Pseudorandomness

Oded Goldreich
Department of Computer Science and Applied Mathematics
Weizmann Institute of Science, Rehovot, ISRAEL.

March 27, 2000

Abstract

We postulate that a distribution is pseudorandom if it cannot be told apart from the uniform distribution by any efficient procedure. This yields a robust definition of pseudorandom generators as efficient deterministic programs stretching short random seeds into longer pseudorandom sequences. Thus, pseudorandom generators can be used to reduce the randomness-complexity in any efficient procedure. Pseudorandom generators and computational difficulty are closely related: loosely speaking, each can be efficiently transformed into the other.

1 Introduction

The second half of this century has witnessed the development of three theories of randomness, a notion which has been puzzling thinkers for ages. The first theory (cf. [9]), initiated by Shannon [33], is rooted in probability theory and is focused at distributions that are not perfectly random. Shannon's Information Theory characterizes perfect randomness as the extreme case in which the *information content* is maximized (and there is no redundancy at all).[1] Thus, perfect randomness is associated with a unique distribution – the uniform one. In particular, by definition, one cannot generate such perfect random strings from shorter random strings.

The second theory (cf. [22, 23]), due to Solomonov [34], Kolmogorov [21] and Chaitin [6], 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.[2] Like Shannon's theory, Kolmogorov Complexity is quantitative and perfect random objects appear as an extreme case. Interestingly, 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 generate strings of high Kolmogorov Complexity from short random strings.

The third theory, initiated by Blum, Goldwasser, Micali and Yao [17, 4, 37], is rooted in complexity theory and is the focus of this survey. This approach is explicitly aimed at providing a

[1] In general, the amount of information in a distribution D is defined as $-\sum_x D(x)\log_2 D(x)$. Thus, the uniform distribution over strings of length n has information measure n, and any other distribution over n-bit strings has lower information measure. Also, for any function $f:\{0,1\}^n \rightarrow \{0,1\}^m$ with $n<m$, the distribution obtained by applying f to a uniformly distributed n-bit string has information measure at most n, which is strictly lower than the length of the output.

[2] For example, the string 1^n has Kolmogorov Complexity $O(1)+\log_2 n$ (by virtue of the program "print n ones" which has length dominated by the binary encoding of n. In contrast, a simple counting argument shows that most n-bit strings have Kolmogorov Complexity at least n (since each program can produce only one string).

notion of perfect randomness that nevertheless allows to efficiently generate perfect random strings from shorter random strings. 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 all of them Alice flips a coin high in the air, and Bob is asked to guess its outcome *before* the coin hits the floor. The alternative ways differ by the knowledge Bob has before making his guess. In the first alternative, Bob has to announce his guess before Alice flips the coin. Clearly, in this case Bob wins with probability 1/2. In the second alternative, Bob has to announce his guess while the coin is spinning in the air. Although the outcome is *determined in principle* by the motion of the coin, Bob does not have accurate information on the motion and thus we believe that also in this case Bob wins with probability 1/2. The third alternative is similar to the second, except that Bob has at his disposal sophisticated equipment capable of providing accurate *information* on the coin's motion as well as on the environment effecting the outcome. However, Bob cannot process this information in time to improve his guess. In the fourth alternative, Bob's recording equipment is directly connected to a *powerful computer* programmed to solve the motion equations and output a prediction. It is conceivable that in such a case Bob can improve substantially his guess of the outcome of the coin.

We conclude that the randomness of an event is relative to the information and computing resources at our disposal. 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.

Orientation Remarks

We consider finite objects, encoded by binary finite sequences called strings. When we talk of distributions we mean discrete probability distributions having a finite support that is a set of strings. Of special interest is the uniform distribution, that for a length parameter n (explicit or implicit in the discussion), assigns each n-bit string $x \in \{0,1\}^n$ equal probability (i.e., probability 2^{-n}). We will colloquially speak of "perfectly random strings" meaning strings selected according to such a uniform distribution.

We associate efficient procedures with probabilistic polynomial-time algorithms. An algorithm is called polynomial-time if there exists a polynomial p so that for any possible input x, the algorithm runs in time bounded by $p(|x|)$, where $|x|$ denotes the length of the string x. Thus, the running time of such algorithm grows moderately as a function of the length of its input. A probabilistic algorithm is one that can take random steps, where, without loss of generality, a random step consists of selecting which of two predetermined steps to take next so that each possible step is taken with probability 1/2. These choices are called the algorithm's internal coin tosses.

Organization, acknowledgment and further details

Sections 2 and 3 provide a basic treatment of *pseudorandom generators* (as briefly discussed in the abstract). The rest of this survey goes somewhat beyond: In Section 4 we treat *pseudorandom* *functions*, and in Section 5 we further discuss the practical and conceptual significance of pseudorandom generators. In Section 6 we discuss alternative notions of pseudorandom generators, viewing them all as special cases of a general paradigm. The survey is based on [11, Chap. 3], and the interested reader is referred to there for further details.

2 The Notion of Pseudorandom Generators

Loosely speaking, a pseudorandom generator is an *efficient* program (or algorithm) that *stretches* short random strings into long *pseudorandom* sequences. The latter sentence emphasizes three fundamental aspects in the notion of a pseudorandom generator:

1. Efficiency: The generator has to be efficient. As we associate efficient computations with polynomial-time ones, we postulate that the generator has to be implementable by a deterministic polynomial-time algorithm.

   This algorithm takes as input a string, called its seed. The seed captures a bounded amount of randomness used by a device that "generates pseudorandom sequences." The formulation views any such device as consisting of a deterministic procedure applied to a random seed.

2. Stretching: The generator is required to stretch its input seed to a longer output sequence. Specifically, it stretches n-bit long seeds into $\ell(n)$-bit long outputs, where $\ell(n) > n$. The function ℓ is called the stretching measure (or stretching function) of the generator.

3. Pseudorandomness: The generator's output has to look random to any efficient observer. That is, any efficient procedure should fail to distinguish the output of a generator (on a random seed) from a truly random sequence of the same length. The formulation of the last sentence refers to a general notion of computational indistinguishability, which is the heart of the entire approach.

2.1 Computational Indistinguishability

Intuitively, two objects are called computationally indistinguishable if no efficient procedure can tell them apart. As usual in complexity theory, an elegant formulation requires asymptotic analysis (or rather a functional treatment of the running time of algorithms in terms of the length of their input).[3] Thus, the objects in question are infinite sequences of distributions, where each distribution has a finite support. Such a sequence will be called a distribution ensemble. Typically, we consider distribution ensembles of the form $\{D_n\}_{n\in\mathbb{N}}$, where for some function $\ell : \mathbb{N}\rightarrow\mathbb{N}$, the support of each D_n is a subset of $\{0,1\}^{\ell(n)}$. Furthermore, typically ℓ will be a positive polynomial. For such D_n, we denote by $e\sim D_n$ the process of selecting e according to distribution D_n. Consequently, for a predicate P, we denote by $\Pr_{e\sim D_n}[P(e)]$ the probability that $P(e)$ holds when e is distributed (or selected) according to D_n.

Definition 1 (Computational Indistinguishability [17, 37]): *Two probability ensembles,* $\{X_n\}_{n\in\mathbb{N}}$ *and* $\{Y_n\}_{n\in\mathbb{N}}$*, are called* computationally indistinguishable *if for any probabilistic polynomial-time algorithm A, for any positive polynomial p, and for all sufficiently large n's*

$$|\Pr_{x\sim X_n}[A(x)=1] - \Pr_{y\sim Y_n}[A(y)=1]| < \frac{1}{p(n)}$$

[3] We stress that the asymptotic (or functional) treatment is not essential to this approach. One may develop the entire approach in terms of inputs of fixed lengths and an adequate notion of complexity of algorithms. However, such an alternative treatment is more cumbersome.

The probability is taken over X_n (resp., Y_n) as well as over the coin tosses of algorithm A.

A couple of comments are in place. Firstly, we have allowed algorithm A (called a distinguisher) to be probabilistic. This makes the requirement only stronger, and seems essential to several important aspects of our approach. Secondly, we view events occuring with probability that is upper bounded by the reciprocal of polynomials as negligible. This is well-coupled with our notion of efficiency (i.e., polynomial-time computations): An event that occurs with negligible probability (as a function of a parameter n), will also occur with negligible probability if the experiment is repeated for poly(n)-many times.

We note that computational indistinguishability is a strictly more liberal notion than statistical indistinguishability (cf. [37, 15]). An important case is the one of distributions generated by a pseudorandom generator as defined next.

2.2 Basic definition and initial discussion

We are now ready for the main definition. Recall that a stretching function, $\ell : \mathbb{N} \to \mathbb{N}$, satisfies $\ell(n) > n$ for all n.

Definition 2 (Pseudorandom Generators [4, 37]): *A deterministic polynomial-time algorithm G is called a* pseudorandom generator *if there exists a* stretching function, $\ell : \mathbb{N} \to \mathbb{N}$, *so that the following two probability ensembles, denoted $\{G_n\}_{n\in\mathbb{N}}$ and $\{R_n\}_{n\in\mathbb{N}}$, are computationally indistinguishable*

1. *Distribution G_n is defined as the output of G on a uniformly selected seed in $\{0,1\}^n$.*
2. *Distribution R_n is defined as the uniform distribution on $\{0,1\}^{\ell(n)}$.*

That is, letting U_m denote the uniform distribution over $\{0,1\}^m$, we require that for any probabilistic polynomial-time algorithm A, for any positive polynomial p, and for all sufficiently large n's

$$|\, \Pr_{s\sim U_n}[A(G(s)) = 1] \;-\; \Pr_{r\sim U_{\ell(n)}}[A(r) = 1]\,| \;<\; \frac{1}{p(n)}$$

Thus, pseudorandom generators are efficient (i.e., polynomial-time) deterministic programs that expand short randomly selected seeds into longer pseudorandom bit sequences, where the latter are defined as computationally indistinguishable from truly random sequences by efficient (i.e., polynomial-time) algorithms. It follows that any efficient randomized algorithm maintains its performance when its internal coin tosses are substituted by a sequence generated by a pseudorandom generator. That is,

Construction 3 (typical application of pseudorandom generators): *Let A be a probabilistic polynomial-time algorithm, and $\rho(n)$ denote an upper bound on its randomness complexity. Let $A(x,r)$ denote the output of A on input x and coin tosses sequence $r \in \{0,1\}^{\rho(|x|)}$. Let G be a pseudorandom generator with stretching function $\ell : \mathbb{N} \to \mathbb{N}$. Then A_G is a randomized algorithm that on input x, proceeds as follows. It sets $k = k(|x|)$ to be the smallest integer such that $\ell(k) \geq \rho(|x|)$, uniformly selects $s \in \{0,1\}^k$, and outputs $A(x,r)$, where r is the $\rho(|x|)$-bit long prefix of $G(s)$.*

It can be shown that it is infeasible to find long x's on which the *input-output behavior* of A_G is *noticeably different* from the one of A, although A_G may use much fewer coin tosses than A. That is

Proposition 4 *Let A and G be as above. For any algorithm D, let $\Delta_{A,D}(x)$ denote the discrepancy, as judged by D, in the behavior of A and A_G on input x. That is,*

$$\Delta_{A,D}(x) \stackrel{\text{def}}{=} |\,\Pr_{r\sim U_{\rho(n)}}[D(x,A(x,r))=1] \;-\; \Pr_{s\sim U_{k(n)}}[D(x,A_G(x,s))=1]\,|$$

where the probabilities are taken over the U_m's as well as over the coin tosses of D. Then for every pair of probabilistic polynomial-time algorithms, a finder *F and* a distinguisher *D, every positive polynomial p and all sufficiently long n's*

$$\Pr\left[\Delta_{A,D}(F(1^n)) > \frac{1}{p(n)}\right] \;<\; \frac{1}{p(n)}$$

where $|F(1^n)| = n$ and the probability is taken over the coin tosses of F.

The proposition is proven by showing that a triplet (A, F, D) violating the claim can be converted into an algorithm D' that distinguishes the output of G from the uniform distribution, in contradiction to the hypothesis. Analogous arguments are applied whenever one wishes to prove that an efficient randomized process (be it an algorithm as above or a multi-party computation) preserves its behavior when one replaces true randomness by pseudorandomness as defined above. Thus, given pseudorandom generators with large stretching function, *one can considerably reduce the randomness complexity in any efficient application.*

2.3 Amplifying the stretch function

Pseudorandom generators as defined above are only required to stretch their input a bit; for example, stretching n-bit long inputs to $(n+1)$-bit long outputs will do. Clearly, generator of such moderate stretch function are of little use in practice. In contrast, we want to have pseudorandom generators with an arbitrary long stretch function. By the efficiency requirement, the stretch function can be at most polynomial. It turns out that pseudorandom generators with the smallest possible stretch function can be used to construct pseudorandom generators with any desirable polynomial stretch function. (Thus, when talking about the existence of pseudorandom generators, we may ignore the stretch function.)

Theorem 5 [14]: *Let G be a pseudorandom generator with stretch function $\ell(n) = n+1$, and ℓ' be any polynomially-bounded stretch function, that is polynomial-time computable. Let $G_1(x)$ denote the $|x|$-bit long prefix of $G(x)$, and $G_2(x)$ denote the last bit of $G(x)$ (i.e., $G(x) = G_1(x)\,G_2(x)$). Then*

$$\begin{aligned} G'(s) \;&\stackrel{\text{def}}{=}\; \sigma_1\sigma_2\cdots\sigma_{\ell'(|s|)}\,, \\ &\quad \textit{where } x_0 = s,\ \sigma_i = G_2(x_{i-1}) \textit{ and } x_i = G_1(x_{i-1}),\ \textit{for } i = 1,...,\ell'(|s|) \end{aligned}$$

is a pseudorandom generator with stretch function ℓ'.

Proof Sketch: The theorem is proven using the *hybrid technique* (cf. [10, Sec. 3.2.3]): One considers distributions H_n^i (for $i = 0,...,\ell(n)$) defined by $U_i^{(1)} P_{\ell(n)-i}(U_n^{(2)})$, where $U_i^{(1)}$ and $U_n^{(2)}$ are independent uniform distributions (over $\{0,1\}^i$ and $\{0,1\}^n$, respectively), and $P_j(x)$ denotes the j-bit long prefix of $G'(x)$. The extreme hybrids correspond to $G'(U_n)$ and $U_{\ell(n)}$, whereas distinguishability of neighboring hybrids can be worked into distinguishability of $G(U_n)$ and U_{n+1}. Loosely speaking, suppose one could distinguish H_n^i from H_n^{i+1}. Then, using $P_j(s) = G_2(s)P_{j-1}(G_1(s))$

(for $j \geq 1$), this means that one can distinguish $H_n^i \equiv (U_i^{(1)}, G_2(U_n^{(2)}), P_{(\ell(n)-i)-1}(G_1(U_n^{(2)})))$ from $H_n^{i+1} \equiv (U_i^{(1)}, U_1^{(1')}, P_{\ell(n)-(i+1)}(U_n^{(2')}))$. Incorporating the generation of $U_i^{(1)}$ and the evaluation of $P_{\ell(n)-i-1}$ into the distinguisher, one could distinguish $(G_1(U_n^{(2)}), G_2(U_n^{(2)})) \equiv G(U_n)$ from $(U_n^{(2')}, U_1^{(1')}) \equiv U_{n+1}$, in contradiction to the pseudorandomness of G. ■

3 How to Construct Pseudorandom Generators

The known constructions transform computation difficulty, in the form of one-way functions (defined below), into pseudorandomness generators. Loosely speaking, a *polynomial-time computable* function is called one-way if any efficient algorithm can invert it only with negligible success probability. For simplicity, we consider only length-preserving one-way functions.

Definition 6 (one-way function): *A* one-way function, *f, is a polynomial-time computable function such that for every probabilistic polynomial-time algorithm A', every positive polynomial $p(\cdot)$, and all sufficiently large n's*

$$\mathrm{Pr}_{x\sim U_n}\left[A'(f(x)) \in f^{-1}(f(x))\right] < \frac{1}{p(n)}$$

where U_n is the uniform distribution over $\{0,1\}^n$.

Popular candidates for one-way functions are based on the conjectured intractability of integer factorization (cf. [30] for state of the art), the discrete logarithm problem (cf. [31] analogously), and decoding of random linear code [16]. The infeasibility of inverting f yields a weak notion of unpredictability: Let $b_i(x)$ denotes the i^{th} bit of x. Then, for every probabilistic polynomial-time algorithm A (and sufficiently large n), it must be the case that $\mathrm{Pr}_{i,x}[A(i, f(x)) \neq b_i(x)] > 1/2n$, where the probability is taken uniformly over $i \in \{1, ..., n\}$ and $x \in \{0,1\}^n$. A stronger (and in fact strongest possible) notion of unpredictability is that of a hard-core predicate. Loosely speaking, a *polynomial-time computable* predicate b is called a hard-core of a function f if any efficient algorithm, given $f(x)$, can guess $b(x)$ only with success probability that is negligible better than half.

Definition 7 (hard-core predicate [4]): *A polynomial-time computable predicate $b : \{0,1\}^* \to \{0,1\}$ is called a* hard-core *of a function f if for every probabilistic polynomial-time algorithm A', every positive polynomial $p(\cdot)$, and all sufficiently large n's*

$$\mathrm{Pr}_{x\sim U_n}[A'(f(x)) = b(x)] < \frac{1}{2} + \frac{1}{p(n)}$$

Clearly, if b is a hard-core of a 1-1 polynomial-time computable function f then f must be one-way.[4] It turns out that any one-way function can be slightly modified so that it has a hard-core predicate.

Theorem 8 (A generic hard-core [13]): *Let f be an arbitrary one-way function, and let g be defined by $g(x, r) \stackrel{\text{def}}{=} (f(x), r)$, where $|x| = |r|$. Let $b(x, r)$ denote the inner-product mod 2 of the binary vectors x and r. Then the predicate b is a hard-core of the function g.*

See proof in [11, Apdx C.2]. We are now ready to present constructions of pseudorandom generators.

[4] Functions that are not 1-1 may have hard-core predicates of information-theoretic nature; but these are of no use to us here. For example, functions of the form $f(\sigma, x) = 0f'(x)$ (for $\sigma \in \{0,1\}$) have an "information theoretic" hard-core predicate $b(\sigma, x) = \sigma$.

3.1 The preferred presentation

In view of Theorem 5, we may focus on constructing pseudorandom generators with stretch function $\ell(n) = n+1$. Such a construction is presented next.

Proposition 9 (A simple construction of pseudorandom generators): *Let b be a hard-core predicate of a polynomial-time computable 1-1 function f. Then, $G(s) \stackrel{\rm def}{=} f(s)\,b(s)$ is a pseudorandom generator.*

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

In a sense, the key point in the above proof is showing that the unpredictability of the output of G implies its pseudorandomness. The fact that (next bit) unpredictability and pseudorandomness are equivalent in general is proven explicitly in the alternative presentation below.

3.2 An alternative presentation

The above presentation is different but analogous to the original construction of pseudorandom generators suggested by Blum and Micali [4]: Given an arbitrary stretch function $\ell:\mathbb{N}\rightarrow\mathbb{N}$, a 1-1 one-way function f with a hard-core b, one defines

$$G(s) \stackrel{\rm def}{=} b(x_0)b(x_1)\cdots b(x_{\ell(|s|)-1})\,,$$

where $x_0 = s$ and $x_i = f(x_{i-1})$ for $i = 1, ..., \ell(|s|)-1$. The pseudorandomness of G is established in two steps, using the notion of (next bit) unpredictability. An ensemble $\{Z_n\}_{n\in\mathbb{N}}$ is called unpredictable if any probabilistic polynomial-time machine obtaining a prefix of Z_n fails to predict the next bit of Z_n with probability non-negligibly higher than $1/2$.

Step 1: One first proves that the ensemble $\{G(U_n)\}_{n\in\mathbb{N}}$, where U_n is uniform over $\{0,1\}^n$, is (next-bit) unpredictable (from right to left) [4].

Loosely speaking, if one can predict $b(x_i)$ from $b(x_{i+1})\cdots b(x_{\ell(|s|)-1})$ then one can predict $b(x_i)$ given $f(x_i)$ (i.e., by computing $x_{i+1}, ..., x_{\ell(|s|)-1}$, and so obtaining $b(x_{i+1})\cdots b(x_{\ell(|s|)})$). But this contradicts the hard-core hypothesis.

Step 2: Next, one uses Yao's observation by which a (polynomial-time constructible) ensemble is *pseudorandom if and only if it is* (next-bit) *unpredictable* (cf. [10, Sec. 3.3.4]).

Clearly, if one can predict the next bit in an ensemble then one can distinguish this ensemble from the uniform ensemble (which in unpredictable regardless of computing power). However, here we need the other direction which is less obvious. Still, one can show that (next bit) unpredictability implies indistinguishability from the uniform ensemble. Specifically, consider the following "hybrid" distributions, where the $i^{\rm th}$ hybrid takes the first i bits from the questionable ensemble and the rest from the uniform one. Thus, distinguishing the extreme hybrids implies distinguishing some neighboring hybrids, which in turn implies next-bit predictability (of the questionable ensemble).

3.3 A general condition for the existence of pseudorandom generators

Recall that given any one-way 1-1 function, we can easily construct a pseudorandom generator. Actually, the 1-1 requirement may be dropped, but the currently known construction – for the general case – is quite complex. Still we do have.

Theorem 10 (On the existence of pseudorandom generators [18]):
Pseudorandom generators exist if and only if one-way functions exist.

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

The interesting direction is the construction of pseudorandom generators based on any one-way function. In general (when f may not be 1-1) the ensemble $f(U_n)$ may not be pseudorandom, and so Construction 9 (i.e., $G(s) = f(s)b(s)$, where b is a hard-core of f) cannot be used *directly*. One idea of [18] is to hash $f(U_n)$ to an almost uniform string of length related to its entropy, using Universal Hash Functions [5]. (This is done after guaranteeing, that the logarithm of the probability mass of a value of $f(U_n)$ is typically close to the entropy of $f(U_n)$.)[5] But "hashing $f(U_n)$ down to length comparable to the entropy" means shrinking the length of the output to, say, $n' < n$. This foils the entire point of stretching the n-bit seed. Thus, a second idea of [18] is to compensate for the $n - n'$ loss by extracting these many bits from the seed U_n itself. This is done by hashing U_n, and the point is that the $(n - n' + 1)$-bit long hash value does not make the inverting task any easier. Implementing these ideas turns out to be more difficult than it seems, and indeed an alternative construction would be most appreciated.

4 Pseudorandom Functions

Pseudorandom generators allow to efficiently generate long pseudorandom sequences from short random seeds. 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). Put 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 efficient machines that may obtain the function values at arguments of their choice. (Such machines are called oracle machines, and if M is such machine and f is a function, then $M^f(x)$ denotes the computation of M on input x when M's queries are answered by the function f.)

Definition 11 (pseudorandom functions [12]): *A* pseudorandom function (ensemble)*, with length parameters* $\ell_{\mathrm{D}}, \ell_{\mathrm{R}} : \mathbb{N} \to \mathbb{N}$*, is a collection of functions* $F \stackrel{\text{def}}{=} \{f_s : \{0,1\}^{\ell_{\mathrm{D}}(|s|)} \to \{0,1\}^{\ell_{\mathrm{R}}(|s|)}\}_{s \in \{0,1\}^*}$ *satisfying*

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

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

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

- (pseudorandomness): *For every probabilistic polynomial-time oracle machine, M, for every positive polynomial p and all sufficiently large n's*

$$\left|\Pr_{f\sim F_n}[M^f(1^n)=1]-\Pr_{\rho\sim R_n}[M^\rho(1^n)=1]\right| < \frac{1}{p(n)}$$

where F_n denotes the distribution on $f_s \in F$ obtained by selecting s uniformly in $\{0,1\}^n$, and R_n denotes the uniform distribution over all functions mapping $\{0,1\}^{\ell_{\rm D}(n)}$ to $\{0,1\}^{\ell_{\rm R}(n)}$.

Suppose, for simplicity, that $\ell_{\rm D}(n)=n$ and $\ell_{\rm R}(n)=1$. Then a function uniformly selected among 2^n functions (of a pseudorandom ensemble) presents an input-output behavior that is indistinguishable in poly(n)-time from the one of a function selected at random among all the 2^{2^n} Boolean functions. Contrast this with the 2^n pseudorandom sequences, produced by a pseudorandom generator, that are computationally indistinguishable from a sequence selected uniformly among all the $2^{\mathrm{poly}(n)}$ many sequences. Still pseudorandom functions can be constructed from any pseudorandom generator.

Theorem 12 (How to construct pseudorandom functions [12]): *Let G be a pseudorandom generator with stretching function $\ell(n)=2n$. Let $G_0(s)$ (resp., $G_1(s)$) denote the first (resp., last) $|s|$ bits in $G(s)$, and*

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

Then, the function ensemble $\{f_s:\{0,1\}^{|s|}\to\{0,1\}^{|s|}\}_{s\in\{0,1\}^*}$, where $f_s(x)\stackrel{\rm def}{=}G_x(s)$, is pseudorandom with length parameters $\ell_{\rm D}(n)=\ell_{\rm R}(n)=n$.

The above construction can be easily adapted to any (polynomially-bounded) length parameters $\ell_{\rm D},\ell_{\rm R}:\mathbb{N}\to\mathbb{N}$. We mention that pseudorandom functions have been used to derive negative results in computational learning theory [35] and in complexity theory (e.g., in the context of Natural Proofs [32]).

5 Further Discussion of Pseudorandom Generators

In this section we discuss some of the applications and conceptual aspects of pseudorandom generators.

5.1 The applicability of 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 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 (since generating perfectly random bits via special hardware is quite expensive). Thus, pseudorandom generators (as defined above) 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.

Indeed, "pseudo-random number generators" have appeared with the first computers. However, typical implementations use generators that are not pseudorandom according to the above definition. Instead, at best, these generators are shown to pass SOME ad-hoc statistical test (cf. [20]).

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 is good for a future (untested) application. Furthermore, the approach of subjecting the generator to some ad-hoc tests fails to provide general results of the type stated above (i.e., of the form "for ALL practical purposes using the output of the generator is as good as using truly unbiased coin tosses"). In contrast, the approach encompassed in Definition 2 aims at such generality, and in fact is tailored to obtain it: The notion of computational indistinguishability, which underlines Definition 2, covers all possible efficient applications postulating that for all of them pseudorandom sequences are as good as truly random ones.

Pseudorandom generators and functions are of key importance in Cryptography. They are typically used to establish private-key encryption and authentication schemes (cf. [11, Sec. 1.5.2 & 1.6.2]). For example, suppose that two parties share a random n-bit string, s, specifying a pseudorandom function (as in Definition 11), and that s is unknown to the adversary. Then, these parties may send encrypted messages to one another by XORing the message with the value of f_s at a random point. That is, to encrypt $m \in \{0,1\}^{\ell_{\mathrm{R}}(n)}$, the sender uniformly selects $r \in \{0,1\}^{\ell_{\mathrm{D}}(n)}$, and sends $(r, m \oplus f_s(r))$ to the receiver. Note that the security of this encryption scheme relies on the fact that, for *every* computationally-feasible adversary (not only to adversary strategies that were envisioned and tested), the values of the function f_s on such r's look random.

5.2 The intellectual contents of pseudorandom generators

We shortly discuss some intellectual aspects of pseudorandom generators as defined above.

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 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 [37, 15]. Thus, distributions that are ontologically very different, are considered equivalent by the behavioristic point of view taken in the definitions above.

A relativistic view of randomness. Pseudorandomness is defined above in terms of its observer. It is a distribution that cannot be told apart from a uniform distribution by any efficient (i.e. polynomial-time) observer. However, pseudorandom sequences may be distinguished from random ones by infinitely powerful computers (not at our disposal!). Specifically, an exponential-time machine can easily distinguish the output of a pseudorandom generator from a uniformly selected string of the same length (e.g., just by trying all possible seeds). Thus, pseudorandomness is subjective to the abilities of the observer.

Randomness and Computational Difficulty. Pseudorandomness and computational difficulty play dual roles: The definition of pseudorandomness relies on the fact that putting computational restrictions on the observer gives rise to distributions that are not uniform and still cannot be distinguished from uniform. Furthermore, the construction of pseudorandom generators rely on conjectures regarding computational difficulty (i.e., the existence of one-way functions), and this is inevitable: given a pseudorandom generator, we can construct one-way functions. Thus, (non-trivial) pseudorandomness and computational hardness can be converted back and forth.

6 A General Paradigm

Pseudorandomness as surveyed above can be viewed as an important special case of a general paradigm. A generic formulation of pseudorandom generators consists of specifying three fundamental aspects – the *stretching 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*). In the above presentation we focused on polynomial-time generators (thus having polynomial stretching measure) that fool any probabilistic polynomial-time observers. A variety of other cases are of interest too, and we briefly discuss some of them.

6.1 Weaker notions of computational indistinguishability

Whenever the aim is to replace random sequences utilized by an algorithm with pseudorandom ones, one may try to capitalize on knowledge of the target algorithm. Above we have merely used the fact that the target algorithm runs in polynomial-time. However, if the application utilizes randomness in a restricted way then feeding it with sequences of lower "randomness-quality" may do. For example, if we know that the algorithm uses very little work-space then we may use weaker forms of pseudorandom generators, which may be easier to construct, that suffice to fool bounded-space distinguishers. Similarly, very weak forms of pseudorandomness suffice for randomized algorithms that can be analyzed when only referring to some specific properties of the random sequence they uses (e.g., pairwise independence of elements of the sequence). In general, weaker notions of computational indistinguishability such as fooling space-bounded algorithms, constant-depth circuits, and even specific tests (e.g., testing pairwise independence of the sequence), arise naturally, and generators producing sequences that fool such distinguishers are useful in a variety of applications. Needless to say that we advocate a rigorous formulation of the characteristics of such applications and rigorous constructions of generators that fool the type of distinguishers that emerge. We mention some results of this type.

Fooling space-bounded algorithms. Here we consider space-bounded randomized algorithms that have on-line access to their random-tape, and so the potential distinguishers have on-line access to the input that they inspect. Two main results in this area are:

Theorem 13 ($\mathcal{RL} \subseteq \mathcal{SC}$ [26, 27]): *Any language decidable by a log-space randomized algorithm is decidable by a polynomial-time deterministic algorithm of poly-logarithmic space complexity.*

Theorem 14 (The Nisan–Zuckerman Generator [29]): *Any language decidable by a linear-space polynomial-time randomized algorithm is decidable by a randomized algorithm of the same complexities that uses only a linear number of coin tosses.*

Both theorems are actually special cases of more general results that refer to arbitrary computations (rather than to decision problems).

Fooling constant-depth circuits. As a special case, we consider the problem of approximately counting the number of satisfying assignments of a DNF formula. Put in other words, we wish to generate "pseudorandom" sequences that are as likely to satisfy a given DNF formula as uniformly selected sequences. Nisan showed that such "pseudorandom" sequences can be produced using seeds of polylogarithmic length [25]. By trying all possible seeds, one can approximately count the number of satisfying assignments of a DNF formula in deterministic quasi-polynomial time.

Pairwise independent generators. We consider distributions of n-long sequences over a finite set S. For $t \in \mathbb{N}$, such a distribution is called t-wise independent if its projection on any t coordinates yields a distribution that is uniform over S^t. We focus on the case where $|S|$ is a prime power, and so S can be identified with a finite field F. In such a case, given $1^n, 1^t$ and a representation of the field F so that $|F| > n$, one can generated a t-wise independent distribution over F^n in polynomial-time, using a random seed of length $t \cdot \log_2 |F|$. Specifically, the seed is used to specify a polynomial of degree $t-1$ over F, and the i^{th} element in the output sequence is the result of evaluating this polynomial at the i^{th} field element (cf. [2, 7]).

Small-bias generators. Here, we consider distributions of n-long sequences over $\{0,1\}$. For $\epsilon \in [0,1]$, such a distribution is called ϵ-bias if for every non-empty subset I, the exclusive-or of the bits at locations I equals 1 with probability at least $(1-\epsilon) \cdot \frac{1}{2}$ and at most $(1+\epsilon) \cdot \frac{1}{2}$.

Theorem 15 (small-bias generators [24]): *Given n and ϵ, one can generate an ϵ-bias distribution over $\{0,1\}^n$ in* $\text{poly}(n, \log(1/\epsilon))$*-time, using a random seed of length $O(\log(n/\epsilon))$.*

See [11, Sec. 3.6.2] for more details.

Samplers (and hitters) and extractors (and dispersers). Here we consider an arbitrary function $\nu : \{0,1\}^n \to [0,1]$, and seeks a universal procedure for approximating the average value of ν, denoted $\overline{\nu}$ (i.e., $\overline{\nu} \stackrel{\text{def}}{=} 2^{-n} \sum_x \nu(x)$). Such a (randomized) procedure is called a sampler. It is given three parameters, n, ϵ and δ, as well as oracle access to ν, and needs to output a value $\widetilde{\nu}$ so that $\Pr[|\overline{\nu} - \widetilde{\nu}| > \epsilon] < \delta$. A hitter is given the parameters, n, ϵ and δ, as well as a value v so that $|\{x : \nu(x) = v\}| > \epsilon \cdot 2^n$ (and oracle access to ν), and is required to find, with probability at least $1-\delta$, a preimage x so that $\nu(x) = v$. A sampler is called non-adaptive if it determines its queries based only on its internal coin tosses (i.e., independently on the answers obtained for previous queries); it is called oblivious if its output is a predetermined function of the sequence of oracle answers; and it is called averaging if its output equals the average value of the oracle answers. (In a sense, a non-adaptive sampler corresponds to a "pseudorandom generator" that produces at random a sequence of queries that, with high probability, needs to be "representative" of the average value of any function.) We focus on the randomness and query complexities of samplers, and mention that any sampler yields a hitter with identical complexities.

Theorem 16 (The Median-of-Averages Sampler [3]): *There exists a polynomial-time* (oblivious) *sampler of randomness complexity $O(n + \log(1/\delta))$ and query complexity $O(\epsilon^{-2} \log(1/\delta))$. Specifically, the sampler outputs the median value among $O(\log(1/\delta))$ values, where each of these values is the average of $O(\epsilon^{-2})$ distinct oracle answers.*[6]

[6] Each of the $O(\log(1/\delta))$ sequences of $O(\epsilon^{-2})$ queries is produced by a pairwise independent generator, and the seeds used for these different sequences are generated by a random walk on an expander graph (cf. [1] and [11, Sec. 3.6.3]).

The randomness complexity can be further reduced to $n+O(\log(1/\epsilon\delta))$, and both complexities are optimal up-to a constant multiplicative factor; see [11, Sec. 3.6.4]. Averaging samplers are closely related to extractors, but the study of the latter tends to focus more closely on the randomness complexity (and allow query complexity that is polynomial in the above).[7] A function $E:\{0,1\}^n \times \{0,1\}^t \to \{0,1\}^m$ is called a (k,ϵ)-extractor if for any random variable X so that $\max_x\{\Pr[X=x]\} \leq 2^{-k}$ it holds that the statistical difference between $E(X,U_t)$ and U_m is at most ϵ, where U_t and U_m are independently and uniformly distributed over $\{0,1\}^t$ and $\{0,1\}^m$, respectively. (An averaging sampler of randomness complexity $r(m,\epsilon,\delta)$ and query complexity $q(m,\epsilon,\delta)$ corresponds to an extractor in which (the yet unspecified parameters are) $n=r(m,\epsilon,\delta)$, $t=\log_2 q(m,\epsilon,\delta)$, and $k=n-\log_2(1/\delta)$.) A landmark in the study of extractors is the following

Theorem 17 (Trevisan's Extractor [36]): *For any $a,b>0$, let $k(n)=n^a$ and $m(n)=\lceil k(n)^{1-b}\rceil$. For $t(n)=O(\log n)$ and $\epsilon(n)>1/\mathrm{poly}(n)$, there exists a polynomial-time computable family of functions $\{E_n:\{0,1\}^n \times \{0,1\}^{t(n)} \to \{0,1\}^{m(n)}\}_{n\in\mathbb{N}}$ so that E_n is an $(k(n),\epsilon(n))$-extractor.*

The theorem is proved by reducing the construction of extractors to the construction of certain pseudorandom generators (considered in the next subsection). The reduction is further discussed at the end of the next subsection.

6.2 Alternative notions of generator efficiency

The above discussion has focused on one aspect of the pseudorandomness question – the resources or type of the observer (or potential distinguisher). Another important question is *at what cost* can pseudorandom sequences be generated (from much shorter seeds, assuming this is at all possible). Throughout this survey we have required the generation process to be at least as efficient as the efficiency limitations of the distinguisher.[8] This seems indeed "fair" and natural. Allowing the generator to be more complex (i.e., use more time or space resources) than the distinguisher seems unfair (and is typically unreasonable in the context of cryptography), but still yields interesting consequences in the context of "de-randomization" (i.e., transforming randomized algorithms into equivalent deterministic algorithms (of slightly higher complexity)). For example, one may consider generators working in time exponential in the length of the seed. As observed by Nisan and Wigderson [28], in some cases we lose nothing by being more liberal (i.e., allowing exponential-time generators). To see why, we consider a typical de-randomization argument, proceeding in two steps: First one replaces the true randomness of the algorithm by pseudorandom sequences generated from much shorter seeds, and next one goes deterministically over all possible seeds and looks for the most frequent behavior of the modified algorithm. Thus, in such a case the deterministic complexity is anyhow exponential in the seed length. The benefit of allowing exponential-time generators is that constructing exponential-time generators may be easier than constructing polynomial-time ones. A typical result in this vein follows.

Theorem 18 (De-randomization of BPP [19] (building upon [28])): *Suppose that there exists a language $L\in\mathcal{E}$ having almost-everywhere exponential circuit complexity.*[9] *Then, $\mathcal{BPP}=\mathcal{P}$.*

[7] The relation between hitters and dispersers is analogous.

[8] If fact, we have required the generator to be more efficient than the distinguisher: The former was required to be a fixed polynomial-time algorithm, whereas the latter was allowed to be any algorithm with polynomial running time.

[9] We say that L is in $\mathcal{E}$ if there exists an exponential time algorithm for deciding L; that is, the running-time of the algorithm on input x is at most $2^{O(|x|)}$. By saying that L has almost-everywhere exponential circuit complexity we mean that there exists a constant $b>0$ such that, for all but finitely many k's, any circuit C_k that correctly decides L on $\{0,1\}^k$ has size at least 2^{bk}.

Proof Sketch: Underlying the proof is a construction of a pseudorandom generator due to Nisan and Wigderson [25, 28]. This construction utilizes a predicate computable in exponential-time but unpredictable, even to within a particular exponential advantage, by any circuit family of a particular exponential size. (The crux of [19] is in supplying such a predicate, given the hypothesis.) Given such a predicate the generator works by evaluating the predicate on exponentially-many subsequences of the bits of the seed so that the intersection of any two subsets is relatively small.[10] Thus, for some constant $b > 0$ and all k's, the generator stretches seeds of length k into sequences of length 2^{bk} that (as loosely argued below) cannot be distinguished from truly random sequences by any circuit of size 2^{bk}. The de-randomization of $\mathcal{BPP}$ proceeds by setting the seed-length to be logarithmic in the input length, and utilizing the above generator.

The above generator fools circuits of the stated size, even when these circuits are presented with the seed as auxiliary input. (These circuits are smaller than the running time of the generator and so they cannot just evaluate the generator on the given seed.) The proof that the generator fools such circuits refers to the characterization of pseudorandom sequences as unpredictable ones. Thus, one proves that the next bit in the generator's output cannot be predicted given all previous bits (as well as the seed). Assuming that a small circuit can predict the next bit of the generator, we construct a circuit for predicting the hard predicate. The new circuit incorporates the best (for such prediction) augmentation of the input to the circuit into a seed for the generator (i.e., the bits not in the specific subset of the seed are fixed in the best way). The key observation is that all other bits in the output of the generator depend only on a small fraction of the input bits (i.e., recall the small intersection clause above), and so circuits for computing these other bits have relatively small size (and so can be incorporated in the new circuit). Using all these circuits, the new circuit forms the adequate input for the next-bit predicting circuit, and outputs whatever the latter circuit does. ■

Connection to extractors. Trevisan's construction [36] adapts the computational framework underlying the Nisan–Wigderson Generator [28] to the information-theoretic context of extractors. His adaptation is based on two key observations. The first observation is that the generator itself uses a (supposedly hard) predicate as a black-box. Trevisan's construction utilizes a "random" predicate which is encoded by the first input to the extractor. For example, the n-bit input may encode a predicate on $\log_2 n$ bits in the obvious manner. The second input to the extractor, having length $t = O(\log n)$, will be used as the seed to the resulting generator (defined by using this random predicate in a black-box manner). The second key observation is that the proof of indistinguishability of the generator provides a black-box procedure for computing the underlying predicate when given oracle access to a distinguisher. Thus, any subset $S \subset \{0,1\}^m$ of the possible outputs of the extractor gives rise to a relatively small set P_S of predicates, so that for each value $x \in \{0,1\}^n$ of the first input to the extractor, if S "distinguishes" the output of the extractor (on a random second input) from the uniform distribution then one of the predicates in P_S equals the predicate associated with x. It follows that for every set S, the set of possible first inputs for which the probability that the extractor hits S does not approximate the density of S is small. This establishes the extraction property.

[10] These subsets have size linear in the length of the seed, and intersect on a constant fraction of their respective size. Furthermore, they can be determined within exponential-time.

References

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

[2] 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.

[3] M. Bellare, O. Goldreich, and S. Goldwasser. Randomness in Interactive Proofs. *Computational Complexity*, Vol. 4, No. 4, pages 319–354, 1993.

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

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

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

[7] 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.

[8] B. Chor and O. Goldreich. Unbiased Bits from Sources of Weak Randomness and Probabilistic Communication Complexity. *SIAM Journal on Computing*, Vol. 17, No. 2, pages 230–261, 1988.

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

[10] O. Goldreich. *Foundation of Cryptography – Fragments of a Book.* February 1995. Available from `http://theory.lcs.mit.edu/~oded/frag.html`.

[11] O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness.* Algorithms and Combinatorics series (Vol. 17), *Springer*, 1998.

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

[13] 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.

[14] O. Goldreich and S. Micali. Increasing the Expansion of Pseudorandom Generators. Unpublished manuscript, 1984.

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

[16] O. Goldreich, H. Krawcyzk and M. Luby. On the Existence of Pseudorandom Generators. *SIAM Journal on Computing*, Vol. 22-6, pages 1163–1175, 1993.

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

[18] 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 ACM Symposium on the Theory of Computing* (1989) and Håstad in *22nd ACM Symposium on the Theory of Computing* (1990).

[19] 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.

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

[21] A. Kolmogorov. Three Approaches to the Concept of "The Amount Of Information". *Probl. of Inform. Transm.*, Vol. 1/1, 1965.

[22] L.A. Levin. Randomness Conservation Inequalities: Information and Independence in Mathematical Theories. *Inform. and Control*, Vol. 61, pages 15–37, 1984.

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

[24] J. Naor and M. Naor. Small-bias Probability Spaces: Efficient Constructions and Applications. *SIAM J. on Computing*, Vol 22, 1993, pages 838–856.

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

[26] N. Nisan. Pseudorandom Generators for Space Bounded Computation. *Combinatorica*, Vol. 12 (4), pages 449–461, 1992.

[27] N. Nisan. $\mathcal{RL} \subseteq \mathcal{SC}$. *Journal of Computational Complexity*, Vol. 4, pages 1-11, 1994.

[28] N. Nisan and A. Wigderson. Hardness vs Randomness. *Journal of Computer and System Science*, Vol. 49, No. 2, pages 149–167, 1994.

[29] N. Nisan and D. Zuckerman. Randomness is Linear in Space. *Journal of Computer and System Science*, Vol. 52 (1), pages 43–52, 1996.

[30] A.M. Odlyzko. The future of integer factorization. *CryptoBytes* (The technical newsletter of RSA Laboratories), Vol. 1 (No. 2), pages 5-12, 1995. Available from `http://www.research.att.com/~amo`

[31] A.M. Odlyzko. Discrete logarithms and smooth polynomials. In *Finite Fields: Theory, Applications and Algorithms*, G. L. Mullen and P. Shiue, eds., Amer. Math. Soc., Contemporary Math. Vol. 168, pages 269–278, 1994. Available from `http://www.research.att.com/~amo`

[32] A.R. Razborov and S. Rudich. Natural proofs. *Journal of Computer and System Science*, Vol. 55 (1), pages 24–35, 1997.

[33] C.E. Shannon. A mathematical theory of communication. *Bell Sys. Tech. Jour.*, Vol. 27, pages 623–656, 1948.

[34] R.J. Solomonoff. A Formal Theory of Inductive Inference. *Inform. and Control*, Vol. 7/1, pages 1–22, 1964.

[35] L. Valiant. A theory of the learnable. *Communications of the ACM*, Vol. 27/11, pages 1134–1142, 1984.

[36] L. Trevisan. Constructions of Near-Optimal Extractors Using Pseudo-Random Generators. In *31st ACM Symposium on the Theory of Computing*, pages 141–148, 1998.

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