Pseudorandomness

Oded Goldreich

This essay considers 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, which 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.

The second half of this century has witnessed the development of three theories of randomness, a notion that has been puzzling thinkers over the ages. The first theory (cf. [3]), initiated by Shannon, is rooted in probability theory and is focused on 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. [11, 12]), due to Solomonov, Kolmogorov, and Chaitin, is rooted in computability theory and specifically in the notion of a universal language (equivalently, 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 [8, 2, 13], is rooted in complexity theory and is the focus of this essay. This approach is explicitly aimed at providing a theory of perfect randomness that nevertheless allows for the efficient generation of 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

Oded Goldreich is professor of computer science at the Weizmann Institute of Science, Israel. His e-mail address is oded@wisdom.weizmann.ac.il.

The author is grateful to Anthony Knapp and Susan Landau for their many useful comments.

[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* $f:\{0,1\}^n \to \{0,1\}^m$ *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 encoding of* n *(say, in binary)). In contrast, a simple counting argument shows that most* n*-bit strings have Kolmogorov complexity at least* n.

procedure. Consequently a distribution that cannot be efficiently distinguished from the uniform distribution will be considered as being random (or rather "random for all practical purposes", which we call "pseudorandom"). Thus, randomness is not an "inherent" property of objects (or distributions) but rather is 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 affecting 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.

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 an 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.

The Definition of Pseudorandom Generators

Loosely speaking, a pseudorandom generator is an *efficient* program (or algorithm) that *stretches* short random strings into long *pseudorandom* sequences. We emphasize 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 that is the heart of the entire approach.

To demonstrate the above, consider the following suggestion for a pseudorandom generator. The seed consists of a pair of 32-bit integers, x and N, and the 100,000-bit output is obtained by repeatedly squaring the current x modulo N and emitting the least significant bit of each intermediate result (i.e., let $x_i \leftarrow x_{i-1}^2 \bmod N$, for $i = 1, \ldots, 10^5$, and output $b_1, b_2, \ldots, b_{10^5}$, where $x_0 \overset{\text{def}}{=} x$ and b_i is the least significant bit of x_i). This process may be generalized to seeds of length n (here we used $n = 64$) and outputs of length $\ell(n)$ (here $l(n) = 10^5$). Such a process certainly satisfies items (1) and (2) above, whereas the question whether item (3) holds is debatable (once a rigorous definition is provided). Anticipating some of the discussion below, we mention that, under the assumption that it is difficult to factor large integers, a slight variant of the above process is indeed a pseudorandom generator.

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 $\ell : \mathbb{N} \to \mathbb{N}$, 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 [8, 13]). *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, any positive polynomial p, and all sufficiently large n*

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

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

A few comments are in order. First, we have allowed algorithm A, which is called a distinguisher, to be probabilistic. This makes the requirement only stronger, and seems essential to several important aspects of our approach. Second, we view events occurring 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 when the experiment is repeated for poly(n)-many times. Third, when allowing A in the above definition to be an arbitrary function (rather than a probabilistic polynomial-time algorithm), one obtains a notion of statistical indistinguishability. The latter is equivalent to requiring that the variation distance between X_n and Y_n (i.e., $\sum_z |X_n(z) - Y_n(z)|$) is negligible (in n).

We note that computational indistinguishability is a strictly more liberal notion than statistical indistinguishability (cf. [13]). An important case is the one of distributions generated by a pseudorandom generator (as defined next): such distributions are computationally indistinguishable from uniform but are not statistically indistinguishable from uniform.

Definition 2. (pseudorandom generators [2, 13]). *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, any positive polynomial p, and all sufficiently large n

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

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 let $\rho(n)$ denote an upper bound on its* randomness complexity *(i.e., number of coins that A tosses on n-bit inputs). Let $A(x,r)$ denote the output of A on input x and coin tossing 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. This is formulated in the proposition below, where F represents an algorithm trying to find x's so that $A(x)$ and $A_G(x)$ are distinguishable (by an algorithm D).

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, i.e., $\Delta_{A,D}(x)$ is defined as*

[3] *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.*

$$\left| \Pr_{r\sim U_{\rho(n)}}[D(x, A(x,r)) = 1] - \Pr_{s\sim U_{k(n)}}[D(x, A_G(x,s)) = 1] \right| ,$$

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 F and D, every positive polynomial p, and all sufficiently long n

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

where 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 multiparty 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.*

Amplifying the Stretch Function

Pseudorandom generators as defined above are required only to stretch their input a bit; for example, stretching n-bit long inputs to $(n+1)$-bit long outputs will do. Clearly, generators with such moderate stretch functions are of little use in practice. In contrast, we want to have pseudorandom generators with an arbitrarily 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 specific stretch function.)

Theorem 5. [5, Sec. 3.3.2]. *Let G be a pseudorandom generator with stretch function $\ell(n) = n + 1$, and let ℓ' be any stretch function such that $\ell'(n)$ is computable in poly(n)-time from n. Let $G_1(x)$ denote the $|x|$-bit long prefix of $G(x)$, and let $G_2(x)$ denote the last bit of $G(x)$ (i.e., $G(x) = G_1(x)\,G_2(x)$). Then*

$$G'(s) \overset{\text{def}}{=} \sigma_1\sigma_2\cdots\sigma_{\ell'(|s|)} ,$$
$$\text{where } x_0 = s, \sigma_i = G_2(x_{i-1}), \text{ and } x_i = G_1(x_{i-1}),$$
$$\text{for } i = 1, \ldots, \ell'(|s|)$$

is a pseudorandom generator with stretch function ℓ'.

How to Construct Pseudorandom Generators

The known constructions transform computational 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*

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

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, the discrete logarithm problem, and decoding of random linear code. The infeasibility of inverting f yields a weak notion of unpredictability: Let $b_i(x)$ denote the i^{th} bit of x. Then, for every probabilistic polynomial-time algorithm A (and sufficiently large n), it must be the case that $\Pr_{i,x}[A(i, f(x)) \neq b_i(x)] > 1/2n$, where the probability is taken uniformly over $i \in \{1, \ldots, n\}$ and $x \in \{0,1\}^n$. A stronger (and in fact the 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 $b(x)$ with probability of success that is only negligibly better than one half.

Definition 7. (hard-core predicate [2]). *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*

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

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.

[4]*Functions that are not 1-1 may have hard-core predicates of an 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 $b(\sigma, x) = \sigma$.*

Theorem 8. (a generic hard-core [7]). *Let f be an arbitrary one-way function, and let g be defined by $g(x,r) \overset{\text{def}}{=} (f(x),r)$, 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.*

A proof may be found in [5, Appen. C.2]. Finally, we get to the construction of pseudorandom generators.

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, $G(s) \overset{\text{def}}{=} f(s)\,b(s)$ (i.e., $f(s)$ followed by $b(s)$) is a pseudorandom generator.*

In a sense, the key point in the proof of the above proposition is showing that the (obvious by definition) 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.

An Alternative Presentation

Our presentation of the construction of pseudorandom generators, via Theorem 5 and Proposition 9, is different but analogous to the original construction of pseudorandom generators suggested by Blum and Micali [2]: Given an arbitrary stretch function $\ell : \mathbb{N} \to \mathbb{N}$ and a 1-1 one-way function f with a hard-core b, one defines

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

where $x_0 = s$ and $x_i = f(x_{i-1})$ for $i = 1,\ldots,\ell(|s|)-1$. A concrete instantiation, based on the assumption that it is difficult to factor large integers, is depicted in Figure 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 nonnegligibly 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) [2]. 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 $f(x_i)$ (namely, by computing $x_{i+1},\ldots,x_{\ell(|s|)-1}$ and so obtaining $b(x_{i+1})\cdots b(x_{\ell(|s|)-1})$). But this contradicts the hard-core hypothesis.

Step 2. Next one uses Yao's observation by which an ensemble is *pseudorandom if and only if it is* (next-bit) *unpredictable* (cf. [4, 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 is unpredictable regardless of computing power). However, here we need the other, less obvious direction. One can show that (next-bit) unpredictability implies indistinguishability from the uniform ensemble. Specifically, consider the "hybrid" distributions in which the i^{th} hybrid takes the first i bits from the ensemble in question 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 ensemble in question).

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 [9]). *Pseudorandom generators exist if and only if one-way functions exist.*

To show that the existence of pseudorandom generators implies 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 $f(x,y) \overset{\text{def}}{=} G(x)$ so that 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.

We assume that it is infeasible to factor integers that are the product of two large primes (each congruent to 3 mod 4). Under this assumption, squaring modulo such integers is a one-way function. Furthermore, squaring modulo such N is 1-1 over the quadratic residues mod N, and the least significant bit of the argument is a corresponding hard-core [1].

The following pseudorandom generator uses a polynomial-time algorithm that when fed with $4m$ bits generates an m-bit number so that when the input $4m$ bits are uniformly distributed, the output is essentially an m-bit random prime (congruent to 3 mod 4).

```
Input: An $n$-bit seed $s = abc$, where $|a| = |b| = 4n/10$.
Initialization Steps:
    1. Use $a$ to produce an $n/10$-bit prime $p \equiv 3 \bmod 4$.
    2. Similarly, use $b$ to produce $q \equiv 3 \bmod 4$.
    3. Multiply $p$ and $q$, obtaining $N$.
    4. Let $x_0 \leftarrow c^2 \bmod N$.
Iterations: For $i = 0,\ldots,\ell(n)-1$
    5. Let $b_i$ be the least significant bit of $x_i$.
    6. Let $x_{i+1} \leftarrow x_i^2 \bmod N$.
Output: $b_0, b_1, .., b_{\ell(n)-1}$
```

Figure 1. A pseudorandom generator based on the intractability of factoring.

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*. Instead, this construction is used together with a couple of other ideas (cf. [9]). Unfortunately, these ideas and more so the details of implementing them are far too complex to be described here. Indeed, an alternative construction of pseudorandom generators based on any one-way function would be most appreciated.

Pseudorandom Functions

Pseudorandom generators allow one 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). 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 a 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 [6]). *A* pseudorandom function (ensemble), *with length parameters $\ell_{\mathrm{D}}, \ell_{\mathrm{R}} : \mathbb{N} \to \mathbb{N}$, is a collection of functions*

$$F \overset{\text{def}}{=} \{f_s : \{0,1\}^{\ell_{\mathrm{D}}(|s|)} \to \{0,1\}^{\ell_{\mathrm{R}}(|s|)}\}_{s \in \{0,1\}^*}$$

satisfying

- (efficient evaluation). *There exists an efficient (deterministic) algorithm that 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 n*

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

  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 $\{0,1\}^{\ell_{\mathrm{D}}(n)}$ to $\{0,1\}^{\ell_{\mathrm{R}}(n)}$.

Suppose, for simplicity, that $\ell_{\mathrm{D}}(n) = n$ and $\ell_{\mathrm{R}}(n) = 1$. Then a function uniformly selected among 2^n functions (of a pseudorandom ensemble) presents an input-output behavior 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 a distribution over 2^n sequences, produced by a pseudorandom generator applied to a random n-bit seed, that is 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 [6]). *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 let*

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

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

The above construction can be easily adapted to any (polynomially bounded) length parameters $\ell_{\mathrm{D}}, \ell_{\mathrm{R}} : \mathbb{N} \to \mathbb{N}$.

We mention that pseudorandom functions have been used to derive negative results in computational learning theory and in complexity theory.

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 use 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 use of randomness into programs that make an economical use of randomness.

Indeed, "pseudo-random number generators" 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. [10]). However, 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. [5, 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 with $\ell_D(n) = \ell_R(n) = n$), 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 $m \in \{0,1\}^n$, the sender uniformly selects $r \in \{0,1\}^n$ and sends $(r, m \oplus f_s(r))$ to the receiver. 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.

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. 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, dependent on 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 relies 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, nontrivial pseudorandomness and computational difficulty can be converted back and forth.

Generalization

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. For more details see [5].

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 we know for example, that the algorithm uses very little workspace, then we may be able to do better. Similarly we may be able to do better if we know that the analysis of the algorithm depends only on some specific properties of the random sequence it uses (e.g., pairwise independence of its elements). 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. Generators producing sequences that fool such tests are useful in a variety of applications; if the application utilizes randomness in a restricted way, then feeding it with sequences of low randomness quality may do. Needless to say, the author advocates a rigorous formulation of the characteristics of such applications and rigorous constructions of generators that fool the type of tests that emerge.

Alternative Notions of Generator Efficiency

The previous paragraph has focused on one aspect of the pseudorandomness question: the resources or type of the observer (or potential distinguisher). Another important question is whether such pseudorandom sequences can be generated from much shorter ones and at what cost (or complexity). Throughout this essay we have required the generation process to be at least as efficient as the efficiency limitations of the distinguisher.[5] 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 but still yields interesting consequences in the context of trying to "de-randomize" randomized complexity classes. For example, one may consider generators working in time exponential in the length of the seed. In some cases we lose nothing by using such a relaxation (i.e., allowing exponential-time generators). To see why, we consider a typical derandomization 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. In such a case the deterministic complexity is anyhow exponential in the seed length. The benefit is that constructing exponential-time generators may be easier than constructing polynomial-time ones.

References

[1] W. Alexi, B. Chor, O. Goldreich, and C. P. Schnorr, RSA/Rabin functions: Certain parts are as hard as the whole, *SIAM J. Comput.* **17** (1988), 194–209.

[2] M. Blum and S. Micali, How to generate cryptographically strong sequences of pseudo-random bits, *SIAM J. Comput.* **13** (1984), 850–864.

[3] T. M. Cover and G. A. Thomas, *Elements of Information Theory*, Wiley, New York, 1991.

[4] O. Goldreich, *Foundation of Cryptography—Fragments of a Book*, February 1995, available from `http://theory.lcs.mit.edu/~oded/frag.html`.

[5] ____, *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*, Algorithms and Combinatorics, vol. 17, Springer-Verlag, New York, 1998.

[6] O. Goldreich, S. Goldwasser, and S. Micali, How to construct random functions, *J. ACM* **33** (1986), 792–807.

[7] O. Goldreich and L. A. Levin, Hard-core predicates for any one-way function, *21st ACM Symposium on the Theory of Computing*, 1989, pp. 25–32.

[8] S. Goldwasser and S. Micali, Probabilistic encryption, *J. Comput. System Sci.* **28** (1984), 270–299.

[9] J. Hastad, R. Impagliazzo, L. A. Levin, and M. Luby, A pseudorandom generator from any one-way function, *SIAM J. Comput.* **28** (1999), 1364–1396.

[10] D. E. Knuth, *Seminumerical Algorithm*s, The Art of Computer Programming, Vol. 2, Addison-Wesley, Reading, MA,1969; second edition, 1981.

[11] L. A. Levin, Randomness conservation inequalities: Information and independence in mathematical theories, *Inform. and Control* **61** (1984), 15–37.

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

[13] A. C. Yao, Theory and application of trapdoor functions, *23rd IEEE Symposium on Foundations of Computer Science*, 1982, pp. 80–91.

[5]*In 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.*