator. This substitution is spelled out next.

Construction 2.1 (typical application of pseudorandom generators): *Let G be a* (general purpose) *pseudorandom generator with stretch function $\ell : \mathbb{N} \to \mathbb{N}$. Let A be a probabilistic polynomial-time algorithm, and $\rho : \mathbb{N} \to \mathbb{N}$ denote its randomness complexity. Denote by $A(x,r)$ the output of A on input x and coin tosses sequence $r \in \{0,1\}^{\rho(|x|)}$. Consider the following randomized algorithm, denoted A_G:*

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

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

Thus, using A_G instead of A, the randomness complexity is reduced from ρ to $\ell^{-1} \circ \rho$, while (as stated in Proposition 2.2) it is infeasible to find inputs (i.e., x's) on which the *noticeable behavior* of A_G is different from the one of A (and the non-existence of such inputs follows in case pseudorandomness holds with respect to polynomial size circuits).[16] For example, if $\ell(k) = k^2$, then the randomness complexity is reduced from ρ to $\sqrt{\rho}$. We stress that the pseudorandom generator G is *universal*; that is, it can be applied to reduce the randomness complexity of *any* probabilistic polynomial-time algorithm A.

Proposition 2.2 *Let A, ρ and G be as in Construction 2.1, and suppose that $\rho : \mathbb{N} \to \mathbb{N}$ is 1-1. Then, for every pair of probabilistic polynomial-time algorithms,* a finder F *and* a tester T*, every positive polynomial p and all sufficiently long n's*

$$\sum_{x \in \{0,1\}^n} \Pr[F(1^n) = x] \cdot \Delta_{A,T}(x) \;<\; \frac{1}{p(n)} \tag{2}$$

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

Algorithm F represents a potential attempt to find an input x on which the output of A_G is distinguishable from the output of A. This "attempt" may be benign as in the case that a user employs algorithm A_G on inputs that are generated by some probabilistic polynomial-time application. However, the attempt may also be adversarial as in the case that a user employs algorithm A_G on inputs that are provided by a potentially malicious party. The potential tester, denoted T, represents the potential use of the output of algorithm A_G, and captures the requirement that this output be as good as a corresponding output produced by A. Thus, T is given x as well as the corresponding output produced either by $A_G(x) \stackrel{\text{def}}{=} A(x, G'(U_{k(|x|)}))$ or by $A(x) = A(x, U_{\rho(|x|)})$, and it is required that T cannot tell the difference. In the case that A is a probabilistic polynomial-time *decision procedure*, this means that it is infeasible to find an x on which A_G decides incorrectly (i.e., differently than A). In the case that A is a *search procedure for some NP-relation*, it is infeasible to find an x on which A_G outputs a wrong solution. For details, see [20, Sec. 8.2.1].

Conclusion. Although Proposition 2.2 refers to standard probabilistic polynomial-time algorithms, a similar construction and analysis applied to any efficient randomized process (i.e., any efficient multi-party computation). Any such process preserves its behavior when replacing its perfect source of randomness (postulated in its analysis) by a pseudorandom sequence (which may be used in the implementation). Thus, given a pseudorandom generator with a large stretch function, *one can significantly reduce the randomness complexity of any efficient application.*

[16] That is, the (non-uniform) existential conclusion follows from a non-uniform hypothesis regarding G (i.e., that $G(U_k)$ is indistinguishable from $U_{\ell(k)}$ by any poly(k)-size circuit).

2.2.2 Pseudorandom Functions

Pseudorandom generators allow to efficiently generate long pseudorandom sequences from short random seeds (e.g., using k random bits, we can efficiently generate a pseudorandom bit-sequence of length k^2). Pseudorandom functions (defined below) are even more powerful: they allow efficient direct access to a huge pseudorandom sequence (which is infeasible to scan bit-by-bit). For example, based on k random bits, we define a sequence of length 2^k such that we can efficiently retrieve any desired bit in this sequence while the retrieved bits look random. In other words, pseudorandom functions can replace truly random functions in any efficient application (e.g., most notably in Cryptography). That is, pseudorandom functions are indistinguishable from random functions by any efficient procedure that may obtain the function values at arguments of its choice.

Definition 2.3 (pseudorandom functions [22]). *A* pseudorandom function (ensemble)*, with length parameters $\ell_{\mathrm{D}}, \ell_{\mathrm{R}} : \mathbb{N} \to \mathbb{N}$* (e.g., *$\ell_{\mathrm{D}}(k) = k$* and *$\ell_{\mathrm{R}}(k) = 1$*), is a collection of functions $\{F_k\}_{k \in \mathbb{N}}$, where*

$$F_k \stackrel{\mathrm{def}}{=} \{f_s : \{0,1\}^{\ell_{\mathrm{D}}(k)} \to \{0,1\}^{\ell_{\mathrm{R}}(k)}\}_{s \in \{0,1\}^k},$$

satisfying

(efficient evaluation) *There exists an efficient* (deterministic) *algorithm that when given a* seed*, s, and an $\ell_{\mathrm{D}}(|s|)$-bit* argument*, x, returns the $\ell_{\mathrm{R}}(|s|)$-bit long value $f_s(x)$.*

(Thus, the seed s is an "effective description" of the function f_s.)

(pseudorandomness) *For every probabilistic polynomial-time oracle machine M, every positive polynomial p, and all sufficiently large k*

$$\left| \Pr_{s \sim U_k}[M^{f_s}(1^k) = 1] - \Pr_{\rho \sim R_k}[M^{\rho}(1^k) = 1] \right| < \frac{1}{p(k)},$$

where R_k denotes the uniform distribution over all functions mapping $\{0,1\}^{\ell_{\mathrm{D}}(k)}$ to $\{0,1\}^{\ell_{\mathrm{R}}(k)}$, and $M^f(x)$ denotes the computation of M on input x when M's queries are answered by the function f.

Although pseudorandom functions seem stronger than pseudorandom generators, the former can be constructed using the latter (see [19, Sec. 3.3]).

We mention two ("non cryptographic") applications of pseudorandom functions to the theory of computation. The first, which originates in Valiant's seminal work on PAC learning [55], is the observation that pseudorandom functions yield concept classes that are infeasible to learn (since a learning algorithm for a concept class consisting of pseudorandom functions would distinguish pseudorandom functions from truly random functions, which cannot be learned at all). The second application is the pivotal role of pseudorandom functions in the "natural proofs" framework of [49].

2.2.3 The Intellectual Contents of Pseudorandom Generators

We shortly discuss some intellectual aspects of general-purpose pseudorandom generators. Actually, the first two aspects apply to all incarnations of the notion of a pseudorandom generator.

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

A relativistic view of randomness. Pseudorandomness is defined in terms of its observer: In the archetypical case of the general-purpose incarnation, a pseudorandom distribution is one that cannot be told apart from a uniform distribution by any efficient (i.e., polynomial-time) observer. However, the output of such pseudorandom generators can be distinguished from uniform sequences by an exponential-time machine (which is not at our disposal), which just tries all possible seeds (and rules that the sequence is random if and only if it is not in the image of the generator). Furthermore, the mere variety of different incarnations of the notion of computational indistinguishability testifies that pseudorandomness depends on the abilities of the observer. Hence, pseudorandomness is a relative notion.

Randomness and computational difficulty. In the archetypical case of the general-purpose incarnation (and also in the case of canonical derandomizers), pseudorandomness and computational difficulty play dual roles: The definition of pseudorandomness is pivoted at a difficult computational task (i.e., the task of distinguishing pseudorandom sequences from truly random ones). Furthermore, the known constructions of pseudorandom generators rely on conjectures regarding computational difficulty (e.g., the existence of one-way functions in the archetypical case), and this is inevitable: The existence of such pseudorandom generators implies some known intractability conjectures (e.g., the existence of one-way functions).

Randomness and Predictability. The connection between pseudorandomness and unpredictability (by efficient procedures) plays an important role in the analysis of several constructions of pseudorandom generators (see [20, Sec. 8.2.5.2] and well as [20, Sec. 8.3.2.2]). We wish to highlight the intuitive appeal of this connection.

3 Probabilistic Proof Systems: A bird's eye view

A proof is whatever convinces me.

Shimon Even (1935–2004)

The glory attributed to the creativity involved in finding proofs makes us forget that it is the less glorified process of verification that *defines* proof systems. The notion of a verification procedure presupposes the notion of computation, and furthermore the notion of efficient computation (because verification, unlike coming up with proofs, is supposed to be easy). Associating the set of valid assertions with a set of objects that have some property, we view a proof system for a set S (e.g., of satisfiable formulae) as a game between an all-powerful prover and an *efficient* verifier: Both receive an input x, and the prover attempts to convince the verifier that $x \in S$. We seek proof systems that are *complete* and *sound*, where completeness means that the prover succeeds for every $x \in S$, and soundness means that *any* prover fails for every $x \notin S$.

When taking the most natural choice of the efficiency requirement, namely restricting the verifier to be a deterministic polynomial-time machine, we get the definition of the class $\mathcal{NP}$ (rephrased as a proof system): *a set S is in $\mathcal{NP}$ if and only if membership in S can be verified by a deterministic polynomial-time machine when given an alleged proof of polynomial length* (i.e., polynomial in $|x|$).

Relaxing the efficiency requirement, we let the verifier be a *probabilistic* polynomial-time machine. Furthermore, we allowing it to "rule by statistical evidence" and hence to err (with low probability, which is explicitly bounded, and can be reduced via repetitions). This relaxation is not suggested as a substitute to the notion of a mathematical proof, but rather as a practical solution to the problem of verifying mundane assertions (like the fact that on input x, the program P halts with output $P(x)$). As we shall see below, this relaxation turns out to yield enormous advances in computer science.

3.1 Interactive Proof Systems

When the verifier is deterministic, we can always assume that the prover simply sends it a single message (the purported "proof"), and based on this message the verifier decides whether to accept or reject the common input x as a member of the target set S. (More extensive interaction does not help here, since the verifier's steps are predictable by the prover.)

When the verifier is probabilistic, *interaction* may add power. We thus consider a (randomized) interaction between the parties. Such an interaction which may be viewed as an "interrogation" of the teacher (prover) by a persistent student (verifier), who asks the teacher "tough" questions in order to be convinced of the correctness of the claim. Interestingly, it turns out that asking "tough" questions is not (significantly) better than asking random questions (even if one cares of the number of rounds [30]). In any case, since the verifier ought to be efficient (i.e., run in time polynomial in $|x|$), this interaction is bounded to have at most polynomially many rounds. The class $\mathcal{IP}$ (for Interactive Proofs) contains all sets S for which there is a verifier that accepts every $x \in S$ with probability 1 (after interacting with an adequate prover), but rejects any $x \notin S$ with probability at least 1/2 (no matter what strategy is employed by the prover).

Clearly, $\mathcal{NP} \subseteq \mathcal{IP}$: To prove that x is in an NP-set S, the prover just sends an adequate NP-witness, which the verifier can easily verify. But how can one prove that x is not in $S \in \mathcal{NP}$? That is, when proving that something (i.e., an NP-witness) exists, the prover merely presents it, but how can the prover convince the verifier that something (i.e., an NP-witness) does not exist? A major result asserts that interactive proofs exists for every set in $\mathcal{PSPACE} \supseteq \text{co}\mathcal{NP}$. In fact, we have

Theorem 3.1 [41, 52]: $\mathcal{IP} = \mathcal{PSPACE}$.

Recalling that it is widely believed that $\mathcal{NP} \neq \mathcal{PSPACE}$, it follows that interactive proofs seem more powerful than standard non-interactive and deterministic proofs (i.e., NP-proofs). In particular, since co$\mathcal{NP} \subseteq \mathcal{PSPACE}$, Theorem 3.1 implies that there are such interactive proofs for every set in co$\mathcal{NP}$, whereas some coNP-sets are believed not to have NP-proofs.

3.2 Zero-Knowledge Proof Systems

Here the thrust is not on being able to prove more assertions, but rather on having proofs with additional properties. Randomized and interactive verification procedures as in Section 3.1 allow the (meaningful) introduction of *zero-knowledge proofs*, which are proofs that yield nothing beyond their own validity. Such proofs seem counter-intuitive and undesirable for educational purposes, but they are very useful in cryptography.

For example, a *zero-knowledge proof* that a certain propositional formula is satisfiable does not reveal a satisfying assignment to the formula nor any partial information regarding such an assignment (e.g., whether the first variable can assume the value `true`). In general, whatever the verifier can efficiently compute after interacting with a zero-knowledge prover, can be efficiently reconstructed from the assertion itself (without interacting with anyone).

Clearly, any set in $\mathcal{BPP}$ has a zero-knowledge proof, in which the prover says nothing (and the verifier decides by itself). What is surprising is that zero-knowledge proofs seem to exist also for sets that are widely believed not to be in $\mathcal{BPP}$. In particular:

Theorem 3.2 [24]: *Assuming the existence of* (non-uniformly hard) *one-way functions, every set in $\mathcal{NP}$ has a zero-knowledge proof system.*

Interestingly, under the same condition any set in $\mathcal{IP}$ has a zero-knowledge proof system [9]. On the other hand, for the actual use of zero-knowledge proof systems, it is crucial that the prover strategy asserted in Theorem 3.2 can be implemented in probabilistic polynomial-time, when given an NP-witness for the common input. Of course, this zero-knowledge strategy does not consist of just sending the NP-witness; it rather consists of sending "commitments" to many "randomized versions" of the NP-witness, and allowing the verifier to inspect few random location in each such randomized witness (by decommitting to the locations selected by the verifier).

3.3 Probabilistically Checkable Proof systems

Let us return to the non-interactive mode, in which the verifier receives a (alleged) written proof. But now we restrict its access to the proof so as to read only a small part of it (which may be randomly selected by it). An excellent analogy is to imagine a referee trying to decide the correctness of a long proof by sampling a few lines of the proof. It seems hopeless to detect a single "bug" unless the entire "proof" is read; but this intuition is valid only for the "natural" way of writing down proofs, and fails when "robust" formats of proofs are used (and one is willing to settle for statistical evidence).

Such "robust" proof systems are called PCPs (for Probabilistically Checkable Proofs). Loosely speaking, a PCP system for a set S consists of a probabilistic polynomial-time verifier having access to an oracle that represents a proof in redundant form, where (as in case of NP-proofs) the length of the proof is polynomial in the length of the input. The verifier accesses only a constant number of the oracle bits, and accepts every $x \in S$ with probability 1 (when given access to an adequate

oracle), but rejects any $x \notin S$ with probability at least 1/2 (no matter to which oracle it is given access).

Theorem 3.3 (The PCP Theorem [3, 2]):[17] *Each set in $\mathcal{NP}$ has a* PCP *system. Furthermore, there exists a polynomial-time procedure for converting any NP-proof to the corresponding* PCP*-oracle.*

Indeed, the proof of the PCP Theorem suggests a way of writing "robust" proofs, in which any bug must "spread" all over.[18] One important application of the PCP Theorem (and its variants) is its connection to the complexity of combinatorial approximation. For example, using the PCP system of [31], it follows that it is NP-complete to decide, when given a linear system of equations over GF(2), *whether the fraction of mutually satisfiable equations is greater than* 99% *or smaller than* 51%.

3.4 Doubly-efficient interactive proof systems

Turning back to interactive proof systems, recall that their definition does not restrict the complexity of the strategy of the prescribed prover. Indeed, the constructions of [41, 52] use prover strategies of high complexity. This fact limits the applicability of these proof systems in practice. (Nevertheless, such proof systems may be actually applied when the prover knows something that the verifier does not know, such as an NP-witness to an NP-claim, and when the proof system offers an advantage such as zero-knowledge [29, 24].)

In constast, the definition of *doubly-efficient* interactive proof systems *requires the prescribed prover strategy to be implemented in polynomial-time and the verifier's strategy to be implemented in almost-linear-time*. (We stress that unlike in *argument systems* [12], the soundness condition holds for all possible cheating strategies (not only for feasible ones).) Restricting the prescribed prover to run in polynomial-time implies that such systems may exist only for sets in $\mathcal{BPP}$, whereas a polynomial-time verifier can check membership in such sets by itself. However, restricting the verifier to run in almost-linear-time implies that something can be gained by interacting with a more powerful prover, even though the latter is restricted to polynomial-time.

The foregoing potential was first demonstrated in [27], which presents doubly-efficient proof systems for any set that has log-space uniform circuits of small depth (e.g., log-space uniform $\mathcal{NC}$). An incomparable recent result of [50] provides such (constant-round) proof systems for any set that can be decided in polynomial-time and small amount of space (e.g., for all sets in $\mathcal{SC}$). That is, denoting by $\mathrm{TiSp}(T, s)$ the class of sets that can be decided by a (randomized) algorithm that runs in time T while using space s, we have:

Theorem 3.4 [50]: *For every polynomial p and $s(n) = \sqrt{n}$, each set in* $\mathrm{TiSp}(p, s)$ *has a* (constant round) *doubly-efficient proof system.*

Recall that each set having a doubly-efficient proof system is in $\mathcal{BPP}$, and note that it is also decidable in almost-linear space.

Acknowledgments

I am grateful to Yuval Ishai for helpful comments.

[17] See also an alternative proof of [14].

[18] The analogy to error correcting codes is indeed in place, and the cross fertilization between these two areas has been very significant.

References

[1] L.M. Adleman. A subexponential algorithm for the discrete logarithm problem with applications to cryptography, In *20th FOCS*, pages 55–60, 1979.

[2] 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.

[3] 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.

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

[5] 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.

[6] 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.

[7] B. Barak. How to Go Beyond the Black-Box Simulation Barrier. In *42nd IEEE Symposium on Foundations of Computer Science*, pages 106–115, 2001.

[8] 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.

[9] 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

[10] 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.

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

[12] 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.

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

[14] I. Dinur. The PCP Theorem by Gap Amplification. *Journal of the ACM*, Vol. 54 (3), Art. 12, 2007.

[15] 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.

[16] L. Fortnow, J. Rompel and M. Sipser. On the power of multi-prover interactive protocols. In *3rd IEEE Symposium on Structure in Complexity*, pages 156–161, 1988. See errata in *5th IEEE Symposium on Structure in Complexity*, pages 318–319, 1990.

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

[18] O. Goldreich. *Secure Multi-Party Computation*. Unpublished manuscript, June 1998. Available from the author's web-page (i.e., `http://www.wisdom.weizmann.ac.il/~oded/pp.html`).

[19] O. Goldreich. *Foundations of Cryptography – A Primer*. Foundations and Trends in TCS, Vol. 1 (1), NOW publishers, 2005.

[20] O. Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008.

[21] O. Goldreich. On Doubly-Efficient Interactive Proof Systems. *Foundations and Trends in Theoretical Computer Science*, Volume 13, Issue 3, 2018.

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

[23] 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.

[24] O. Goldreich, S. Micali and A. Wigderson. Proofs that Yield Nothing but their Validity or All Languages in NP Have Zero-Knowledge Proof Systems. *Journal of the ACM*, Vol. 38, No. 1, pages 691–729, 1991. Preliminary version in *27th FOCS*, 1986.

[25] 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. See details in [18].

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

[27] S. Goldwasser, Y. Kalai, and G.N. Rothblum. Delegating Computation: Interactive Proofs for Muggles. *Journal of the ACM*, Vol. 62(4), Art. 27:1-27:64, 2015. Extended abstract in *40th STOC*, pages 113–122, 2008.

[28] 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.

[29] 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.

[30] 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.

[31] J. Håstad. Getting optimal in-approximability results. *Journal of the ACM*, Vol. 48, pages 798–859, 2001. Extended abstract in *29th STOC*, 1997.

[32] 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).

[33] R. Impagliazzo, V. Kabanets, and A. Wigderson. In Search of an Easy Witness: Exponential Time vs. Probabilistic Polynomial Time. In *16th IEEE Conference on Computational Complexity*, pages 2–12, 2001.

[34] R. Impagliazzo, R. Jaiswal, V. Kabanets, and A. Wigderson. Uniform Direct Product Theorems: Simplified, Optimized, and Derandomized. *SIAM Journal on Computing*, Volume 39, Number 4, pages 1637–1665, 2010.

[35] R. Impagliazzo and S. Rudich. Limits on the Provable Consequences of One-Way Permutations. In *21st ACM Symposium on the Theory of Computing*, pages 44–61, 1989. Also presented in *CRYPTO'88*.

[36] 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.

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

[38] D.E. Knuth. *The Art of Computer Programming*, Vol. 2 (*Seminumerical Algorithms*). Addison-Wesley Publishing Company, Inc., 1969 (first edition) and 1981 (second edition).

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

[40] M. Li and P. Vitanyi. *An Introduction to Kolmogorov Complexity and its Applications*. Springer Verlag, August 1993.

[41] 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.

[42] R. Merkle. *Secrecy, authentication, and public key systems*. Ph.D. dissertation, Department of Electrical Engineering, Stanford University, 1979.

[43] N. Nisan. Pseudorandom bits for constant depth circuits. *Combinatorica*, Vol. 11 (1), pages 63–70, 1991.

[44] N. Nisan. Pseudorandom Generators for Space Bounded Computation. *Combinatorica*, Vol. 12 (4), pages 449–461, 1992. Preliminary version in *22nd STOC*, 1990.

[45] N. Nisan. $\mathcal{RL} \subseteq \mathcal{SC}$. *Computational Complexity*, Vol. 4, pages 1-11, 1994. Preliminary version in *24th STOC*, 1992.

[46] 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.

[47] 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.

[48] J. Pollard. Monte Carlo methods for index computations (mod p). *Math. Comp.*, Vol 32, pages 918–924, 1978.

[49] 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.

[50] O. Reingold, G. Rothblum, R. Rothblum. Constant-round interactive proofs for delegating computation. In *48th ACM Symposium on the Theory of Computing*, pages 49–62, 2016.

[51] O. Reingold, L. Trevisan, and S. Vadhan. Notions of Reducibility between Cryptographic Primitives. *1st TCC*, pages 1–20, 2004.

[52] A. Shamir. IP = PSPACE. *Journal of the ACM*, Vol. 39, No. 4, pages 869–877, 1992. Preliminary version in *31st FOCS*, 1990.

[53] L. Trevisan. Extractors and Pseudorandom Generators. *Journal of the ACM*, Vol. 48 (4), pages 860–879, 2001. Preliminary version in *31st STOC*, 1999.

[54] S. Vadhan. *Pseudorandomness*. Foundations and Trends in TCS, Vol. 7 (1–3), NOW publishers, 2012.

[55] L.G. Valiant. A Theory of the Learnable. *Communications of the ACM*, Vl. 27 (11), pages 1134–1142, 1984.

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