Lecture Notes on Pseudorandomness – Part I

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

January 23, 2001

Abstract

A fresh view at the *question of randomness* was taken in the theory of computing: It has been postulated that a distribution is pseudorandom if it cannot be told apart from the uniform distribution by any efficient procedure. This paradigm, originally associating efficient procedures with polynomial-time algorithms, has been applied also with respect to a variety of other classes of distinguishing procedures. We focus on pseudorandom generators; that is, deterministic programs that stretch short (random) seeds into much longer pseudorandom sequences.

The current lecture series focuses on the case where the pseudorandom generator runs in polynomial-time and withstands any polynomial-time distinguisher. In particular, the distinguisher may be more complex (i.e., have a higher polynomial running time) than the generator. This framework is natural in the context of designing general-purpose pseudorandom generators that can be used in *any* efficient (i.e., polynomial-time) application. Furthermore, this framework is almost mandatory in cryptographic applications, where the adversary is typically willing to invest more effort than the legitimate users.

A companion lecture series (i.e., "Pseudorandomness – Part 2" by Luca Trevisan) focuses on the case where the pseudorandom generator runs in exponential-time (w.r.t the seed length) and withstands distinguisher of running time bounded by a specific polynomial (in the length of the generator's output). In particular, the generator may be more complex than the distinguisher. As explained in the companion lecture series, this framework is natural in the context of derandomization (i.e., converting randomized algorithms to deterministic ones).

Contents

Preface

The heart of the theory of pseudorandomness is the suggestion to consider the class of distributions that cannot be told apart from the uniform distribution by any efficient procedure. These distributions are called pseudorandom, and can be generated deterministically from short random seeds (which are much shorter than the length of the pseudorandom output). Specifically, the ability to efficiently generate pseudorandom objects is closely related to computational difficulty.

In this lecture series, we consider a specific instantiation of the paradigm outlined by the above paragraph, associating efficient computation with polynomial-time algorithms. We believe that this instantiation (of the above paradigm) is the most natural one, but other instantiations are important too (see remarks in Section 3.3.3). In particular, in the next lecture series, a different instantiation of the above paradigm is considered.

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, and we consider the probability distribution of the output of the algorithm, where the probability space is taken over these coin tosses. That is, if A_1 is such an algorithm and $x \in \{0,1\}^*$, then $A_1(x)$ denotes the output distribution of A_1 on input x. An equivalent way of looking at a *probabilistic* algorithm is to view its random steps as determined by a sequence of outcomes supplied from the outside. In this case, we consider a two-input deterministic algorithm, where the first input is the actual input and the second input represents the outcomes of a sequence of coin tosses, and refer to the distribution induced on the output when fixing the first input and letting the second input be random. That is, if A_2 is such an algorithm and $x \in \{0,1\}^*$, then $A_2(x,r)$ denotes the output of A_2 on actual input x and random input r, and we consider the distribution of $A_2(x,\cdot)$ when the second input is uniformly selected in $\{0,1\}^t$, for some sufficiently large t.

Background material and further reading

We assume the reader is fairly comfortable with basic notions of the theory of computation (see, e.g., [8, 32]) and elementary probability theory (see, e.g., [7]). Familiarity with some randomized algorithms may be useful too (see, e.g., [27] or [10, Apdx. B]).

The interested reader is directed to two texts by the author: Chapter 3 of [11], which is a textbook devoted to the foundations of cryptography, provides a detailed treatment of the subject matter of the current lecture series (including full proofs for results for which only proof sketches appear below). On the other hand, Chapter 3 of [10] provides a much wider perspective on pseudorandomness (but less details on the subject matter of the current lecture series). Other parts of [10] survey related areas such as probabilistic proof systems, cryptography, and randomized computation.

Lecture 1

Computational Indistinguishability

1.1 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. [5]), initiated by Shannon [31], 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. [23, 26]), due to Solomonov [33], Kolmogorov [22] and Chaitin [4], 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 [19, 2, 35], is rooted in complexity theory and is the focus of this lecture series. This approach is explicitly aimed at providing a 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.

[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 at most $O(1) + \log_2 n$, 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).

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.

1.2 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.

1.3 The Definition of 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. Consequently, for a probabilistic algorithm A, we denote by $\Pr_{e\sim D_n}[A(e)=1]$ the probability that $A(e)=1$ holds when e is distributed (or selected) according to D_n.

Definition 1.1 (Computational Indistinguishability [19, 35]) *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*

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

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.

1.4 Relation to Statistical Closeness

Computational indistinguishability is a (strict) coarsening of a traditional notion from probability theory. We call two ensembles $X \stackrel{\rm def}{=} \{X_n\}_{n\in\mathbb{N}}$ and $Y \stackrel{\rm def}{=} \{Y_n\}_{n\in\mathbb{N}}$, statistically close if their statistical difference is negligible, where the statistical difference (also known as variation distance) of X and Y is defined as the function

$$\Delta(n) \stackrel{\rm def}{=} \frac{1}{2}\cdot\sum_{\alpha}|\Pr[X_n\!=\!\alpha]-\Pr[Y_n\!=\!\alpha]| \tag{1.1}$$

Clearly, if the ensembles X and Y are statistically close then they are also computationally indistinguishable[4]. The converse, however, is not true. In particular, letting U_n denote the uniform distribution over strings of length n, we have:

Proposition 1.2 ([35, 13]) *There exist an ensemble $X=\{X_n\}_{n\in\mathbb{N}}$ so that X is* NOT *statistically close to the uniform ensemble, $U \stackrel{\rm def}{=} \{U_n\}_{n\in\mathbb{N}}$, and yet X and U are computationally indistinguishable. Furthermore, X_n assigns all its probability mass to at most $2^{n/2}$ strings (of length 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.

[4] The proof of this claim is left as an exercise.

Although X and U are computationally indistinguishable, one can define a function $f:\{0,1\}^* \to \{0,1\}$ so that f has average 1 over X while having average almost 0 over U (e.g., $f(x)=1$ if and only if $\Pr[X=x]>0$). Hence, X and U have different "profile" with respect to the function f, yet f is (necessarily) impossible to compute in polynomial-time.

Proposition 1.2 presents a pair of ensembles that are computational indistinguishable although they are statistically far apart. One of the two ensembles is not constructible in polynomial-time (see Definition 1.4 below). Interestingly, a pair of polynomial-time constructible ensembles being both computationally indistinguishable and having a noticeable statistical difference exists only if pseudorandom generators exist (cf. [9]). Jumping ahead, we note that this necessary condition is also sufficient. (The latter observation follows from the fact that pseudorandom generators give rise to a polynomial-time constructible ensemble that is computationally indistinguishable from the uniform ensemble and yet is statistically far from it.)[5]

Proof of Proposition 1.2: We *claim* that, for all sufficiently large n, there exist a random variable X_n, distributed over some set of at most $2^{n/2}$ strings (each of length n), so that for every circuit, C_n, of size (i.e., number of gates) $2^{n/8}$ it holds that

$$|\Pr[C_n(U_n)=1] - \Pr[C_n(X_n)=1]| < 2^{-n/8} \tag{1.2}$$

The proposition follows from this claim, since polynomial-time distinguishers (even probabilistic ones) yield polynomial-size circuits with at least as big a distinguishing gap.

The above claim is proven using a probabilistic argument. That is, we actually show that most distributions of a certain class "fool" all circuits of size $2^{n/8}$. Specifically, we show that if we select uniformly a multi-set of $2^{n/2}$ strings in $\{0,1\}^n$, and let X_n be uniform over this multi-set then Eq. (1.2) holds with overwhelmingly high probability (over the choices of the multi-set).

Let C_n be some fixed circuit with n inputs, and let $p_n \stackrel{\rm def}{=} \Pr[C_n(U_n)=1]$. We select, independently and uniformly $2^{n/2}$ strings, denoted $s_1,...,s_{2^{n/2}}$, in $\{0,1\}^n$. Define random variables ζ_i's so that $\zeta_i = C_n(s_i)$; i.e., these random variables depend on the random choices of the corresponding s_i's. Using Chernoff Bound, we get that

$$\Pr\left[\left|p_n - \frac{1}{2^{n/2}}\cdot\sum_{i=1}^{2^{n/2}}\zeta_i\right| \geq 2^{-n/8}\right] \;\leq\; 2e^{-2\cdot 2^{n/2}\cdot(2^{-n/8})^2} \;<\; 2^{-2^{n/4}}$$

Since there are at most $2^{2^{n/4}}$ different circuits of size (number of gates) $2^{n/8}$, it follows that there exists a sequence of $s_1,...,s_{2^{n/2}} \in \{0,1\}^n$, so that for every circuit C_n of size $2^{n/8}$ it holds that

$$\left|\Pr[C_n(U_n)=1] - \frac{\sum_{i=1}^{2^{n/2}} C_n(s_i)}{2^{n/2}}\right| \;<\; 2^{-n/8}$$

Fixing such a sequence of s_i's, and letting X_n be distributed uniformly over the elements in the sequence, the claim follows. ■

1.5 Indistinguishability by Repeated Experiments

By Definition 1.1, two ensembles are considered computationally indistinguishable if no efficient procedure can tell them apart based on a single sample. We now show that, for "efficiently con-

[5] The reader may want to rigorously verify this claim after inspecting the definition of a pseudorandom generator (i.e., Definition 2.1).

structible" ensembles, computational indistinguishability (based on a single sample) implies computational indistinguishability based on multiple samples. We start by presenting definitions of "indistinguishability by multiple samples" and "efficiently constructible ensembles".

Definition 1.3 (indistinguishability by repeated sampling) *Two ensembles, $X \stackrel{\rm def}{=} \{X_n\}_{n\in\mathbb{N}}$ and $Y \stackrel{\rm def}{=} \{Y_n\}_{n\in\mathbb{N}}$, are* indistinguishable by polynomial-time sampling *if for every probabilistic polynomial-time algorithm, D, every two positive polynomials $m(\cdot)$ and $p(\cdot)$, and all sufficiently large n's*

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

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

Definition 1.4 (efficiently constructible ensembles) *An ensemble, $X \stackrel{\rm def}{=} \{X_n\}_{n\in\mathbb{N}}$, is said to be* polynomial-time constructible *if there exists a probabilistic polynomial time algorithm S so that for every n, the random variables $S(1^n)$ and X_n are identically distributed.*

Theorem 1.5 *Let $X \stackrel{\rm def}{=} \{X_n\}_{n\in\mathbb{N}}$ and $Y \stackrel{\rm def}{=} \{Y_n\}_{n\in\mathbb{N}}$ be two polynomial-time constructible ensembles, and suppose that X and Y are computationally indistinguishable (as in Definition 1.1). Then X and Y are indistinguishable by polynomial-time sampling (as in Definition 1.3).*

An alternative formulation of Theorem 1.5 proceeds as follows. For every ensemble $Z \stackrel{\rm def}{=} \{Z_n\}_{n\in\mathbb{N}}$ and every polynomial $m(\cdot)$ define the $m(\cdot)$-product of Z as the ensemble $\{(Z_n^{(1)},...,Z_n^{(m(n))})\}_{n\in\mathbb{N}}$, where the $Z_n^{(i)}$'s are independent copies of Z_n. Theorem 1.5 asserts that *if the ensembles X and Y are computationally indistinguishable, and each is polynomial-time constructible, then, for every polynomial $m(\cdot)$, the $m(\cdot)$-product of X and the $m(\cdot)$-product of X are computationally indistinguishable.*

The information theoretic analogue of the above theorem is quite obvious: if two ensembles are statistically close then also their polynomial-products are statistically close[6]. The non-triviality of the computational version (given in Theorem 1.5) is evident from the presence of an extra condition; that is, that both ensembles are polynomial-time constructible. In contrast, we mention that there exists pairs of ensembles that are computationally indistinguishable but their 2-products are not computationally indistinguishable (since one of these ensembles is not polynomial-time constructible) [16].

Proof of Theorem 1.5: The proof is by a "reducibility argument". We show that the existence of an efficient algorithm that distinguishes the ensembles X and Y based on several samples, implies the existence of an efficient algorithm that distinguishes the ensembles X and Y based on a single sample. The implication is proven using the following argument, which is called a "hybrid argument".

Suppose, to the contradiction, that there is a probabilistic polynomial-time algorithm D, and polynomials $m(\cdot)$ and $p(\cdot)$, so that for infinitely many n's it holds that

$$\begin{aligned}\Delta(n) &\stackrel{\rm def}{=} \left|\Pr\left[D(X_n^{(1)},...,X_n^{(m)})=1\right] - \Pr\left[D(Y_n^{(1)},...,Y_n^{(m)})=1\right]\right| \\ &> \frac{1}{p(n)}\end{aligned} \tag{1.3}$$

[6] The proof of this claim is left as an exercise.

where $m \stackrel{\text{def}}{=} m(n)$, and the $X_n^{(i)}$'s and $Y_n^{(i)}$'s are as in Definition 1.3. In the sequel, we will derive a contradiction by presenting a probabilistic polynomial-time algorithm, D', that distinguishes the ensembles X and Y (in the sense of Definition 1.1).

For every k, $0 \leq k \leq m$, we define the *hybrid* random variable H_n^k as a (m-long) sequence consisting of k independent copies of X_n followed by $m-k$ independent copies of Y_n. Namely,

$$H_n^k \stackrel{\text{def}}{=} (X_n^{(1)}, ..., X_n^{(k)}, Y_n^{(k+1)}, ..., Y_n^{(m)})$$

where $X_n^{(1)}$ through $X_n^{(k)}$ and $Y_n^{(k+1)}$ through $Y_n^{(m)}$, are independent random variables with each $X_n^{(i)}$ identical to X_n and each $Y_n^{(i)}$ identical to Y_n. Clearly, $H_n^m = (X_n^{(1)}, ..., X_n^{(m)})$, whereas $H_n^0 = (Y_n^{(1)}, ..., Y_n^{(m)})$.

By our hypothesis, algorithm D can distinguish the extreme hybrids (i.e., H_n^0 and H_n^m). Since the total number of hybrids is polynomial in n, a non-negligible gap between (the "accepting" probability of D on) the extreme hybrids translates into a non-negligible gap between (the "accepting" probability of D on) a pair of neighboring hybrids. It follows that D, although not "designed to work on general hybrids", can distinguish a pair of neighboring hybrids. The punch-line is that, algorithm D can be easily modified into an algorithm D' that distinguishes X and Y. Details follow.

We construct an algorithm D' which uses algorithm D as a subroutine. On input α (supposedly in the range of either X_n or Y_n), algorithm D' proceeds as follows. Algorithm D', first selects k uniformly in the set $\{0, 1, ..., m-1\}$. Using the efficient sampling algorithm for the ensemble X, algorithm D' generates k independent samples of X_n. These samples are denoted $x^1, ..., x^k$. Likewise, using the efficient sampling algorithm for the ensemble Y, algorithm D' generates $m-k-1$ independent samples of Y_n, denoted $y^{k+2}, ..., y^m$. Finally, algorithm D' invokes algorithm D and halts with output $D(x^1, ..., x^k, \alpha, y^{k+2}, ..., y^m)$.

Clearly, D' can be implemented in probabilistic polynomial-time. The verification of the following two claims is left as an exercise (hint: use the first claim in proving the second).

Claim 1.5.1:

$$\Pr[D'(X_n)=1] = \frac{1}{m} \sum_{k=0}^{m-1} \Pr[D(H_n^{k+1})=1]$$

and

$$\Pr[D'(Y_n)=1] = \frac{1}{m} \sum_{k=0}^{m-1} \Pr[D(H_n^{k})=1]$$

Claim 1.5.2: For $\Delta(n)$ as in Eq. (1.3),

$$|\Pr[D'(X_n)=1] - \Pr[D'(Y_n)=1]| = \frac{\Delta(n)}{m(n)}$$

Since by our hypothesis $\Delta(n) > \frac{1}{p(n)}$, for infinitely many n's, it follows that the probabilistic polynomial-time algorithm D' distinguishes X and Y in contradiction to the hypothesis of the theorem. Hence, the theorem follows. ■

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

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

2. *Neighboring hybrids are easily related to the basic ensembles*: this property is essential since what we know (i.e., indistinguishability of the basic ensembles) relates to the basic ensembles. We need to be able to translate our knowledge (i.e., computational indistinguishability) of the basic ensembles to knowledge (i.e., computational indistinguishability) of any pair of neighboring hybrids. Typically, it is required to efficiently transform strings in the range of a basic distribution into strings in the range of a hybrid, so that the transformation maps the first basic distribution to one hybrid and the second basic distribution to the neighboring hybrid. (In the proof of Theorem 1.5, the hypothesis that both X and Y are polynomial-time constructible is instrumental for such an efficient transformation.)

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

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

Lecture 2

Pseudorandom Generators

2.1 Basic definition and initial discussion

Following the discussion in the previous lecture, we now present the definition of pseudorandom generators. Recall that a stretching function, $\ell:\mathbb{N}\to\mathbb{N}$, satisfies $\ell(n) > n$ for all n.

Definition 2.1 (Pseudorandom Generators [2, 35]) *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 2.2 (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 2.3 *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,*

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

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.

In particular, if A solves a decision problem then we may define $D(x,\sigma) \stackrel{\text{def}}{=} \sigma$; whereas if A solves an NP-search problem then we may define $D(x,y) \stackrel{\text{def}}{=} 1$ if y is a valid solution to instance x (and $D(x,y) \stackrel{\text{def}}{=} 0$ otherwise). Proposition 2.3 is proven by showing that any 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.[1] 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.*

2.2 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, generators 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 2.4 ([18]) *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*

$$\begin{aligned} G'(s) &\stackrel{\text{def}}{=} \sigma_1\sigma_2\cdots\sigma_{\ell'(|s|)}\,, \\ &\quad \textit{where } x_0 = s,\ \sigma_i = g_2(x_{i-1}) \textit{ and } x_i = g_1(x_{i-1}),\ \textit{for } i = 1,\ldots,\ell'(|s|) \end{aligned}$$

is a pseudorandom generator with stretch function ℓ'.

[1] Specifically, ignoring its own input, algorithm D' may first find an adequate A-input x by repeatedly invoking F, and testing each candidate x by approximating $\Delta_{A,D}(x)$ (using algorithms A, G and D). Next, using its own input denoted α, algorithm D' invokes D on input $(x, A(x,\alpha))$, thus distinguishing $U_{\rho(n)}$ from $G(U_{k(n)})$. Filling-up all details is left as an exercise for the reader.

Proof Sketch: The theorem is proven using the *hybrid technique* (see above): One considers distributions H_n^i (for $i = 0, ..., \ell'(n)$) defined by $U_i^{(1)} G'_{\ell'(n)-i}(U_n^{(2)})$, where $U_i^{(1)}$ and $U_n^{(2)}$ are independent uniform distributions (over $\{0,1\}^i$ and $\{0,1\}^n$, respectively), and $G'_j(x)$ 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}. The heart of the argument is the latter assertion, which is justified below.

We observe that $G'_j(s) = g_2(s) G'_{j-1}(g_1(s))$, for $j \geq 1$. Letting $p_1(x)$ (resp., $p_2(x)$) denotes the n-bit long prefix (resp., the last bit) of $x \in \{0,1\}^{n+1}$, we also have $g_1(s) = p_1(G(s))$ and $g_2(s) = p_2(G(s))$. Thus, for $i < \ell'(n)$, we can write

$$\begin{aligned}
H_n^i &\equiv (U_i^{(1)}, G'_{\ell'(n)-i}((U_n^{(2)}))) \\
&\equiv (U_i^{(1)}, g_2(U_n^{(2)}), G'_{(\ell'(n)-i)-1}(g_1(U_n^{(2)}))) \\
&\equiv (U_i^{(1)}, p_2(G(U_n^{(2)})), G'_{\ell'(n)-i-1}(p_1(G(U_n^{(2)})))) \\
H_n^{i+1} &\equiv (U_{i+1}^{(1)}, G'_{\ell'(n)-(i+1)}(U_n^{(2'')})) \\
&\equiv (U_i^{(1')}, U_1^{(1'')}, G'_{\ell'(n)-i-1}(U_n^{(2'')})) \\
&\equiv (U_i^{(1)}, p_2(U_{n+1}^{(2')}), G'_{\ell'(n)-i-1}(p_1(U_{n+1}^{(2')}))) \,,
\end{aligned}$$

where the various $U_j^{(k)}$'s are independent uniform distributions (over $\{0,1\}^j$). Suppose one could distinguish H_n^i from H_n^{i+1}. Incorporating the generation of $U_i^{(1)}$ and the evaluation of $G'_{\ell'(n)-i-1}$ into the distinguisher, one could distinguish $G(U_n)$ from U_{n+1}. Specifically, on input $\alpha \in \{0,1\}^{n+1}$, the new distinguisher parses α as $\beta\sigma$, where $|\beta| = n$ and $\sigma \in \{0,1\}$, uniformly selects $r \in \{0,1\}^i$, and invokes the given distinguisher on input $r\sigma G'_{\ell'(n)-i-1}(\beta)$. The reader can verify that on input $G(U_n)$ (resp., U_{n+1}) the new distinguisher invokes the given distinguisher on input H_n^i (resp., H_n^{i+1}). Thus, the new distinguisher distinguishes $G(U_n)$ from U_{n+1}, in contradiction to the pseudorandomness of G. ■

2.3 How to Construct Pseudorandom Generators

The mere fact that we have defined something does not mean that it exist. *Do pseudorandomness generators exist?* We do not know the answer to this question. But we do not know many other things; we even do not know whether $\mathcal{P}$ is *strictly* contained in $\mathcal{NP}$. In fact, if pseudorandomness generators exist then $\mathcal{P}$ is *strictly* contained in $\mathcal{NP}$. Thus, one should not expect, at this stage of history, to see an unconditional proof of the existence of pseudorandomness generators. Furthermore, the existence of pseudorandomness generators implies even stronger forms of computation difficulty (than the assertion that $\mathcal{NP}$ contains problems that are not solvable in probabilistic polynomial-time). Thus, the best we can hope for (now) are results that transform computation difficulty into pseudorandomness generators. This is indeed the type of results that are known and will be presented below.

The known constructions of pseudorandomness generators utilize computation difficulty, in the form of one-way functions (defined below). 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 2.5 (one-way function [6, 35]) *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*

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

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

We stress that the inverting algorithm is not required to retrieve the "original" preimage x; retrieving any preimage of $f(x)$ is considered a success. Still, if f is one-way then no probabilistic polynomial-time algorithm may succeed with non-negligible probability. Popular candidates for one-way functions are based on the conjectured intractability of integer factorization (cf. [28] for state of the art), the discrete logarithm problem (cf. [29] analogously), and decoding of random linear code [14].

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 $\Pr_{i,x}[A(i,f(x)) \neq b_i(x)] > 1/2n$, where the probability is taken uniformly over $i \in \{1,...,n\}$ and $x \in \{0,1\}^n$.[2] 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 2.6 (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's*

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

A predicate b may be a hard-core of f for the trivial reason that $f(U_n)$ contains no information on $b(U_n)$. Consider for example, the function $f(\sigma,x)=(1,f'(x))$, and the predicate $b(\sigma,x)=\sigma$, where $\sigma \in \{0,1\}$ and $x \in \{0,1\}^*$. In contrast, we are interested in the case where the hard-core property is due to computational reasons and not to information loss. In particular, a 1-1 function does not loss any information, and so if it has a hard-core predicate then this must be due to computational reasons. In fact, the reader may verify that if b is a hard-core of a 1-1 polynomial-time computable function f then f must be one-way. It turns out that any one-way function can be slightly modified so that it has a hard-core predicate.

Theorem 2.7 (A generic hard-core [15]) *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.*

A proof of Theorem 2.7 is given in the appendix to this lecture series. We are now ready to present constructions of pseudorandom generators.

[2] Otherwise, algorithm A can be used to recover all the bits of x from $f(x)$ (with success probability at least $1-n\cdot(1/2n)$). In contrast, some of these $b_i(x)$'s may be easy to compute from $f(x)$, and all $b_i(x)$'s may be easy to predict with probability 3/4. The proof of these claims is left as an exercise. (Hint: using any one-way function f', consider one-way functions such as $f(x,y)=(x,f'(y))$ and $f(x,y)=(x',f'(x''),y)$, where x' denotes some projection of x specified by y and x'' denotes the rest of the bits of y.)

2.3.1 The preferred presentation

In view of Theorem 2.4, we may focus on constructing pseudorandom generators with stretch function $\ell(n) = n + 1$. Such a construction is presented next.

Proposition 2.8 (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: 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 $f(s)$.

Formally, given any algorithm (denoted D) that distinguishes $\{G(U_n)\}$ and $\{U_{n+1}\}$, we construct a predictor (denoted A) of $b(U_n)$ based on $f(U_n)$. We assume, to the contradiction and without loss of generality, that for some polynomial p and infinitely many n's

$$\Pr[D(f(U_n)b(U_n)) = 1] - \Pr[D(U_{n+1}) = 1] \; > \; \frac{1}{p(n)} \tag{2.1}$$

Since f is 1-1 and onto $\{0,1\}^{|s|}$, it follows that U_{n+1} equals $f(U_n)b(U_n)$ with probability one half and equals $f(U_n)\overline{b}(U_n)$ otherwise, where $\overline{b}(x) \stackrel{\rm def}{=} 1 - b(x)$. Thus, Eq. (2.1) yields

$$\Pr[D(f(U_n)b(U_n)) = 1] - \Pr[D(f(U_n)\overline{b}(U_n)) = 1] \; > \; \frac{2}{p(n)} \tag{2.2}$$

Using D as a subroutine, we construct an algorithm A as follows. On input $y = f(x)$, algorithm A proceeds as follows:

(1) Select σ uniformly in $\{0,1\}$.
(2) If $D(y\sigma) = 1$ then output σ, otherwise output $1 - \sigma$.

Then, letting U_1 be independent of U_n (where U_1 represents the choice of σ in Step (1) of algorithm A), we have

$$\begin{aligned}
&\Pr[A(f(U_n)) = b(U_n)] \\
&\quad= \Pr[D(f(U_n)U_1) = 1 \;\&\; U_1 = b(U_n)] \\
&\qquad + \Pr[D(f(U_n)U_1) = 0 \;\&\; 1 - U_1 = b(U_n)] \\
&\quad= \Pr[D(f(U_n)b(U_n)) = 1 \;\&\; U_1 = b(U_n)] \\
&\qquad + \Pr[D(f(U_n)\overline{b}(U_n)) = 0 \;\&\; U_1 = \overline{b}(U_n)] \\
&\quad= \frac{1}{2} \cdot \Pr[D(f(U_n)b(U_n)) = 1] + \frac{1}{2} \cdot \left(1 - \Pr[D(f(U_n)\overline{b}(U_n)) = 1]\right) \\
&\quad= \frac{1}{2} + \frac{1}{2} \cdot \left(\Pr[D(f(U_n)b(U_n)) = 1] - \Pr[D(f(U_n)\overline{b}(U_n)) = 1]\right) \\
&\quad> \frac{1}{2} + \frac{1}{p(n)}
\end{aligned}$$

where the inequality is due to Eq. (2.2). But this contradicts the theorem's hypothesis by which b is a hard-core of f. ■

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.

2.3.2 An alternative presentation

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

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

where $x_0 = s$ and $x_i = f(x_{i-1})$ for $i = 1, ..., \ell(|s|) - 1$. 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) [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 $f(x_i)$ (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. [11, Sec. 3.3.5]).

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 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^{th} hybrid takes the first i bits from the questionable ensemble and the rest from the uniform one. It can be shown that distinguishing the extreme hybrids implies distinguishing some neighboring hybrids, which in turn implies next-bit predictability (of the questionable ensemble).

2.3.3 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 2.9 (On the existence of pseudorandom generators [20]) *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. (Turning the above argument into a rigorous proof is left as an exercise.)

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 the construction in Proposition 2.8 (i.e., $G(s) = f(s)b(s)$, where b is a hard-core of f) cannot be used *directly*. One idea of [20] is to hash $f(U_n)$ to an almost uniform string of length related to its entropy, using Universal Hash Functions [3]. (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 $f(U_n)$.)[3] 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 [20] 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.

[3] 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 $f(x_1, ..., x_{n^2}) \stackrel{\text{def}}{=} f'(x_1), ..., f'(x_{n^2})$.

Lecture 3

Pseudorandom Functions and Concluding Remarks

3.1 Definition and Construction of Pseudorandom Functions

Pseudorandom generators allow to explicitly generate (in an efficient manner) large pseudorandom objects using only a small amount of randomness. Pseudorandom functions (defined below) are even more powerful: They allow to implicitly generate (in an efficient manner) huge pseudorandom objects using the same small amount of randomness.

Consider a family of functions, each mapping poly(n)-bit long strings to poly(n)-bit long strings, and being specified by an n-bit long string called an index. We shall consider such families coupled with an efficient evaluation algorithm that, given the function's index and an argument, outputs the function's value at this argument. Such an algorithm coupled with an index, provides an implicit representation of a (relatively) huge object. We shall say that such a family is pseudorandom if functions uniformly selected in it are indistinguishable from truly 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 3.1 (pseudorandom functions [12]) *A* pseudorandom function (ensemble)*, with length parameters* $\ell_{\rm D}, \ell_{\rm R} : \mathbb{N} \to \mathbb{N}$*, is a collection of functions* $F \stackrel{\rm def}{=} \{f_s : \{0,1\}^{\ell_{\rm D}(|s|)} \to \{0,1\}^{\ell_{\rm R}(|s|)}\}_{s\in\{0,1\}^*}$ *satisfying*

- (efficient evaluation): *There exists an efficient* (deterministic) *algorithm that given a* seed*, s, and an $\ell_{\rm D}(|s|)$-bit* argument*, x, returns the $\ell_{\rm R}(|s|)$-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, 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 $\ell_{\mathrm{D}}(n) = n$ and $\ell_{\mathrm{R}}(n) = 1$. Then a function uniformly selected among 2^n functions (of a pseudorandom ensemble) presents an input-output behavior 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 3.2 (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*

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

Then, the function ensemble $\{f_s : \{0,1\}^{|s|} \to \{0,1\}^{|s|}\}_{s\in\{0,1\}^*}$, where $f_s(x) \stackrel{\mathrm{def}}{=} G_x(s)$, 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 $\ell_{\mathrm{D}}, \ell_{\mathrm{R}} : \mathbb{N} \to \mathbb{N}$.

Proof Sketch: The proof uses the hybrid technique: The i^{th} hybrid, H_n^i, is a function ensemble consisting of $2^{2^i \cdot n}$ functions $\{0,1\}^n \to \{0,1\}^n$, each defined by a sequence of 2^i (random) n-bit strings, denoted $\langle s_\alpha \rangle_{\alpha\in\{0,1\}^i}$. The value of such function at $x = \beta\alpha$, with $|\alpha| = i$, equals $G_\beta(s_\alpha)$. The extreme hybrids correspond to our indistinguishability claim (i.e., $H_n^0 \equiv f_{U_n}$ and $H_n^n \equiv R_n$), and neighboring hybrids correspond to our indistinguishability hypothesis (specifically, to the indistinguishability of $G(U_n)$ and U_{2n} under multiple samples). ■

Summary: Pseudorandom functions can replace truly random functions in any efficient application (e.g., most notably in cryptography). The advantage of such a replacement is that a pseudorandom function can be easily specified and shared among parties, whereas a truly random function is infeasible to specify and share. Specific examples where this issue is important are given next.

3.2 Applications of Pseudorandom Functions

3.2.1 Applications to Cryptography

Pseudorandom generators and functions are of key importance in Cryptography. Here we present two central applications of pseudorandom functions to private-key cryptography, showing how to establish private-key encryption and message authentication schemes.

Private-key encryption schemes. Loosely speaking, the goal of private-key encryption is to provide *private communication between mutually trustful parties that communicate over a public channel that may be eavesdropped by an adversary.* It is assumed that the parties have agreed on a secret (random) *key* prior to their interaction, and that the adversary does not know this key. Using this key and a corresponding *encryption algorithm*, one party may transform any *plaintext* that it wishes to transmit into a *ciphertext* that is being sent over the public channel, and the other party may retrieve the *plaintext* from the *ciphertext* by using the same key and a corresponding *decryption algorithm*. It should be infeasible for the adversary, seeing only the ciphertexts sent over the channel, to learn anything about the plaintexts (beyond what it knows a priori).[1]

Using pseudorandom functions, we implement private-key encryption as follows. The key, s, shared by the communicating parties is a random n-bit string specifying a pseudorandom function (as in Definition 3.1). The 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 a plaintext $m \in \{0,1\}^{\ell_{\mathrm{R}}(n)}$, the sender (using key s) uniformly selects $r \in \{0,1\}^{\ell_{\mathrm{D}}(n)}$, and sends $(r, m \oplus f_s(r))$ to the receiver. The plaintext is recovered in the straightforward manner; that is, by using key s and the ciphertext (r, y), the receiver recovers the plaintext $y \oplus f_s(r)$. 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. We comment that this encryption scheme withstands an attack in which, prior to being presented the challenge ciphertext, the adversary may ask the parties to encrypt (resp., decrypt) any plaintext (resp., ciphertext) of its choice relative to the same key.[2]

Message authentication schemes. Loosely speaking, the goal of message authentication is to provide *reliable communication between mutually trustful parties that communicate over a public channel that may be tampered by an adversary*. Again, the parties share a secret (random) key agreed upon prior to their interaction, and the adversary does not know this key. Using this key and a corresponding *tagging algorithm*, one party may compute an authentication tag for the message that it wishes to transmit and send this tag along with the message so that the other party may verify that the message was indeed sent by the legitimate sender (by using the same key and a corresponding *verification algorithm*). It should be infeasible for the adversary, seeing only the ciphertexts sent over the channel, to produce a new message and a tag that will be accepted by the legitimate parties. This should hold even in case the adversary can make the party tag any message of its choice; even in such a case it should be infeasible for the adversary to form an accepted authentication tag for any other message.

Using pseudorandom functions, we implement a message authentication scheme as follows. Again the key, s, shared by the communicating parties is a random n-bit string specifying a pseudorandom function. Authentication tags are produced and verified by applying the function f_s to the message. That is, to authenticate a message $m \in \{0,1\}^{\ell_{\mathrm{R}}(n)}$, the sender computes and sends along the tag $f_s(m)$. Verification of the pair (m, t), relative to the key s, is done by checking whether $t = f_s(m)$.

3.2.2 Other Applications

We mention that pseudorandom functions have been used to derive negative results in computational learning theory (cf. [34]). In particular, any concept class that contains a family of pseudorandom functions cannot be efficiently learnable even under the uniform distribution and with the help of membership queries.[3] A different set of negative results refers to Natural Proofs [30].

[1] The precise definition of the privacy condition is based on the "simulation paradigm" (cf. [19, 11]). This is done in a way analogous to the formulation of the zero-knowledge condition (cf. lecture notes on zero-knowledge proofs in this book).

[2] Resilience to an even stronger attack in which the adversary's queries may depend on its challenge can be obtained by using a message authentication scheme.

[3] A concept class is a set of boolean functions. A learning algorithm is given oracle access to any function in the class and is required to output a description of a function that is close to the target function (being queried), where distance between functions is defined as the fraction of inputs on which they agree. Clearly, no efficient algorithm given oracle access to a pseudorandom function can output a function that agrees with the target function

These are circuit lower bound proofs that proceed by identifying a "natural" superset of the set of functions computed by low complexity circuits.[4] Such circuit lower bounds (i.e., Natural Proofs) cannot exist for complexity classes containing a family of pseudorandom functions (see [30]).

3.3 Concluding Remarks

We start with a high-level discussion of the applicability of pseudorandom generators, and turn to a "philosophical" discussion of the nature of pseudorandom generators. We conclude this lecture series with comments regarding generalizations of the notion of pseudorandom generators (i.e., other instantiations of the main paradigm underlying this lecture series).

3.3.1 The applicability of pseudorandom generators

As discussed above, pseudorandom generators and functions are of key importance to Cryptography. Here we wish to discuss the applicability of pseudorandom generators to algorithmic design at large.

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. [21]). 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 2.1 aims at such generality, and in fact is tailored to obtain it: The notion of computational indistinguishability, which underlines Definition 2.1, covers all possible efficient applications postulating that for all of them pseudorandom sequences are as good as truly random ones.

3.3.2 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

on significantly more than half of the inputs.

[4] The *natural* superset should have small measure (within the set of all functions), and should have a low complexity decision procedure. That is, given the truth-table of a function, the procedure should determine whether or not the function belongs to the natural set. Note that the complexity of the decision procedure is viewed in terms of the length of the truth-table of the function, which is exponential in the length of the input to the function. Currently known circuit lower bounds tend to be natural with respect to very low complexity (of this decision procedure) [30].

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 [35, 13]. 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.

3.3.3 A General Paradigm

Pseudorandomness as surveyed in this lecture series 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 [10, Chap. 3].

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, for example, if we know that the algorithm uses very little work-space, 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; that is, 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). Above, 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. The benefit of this relaxation is that constructing exponential-time generators may be easier than constructing polynomial-time ones. 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. For further details, see the next lecture series.

[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.

Appendix

Proof of the existence of a generic hard-core

Theorem 2.7, conjectured by Levin [24] and proven by Goldreich and Levin [15], relates two computational tasks: The first task is inverting a function f; namely given y find an x so that $f(x) = y$. The second task is predicting, with non-negligible advantage, the exclusive-or of a subset of the bits of x when only given $f(x)$. More precisely, it has been proved that if f cannot be efficiently inverted then given $f(x)$ and r it is infeasible to predict the inner-product mod 2 of x and r (with success probability better than the obvious).

The proof presented here is not the original one presented in [15] (see generalization in [17]), but rather an alternative suggested by Charlie Rackoff. The alternative proof, inspired by [1], has two main advantages over the original one: It is simpler to explain, and it leads to better security (i.e., a more efficient reduction of inverting f to predicting the inner-product) [25].

Theorem 3.3 (Theorem 2.7 – restated): *Let $b(x,r)$ denote the inner-product mod 2 of the binary vectors x and r. Suppose we have oracle access to a random process $b_x : \{0,1\}^n \to \{0,1\}$, so that*

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

where the probability is taken uniformly over the internal coin tosses of b_x and all possible choices of $r \in \{0,1\}^n$. Then there exists an algorithm that, in time polynomial in n/ϵ and with probability at least $\mathrm{poly}(\epsilon/n)$*, outputs x.*

Theorem 2.7 is derived from the above by using standard arguments. We prove this fact first.

Proposition 3.4 *Theorem 3.3 implies Theorem 2.7.*

Proof: We assume for contradiction the existence of an efficient algorithm predicting the inner-product with advantage which is not negligible, and derive an algorithm that inverts f with related (i.e., non-negligible) success probability. This contradicts the hypothesis that f is a one-way function. Thus, the proof uses a "reducibility argument" – that is, we reduce the task of inverting f to the task of predicting $b(x,r)$ from $(f(x), r)$.

Let G be a (probabilistic polynomial-time) algorithm that on input $f(x)$ and r tries to predict the inner-product (mod 2) of x and r. Denote by $\epsilon_G(n)$ the (overall) advantage of algorithm G in predicting $b(x,r)$ from $f(x)$ and r, where x and r are uniformly chosen in $\{0,1\}^n$. Namely,

$$\epsilon_G(n) \stackrel{\mathrm{def}}{=} \Pr\left[G(f(X_n), R_n) = b(X_n, R_n)\right] - \frac{1}{2}$$

where here and in the sequel X_n and R_n denote two independent random variables, each uniformly distributed over $\{0,1\}^n$. In the sequel we shorthand ϵ_G by ϵ.

Our first observation is that, on at least an $\frac{\epsilon(n)}{2}$ fraction of the x's of length n, algorithm G has an $\frac{\epsilon(n)}{2}$ advantage in predicting $b(x, R_n)$ from $f(x)$ and R_n. Namely,

Claim: There exists a set $S_n \subseteq \{0,1\}^n$ of cardinality at least $\frac{\epsilon(n)}{2} \cdot 2^n$ such that for every $x \in S_n$, it holds that

$$s(x) \stackrel{\text{def}}{=} \Pr[G(f(x), R_n) = b(x, R_n)] \geq \frac{1}{2} + \frac{\epsilon(n)}{2}$$

Here the probability is taken over all possible values of R_n and all internal coin tosses of algorithm G, whereas x is fixed.

Proof: The observation follows by an averaging argument. Namely, write $\text{Exp}(s(X_n)) = \frac{1}{2} + \epsilon(n)$, and apply Markov Inequality. $\square$

Thus, we restrict our attention to x's in S_n. For each such x, the conditions of Theorem 3.3 hold, and so within time $\text{poly}(n/\epsilon(n))$ and with probability at least $1/2$ we retrieve a list of strings containing x. Contradiction to the one-wayness of f follows, since the probability we invert f on uniformly selected x is at least $\frac{1}{2} \cdot \Pr[x \in S_n] \geq \frac{\epsilon(n)}{4}$. $\blacksquare$

A motivating discussion

Let $s(x) \stackrel{\text{def}}{=} \Pr[b_x(r) = b(x, r)]$, where r is uniformly distributed in $\{0,1\}^{|x|}$. Then, by the hypothesis of Theorem 3.3, $s(x) \geq \frac{1}{2} + \epsilon$. Suppose, for a moment, that $s(x) > \frac{3}{4} + \epsilon$. In this case, retrieving x by querying the oracle b_x is quite easy. To retrieve the i^{th} bit of x, denoted x_i, we uniformly select $r \in \{0,1\}^n$, and obtain $b_x(r)$ and $b_x(r \oplus e^i)$, where e^i is an n-dimensional binary vector with 1 in the i^{th} component and 0 in all the others, and $v \oplus u$ denotes the addition mod 2 of the binary vectors v and u. Clearly, if both $b_x(r) = b(x, r)$ and $b_x(r \oplus e^i) = b(x, r \oplus e^i)$ then

$$\begin{aligned} b_x(r) \oplus b_x(r \oplus e^i) &= b(x, r) \oplus b(x, r \oplus e^i) \\ &= b(x, e^i) \\ &= x_i \end{aligned}$$

The probability that both equalities hold (i.e., both $b_x(r) = b(x, r)$ and $b_x(r \oplus e^i) = b(x, r \oplus e^i)$) is at least $1 - 2 \cdot (\frac{1}{4} - \epsilon) = \frac{1}{2} + 2\epsilon$. Hence, repeating the above procedure sufficiently many times and ruling by majority we retrieve x_i with very high probability. Similarly, we can retrieve all the bits of x, and hence obtain x itself. However, the entire analysis was conducted under (the unjustifiable) assumption that $s(x) > \frac{3}{4} + \epsilon$, whereas we only know that $s(x) > \frac{1}{2} + \epsilon$.

The problem with the above procedure is that it doubles the original error probability of the oracle b_x (on random queries). Under the unrealistic assumption, that the b_x's error on such inputs is significantly smaller than $\frac{1}{4}$, the "error-doubling" phenomenon raises no problems. However, in general (and even in the special case where b_x's error is exactly $\frac{1}{4}$) the above procedure is unlikely to yield x. Note that the error probability of b_x can *not* be decreased by querying b_x several times on the same instance (e.g., b_x may always answer correctly on three quarters of the inputs, and always err on the remaining quarter). What is required is an *alternative way of using* b_x – a way that does not double the original error probability of b_x. The key idea is to generate the r's in a way that requires querying b_x only once (e.g., on $(x, r \oplus e^i)$) per each r (and i), instead of twice. The good news are that the error probability is no longer doubled, since we will only use b_x to get an "estimate" of $b(x, r \oplus e^i)$. The bad news are that we still need to know $b(x, r)$, and it is not clear how we can know $b(x, r)$ without querying b_x. The answer is that we can guess $b(x, r)$ by ourselves. This is fine if we only need to guess $b(x, r)$ for one r (or logarithmically in

$|x|$ many r's), but the problem is that we need to know (and hence guess) $b(x,r)$ for polynomially many r's. An obvious way of guessing these $b(x,r)$'s yields an exponentially vanishing success probability. The solution is to generate these polynomially many r's so that, on one hand they are "sufficiently random" whereas on the other hand we can guess all the $b(x,r)$'s with non-negligible success probability. Specifically, generating the r's in a *particular* pairwise independent manner will satisfy both (seemingly contradictory) requirements. We stress that in case we are successful (in our guesses for the $b(x,r)$'s), we can retrieve x with high probability. Hence, we retrieve x with non-negligible probability.

A word about the way in which the pairwise independent r's are generated (and the corresponding $b(x,r)$'s are guessed) is indeed in place. To generate $m = \text{poly}(n/\epsilon)$ many r's, we uniformly (and independently) select $l \stackrel{\text{def}}{=} \log_2(m+1)$ strings in $\{0,1\}^n$. Let us denote these strings by $s^1,...,s^l$. We then guess $b(x,s^1)$ through $b(x,s^l)$. Let us denote these guesses, which are uniformly (and independently) chosen in $\{0,1\}$, by σ^1 through σ^l. Hence, the probability that all our guesses for the $b(x,s^i)$'s are correct is $2^{-l} = \frac{1}{\text{poly}(n/\epsilon)}$. The different r's correspond to the different non-empty subsets of $\{1,2,...,l\}$. We compute $r^J \stackrel{\text{def}}{=} \bigoplus_{j\in J} s^j$. The reader can easily verify that the r^J's are pairwise independent and each is uniformly distributed in $\{0,1\}^n$. The key observation is that

$$b(x,r^J) = b(x,\bigoplus_{j\in J} s^j) = \bigoplus_{j\in J} b(x,s^j)$$

Hence, our guesses for the $b(x,r^J)$'s are the corresponding $\bigoplus_{j\in J}\sigma^j$'s, and with non-negligible probability all our guesses are correct.

Back to the formal argument

Following is a formal description of the recovering algorithm, denoted A. On input n and ϵ (and oracle access to b_x), algorithm A sets $l \stackrel{\text{def}}{=} \lceil \log_2(n\cdot\epsilon^{-2}+1)\rceil$. Algorithm A uniformly and independently select $s^1,...,s^l \in \{0,1\}^n$, and $\sigma^1,...,\sigma^l \in \{0,1\}$. It then computes, for every non-empty set $J \subseteq \{1,2,...,l\}$, a string $r^J \leftarrow \bigoplus_{j\in J} s^j$ and a bit $\rho^J \leftarrow \bigoplus_{j\in J}\sigma^j$. For every $i \in \{1,...,n\}$ and every *non-empty* $J \subseteq \{1,..,l\}$, algorithm A computes $z_i^J \leftarrow \rho^J \oplus b_x(r^J \oplus e^i)$. Finally, algorithm A sets z_i to be the majority of the z_i^J values, and outputs $z = z_1\cdots z_n$.[6]

Clearly, A makes $n\cdot 2^l = n^2/\epsilon^2$ oracle calls to b_x, and the same amount of other elementary computations. Following is a detailed analysis of the success probability of algorithm A. We start by showing that, in case all the σ^j's are correct (i.e., equal the corresponding $b(x,s^j)$'s), with constant probability, $z_i = x_i$ for all $i \in \{1,...,n\}$. This is proven by bounding from below the probability that the majority of the z_i^J's equals x_i.

Claim: For every $1 \le i \le n$,

$$\Pr\left[|\{J : b(x,r^J)\oplus b_x(r^J\oplus e^i) = x_i\}| \;>\; \frac{1}{2}\cdot(2^l-1)\right] \;>\; 1-\frac{1}{4n}$$

where $r^J \stackrel{\text{def}}{=} \bigoplus_{j\in J} s^j$ and the s^j's are independently and uniformly chosen in $\{0,1\}^n$.

Proof: For every J, define a 0-1 random variable ζ^J, so that ζ^J equals 1 if and only if $b(x,r^J)\oplus b_x(r^J\oplus e^i) = x_i$. The reader can easily verify that each r^J is uniformly distributed in $\{0,1\}^n$. It follows that

[6] An alternative implementation of the above ideas results in an alternative algorithm, denoted A'. Rather than selecting at random a setting of $\sigma^1,...,\sigma^l \in \{0,1\}$, algorithm A' tries all possible values for $\sigma^1,...,\sigma^l$. It outputs a list of 2^l candidates z's, one per each of the possible settings of $\sigma^1,...,\sigma^l \in \{0,1\}$. It can be shown that, with probability at least 3/4, the list output by A' contains x.

each ζ^J equals 1 with probability $\frac{1}{2}+\epsilon$. We show that the ζ^J's are pairwise independent by showing that the r^J's are pairwise independent. For every $J \neq K$ we have, without loss of generality, $j \in J$ and $k \in K \setminus J$. Hence, for every $\alpha, \beta \in \{0,1\}^n$, we have

$$\begin{aligned} \Pr\left[r^K = \beta \,|\, r^J = \alpha\right] &= \Pr\left[s^k = \beta \,|\, s^j = \alpha\right] \\ &= \Pr\left[s^k = \beta\right] \\ &= \Pr\left[r^K = \beta\right] \end{aligned}$$

and pairwise independence of the r^J's follows. Let $m \stackrel{\text{def}}{=} 2^l - 1$. Using Chebyshev's Inequality, we get

$$\begin{aligned} \Pr\left[\sum_J \zeta^J \leq \frac{1}{2} \cdot m\right] &\leq \Pr\left[\left|\sum_J \zeta^J - (0.5+\epsilon) \cdot m\right| \geq \epsilon \cdot m\right] \\ &< \frac{\max_J\{\mathrm{Var}(\zeta^J)\}}{\epsilon^{-2} \cdot (n/\epsilon^2)} \\ &< \frac{1}{4n} \end{aligned}$$

The claim follows. $\square$

Recall that if $\sigma^j = b(x, s^j)$, for all j's, then $\rho^J = b(x, r^J)$ for all non-empty J's. In this case z output by algorithm A equals x, with probability at least $3/4$. However, the first event happens with probability $2^{-l} = \frac{1}{n/\epsilon^2}$ independently of the events analyzed in the Claim. Hence, algorithm A recovers x with probability at least $\frac{3}{4} \cdot \frac{\epsilon^2}{n}$. Theorem 3.3 follows. $\blacksquare$

Bibliography

[1] W. Alexi, B. Chor, O. Goldreich and C.P. Schnorr. RSA/Rabin Functions: Certain Parts are As Hard As the Whole. *SIAM Journal on Computing*, Vol. 17, April 1988, pages 194–209.

[2] 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.

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

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

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

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

[7] W. Feller. *An Introduction to Probability Theory and Its Applications.* John Wiley, New York, 1968.

[8] M.R. Garey and D.S. Johnson: *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.

[9] O. Goldreich. A Note on Computational Indistinguishability. *Information Processing Letters*, Vol. 34, pages 277–281, May 1990.

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

[11] O. Goldreich. *Foundations of Cryptography – Basic Tools.* To be published in 2000/2001 by Cambridge University Press. Preliminary versions and further information available from `http://www.wisdom.weizmann.ac.il/~oded/foc-book.html`.

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

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

[15] 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.

[16] O. Goldreich and B. Meyer. Computational Indistinguishability – Algorithms vs. Circuits. *Theoretical Computer Science*, Vol. 191, pages 215–218, 1998.

[17] O. Goldreich, R. Rubinfeld and M. Sudan. Learning polynomials with queries: the highly noisy case. *SIAM J. on Disc. Math.*, March 2000.

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

[19] 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.

[20] J. Hastad, 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 Hastad in *22nd ACM Symposium on the Theory of Computing* (1990).

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

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

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

[24] L.A. Levin. One-Way Function and Pseudorandom Generators. *Combinatorica*, Vol. 7, pages 357–363, 1987.

[25] L.A. Levin. Randomness and Non-determinism. *J. Symb. Logic*, Vol. 58(3), pages 1102–1103, 1993.

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

[27] R. Motwani and P. Raghavan. *Randomized Algorithms*, Cambridge University Press, 1995.

[28] 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`

[29] 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`

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

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

[32] M. Sipser. *Introduction to the Theory of Computation.* PWS Publishing Company, 1997.

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

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

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