Introduction to Complexity Theory*
Lecture 13: Pseudorandom Generators

Notes taken by Sergey Benditkis, Boris Temkin and Il'ya Safro

March 26, 1999

Summary: Pseudorandom generators are defined as efficient deterministic algorithms which stretch short random seeds into longer pseudorandom sequences. The latter are indistiguishable from truely random sequences by any efficient observer. We show that, for efficiently sampleable distributions, computational indistiguishability is preserved under multiple samples. We related pseudorandom generators and one-way functions, and show how to increase the stretching of pseudorandom generators.

1 Instead of an introduction

Oded's Note: *See introduction and motivation in the Appendix. Actually, it is recommended to read the appendix before reading the following notes, and to refer to the notes only for full details of some statements made in Sections 6.2 and 6.3 of the appendix.*

Oded's Note: *Loosely speaking, pseudorandom generators are defined as efficient deterministic algorithms which stretch short random seeds into longer pseudorandom sequences. We stress three aspects: (1) the efficiency of the generator; (2) the stretching of seeds to longer strings; (3) the pseudorandomness of output sequences. The third aspect refers to the key notion of computational indistinguishability. We start with a definition and discussion of the latter.*

2 Computational Indistinguishability

We have two things which are called "probability ensembles", and denoted by $\{X_n\}_{n\in\mathsf{N}}$ and $\{Y_n\}_{n\in\mathsf{N}}$. We are talking about infinite sequences of distributions like we talk about the language and each distribution seats on some final domain. Typicaly the distribution X_n will have as a support strings of length polynomial of n, not more and not much less.

Definition 1 (probability ensembles): *A* probability ensemble *X is a family $X = \{X_n\}_{n\geq 1}$ such that X_n is a probability distribution on some finite domain.*

What is to say that these ensembles are computational indistinguishable? We want to look at the particular algorithm A and want to ask what is the probability of the event: when you give to A

* Lecture Notes for a course given by Oded Goldreich at the Weizmann Institute of Science, Israel.

an input X_n then it says 1 (or 1 is just arbitrary) and look at the difference of the probabilities for answers of execution this algorithm A for two inputs $\{X_n\}_{n\in\mathsf{N}}$ and $\{Y_n\}_{n\in\mathsf{N}}$. And if this difference is negligible, when you look at n as the parameter then we will say that we can not distinguish the first ensamble from the second one.

Definition 2 (negligible functions): *The function $f : \mathsf{N} \mapsto [0,1]$ is* negligible *if for all polynomials p, and for all sufficiently large n's, $f(n) < 1/p(n)$.*

Suppose we have two *probability ensambles* $\{X_n\}_{n\in\mathsf{N}}$ and $\{Y_n\}_{n\in\mathsf{N}}$, where X_n and Y_n are distributions over some finite domain.

Definition 3 (indistinguishability by a specific algorithm): *Consider some probabilistic algorithm A. We will say that $\{X_n\}$ and $\{Y_n\}$ are* indistinguishable by A *if*

$$|\Pr(A(X_n) = 1) - \Pr(A(Y_n) = 1)| < \frac{1}{p(n)}$$

for every polinomial $p()$ and for every sufficiently large n.

2.1 Two variants

Our main focus will be on indistinguishability by any probabilistic polynomial-time algorithm. That is,

Definition 4 (canonic notion of computational indistinguishability): *Two probability ansambles $\{X_n\}_{n\in\mathsf{N}}$ and $\{Y_n\}_{n\in\mathsf{N}}$ are* computationally indistinguishable *if they are indistinguishable by any probabilistic polynomial-time algorithm. That is, for every probabilistic polynomial-time algorithm A, and every polinomial $p()$ there exists N s.t. for all $n > N$*

$$|\Pr(A(X_n) = 1) - \Pr(A(Y_n) = 1)| < \frac{1}{p(n)}$$

Another notion that we talk about is indistinguhability by circuits.

Definition 5 *Two probability ensembles $\{X_n\}_{n\in\mathsf{N}}$ and $\{Y_n\}_{n\in\mathsf{N}}$ are* indistinguishable by small circuits *if for all families of polynomial-size circuits $\{C_n\}$*

$$|\,[\Pr(C_n(X_n) = 1) - \Pr(C_n(Y_n) = 1)]\,|$$

is neglibible.

2.2 Relation to Statistical Closeness

Oded's Note: *This subsection was rewritten by me.*

The notion of Computational Indistinguishability is a relaxation of the notion of *statistical closeness* (or *statistical indistinguishability*).

Definition 6 (statistical closeness): *The* statistical difference (or variation distance) *between two distributions, X and Y, is defined by*

$$\Delta(X,Y) \stackrel{\text{def}}{=} \frac{1}{2} \cdot \sum_{\alpha} |\Pr[X=\alpha] - \Pr[Y=\alpha]|$$

Two probability ensembles $\{X_n\}_{n\in\mathsf{N}}$ and $\{Y_n\}_{n\in\mathsf{N}}$ are statistical close *if $\Delta(X_n,Y_n)$ is a negligible function of n. That is, for all polynomial $p()$ there exists N s.t. for all $n > N$ $\Delta(X_n,Y_n) < 1/p(n)$.*

An equivalent definition of $\Delta(X_n,Y_n)$, is the maximum over all subsets, S, of $\Pr[X_n \in S] - \Pr[Y_n \in S]$. (A set S which obtains the maximum is the set of all z's satisfying $\Pr[X_n = z] > \Pr[Y_n = z]$, which proves the equivalence.) Yet another equivalent definition of $\Delta(X_n,Y_n)$ is the maximum over all Boolean f's of $\Pr[f(X_n)=1] - \Pr[f(Y_n)=1]$. Thus,

Proposition 2.1 *If two probability ensembles are statistical close then they are computationally indistinguishable.*

We note that there are computationally indistinguishable probability ensembles which are not statistical close.

2.3 Computational indistinguishability and multiple samples

Oded's Note: *We show that under certain conditions, computational indistinguishability is preserved under multiple samples.*

Definition 7 (constructability of ensembles): *The ensemble $\{Z_n\}_{n\in\mathsf{N}}$ is* probabilistic polynomial-time constructable *if there exists a probabilistic polynomial time algorithm S such that for every n, $S(1_n) \equiv Z_n$.*

Theorem 8 *Let $\{X_n\}$ and $\{Y_n\}$ computationally indistinguishable (i.e., indistinguishable by any probabilistic polynomial time algorithm). Suppose they are both probabilistic polynomial time constructable. Let $t()$ be a positive polinomial. Define $\{\overline{X_n}\}_{n\in N}$ and $\{\overline{Y_n}\}_{n\in N}$ in the following way:*

$$\overline{X_n} = X_n^1 \circ X_n^2 \circ \ \dots\ \circ X_n^{t(n)}, \quad \overline{Y_n} = Y_n^1 \circ Y_n^2 \circ \ \dots\ \circ Y_n^{t(n)}$$

The X_n^i's (resp. Y_n^i's) are independent *copies of X_n (Y_n). Then $\{\overline{X_n}\}$ and $\{\overline{Y_n}\}$ are Probabilistic Polynomial Time indistinguishable.*

Proof: Suppose, there exists a distinguisher D, between $\{\overline{X_n}\}$ and $\{\overline{Y_n}\}$.

Oded's Note: *We use the "hybrid technique": We define hybrid distributions so that the extreme hybrids coincide with $\{\overline{X_n}\}$ and $\{\overline{Y_n}\}$, and link distinguishability of neighboring hybrids to distinguishability of $\{X_n\}$ and $\{Y_n\}$.*

Then define

$$H_n^{(i)} = \left(X_n^{(1)} \circ X_n^{(2)} \circ \ \dots\ X_n^{(i)} \circ Y_n^{(i+1)} \circ \ \dots\ Y_n^{(t(n))}\right)$$

It is easy to see that $H_n^{(0)} = \overline{Y_n}$, $H_n^{(t(n))} = \overline{X_n}$.

Oded's Note: *Also note that $H_n^{(i)}$ and $H_n^{(i+1)}$ differ only in the distribution of the $i+1^{\text{st}}$ component, which is identical to Y_n in the first hybrid and to X_n in the second. The idea is to distinguish Y_n and X_n by pluging them in the $i+1^{\text{st}}$ component of a distribution. The new distribution will be distributed identically to either $H_n^{(i)}$ or $H_n^{(i+1)}$, respectively.*

Define algorithm D' as follows:

Begin Algorithm Distinguisher
Input α , (taken from X_n or Y_n)
(1) Choose $i \in_R \{1 .. t(n)\}$ (i.e., uniformly in $\{1, ..., t(n)\}$)
(2) Construct $Z = \left(X_n^{(1)} \circ X_n^{(2)} \circ \ ... \ X_n^{(i-1)} \circ \alpha \circ Y_n^{(i+1)} \circ \ ... \ Y_n^{(t(n))}\right)$
Return $D(Z)$
end.

$$\begin{aligned}
\Pr\left[D'(X_n) = 1\right] &= \frac{1}{t(n)} \sum_{i=1}^{t(n)} \Pr\left[D(X_n^{(1)} \circ X_n^{(2)} \circ \ ... \ X_n^{(i-1)} \circ X_n \circ Y_n^{(i+1)} \circ \ ... \ Y_n^{(t(n))}) = 1\right] \\
&= \frac{1}{t(n)} \sum_{i=1}^{t(n)} \Pr\left[D(H_n^{(i)}) = 1\right]
\end{aligned}$$

whereas

$$\begin{aligned}
\Pr\left[D'(Y_n) = 1\right] &= \frac{1}{t(n)} \sum_{i=1}^{t(n)} \Pr\left[D(X_n^{(1)} \circ X_n^{(2)} \circ \ ... \ X_n^{(i-1)} \circ Y_n \circ Y_n^{(i+1)} \circ \ ... \ Y_n^{(t(n))}) = 1\right] \\
&= \frac{1}{t(n)} \sum_{i=1}^{t(n)} \Pr\left[D(H_n^{(i-1)}) = 1\right].
\end{aligned}$$

Thus,

$$\begin{aligned}
&\left|\Pr\left[D'(X_n) = 1\right] - \Pr\left[D'(Y_n) = 1\right]\right| \\
&= \frac{1}{t(n)} \cdot \left| \sum_{i=1}^{t(n)} \Pr[D(H_n^{(i)}) = 1] - \sum_{i=1}^{t(n)} \Pr[D(H_n^{(i-1)}) = 1] \right| \\
&= \frac{1}{t(n)} \cdot \left| \Pr[D(H_n^{(t(n))}) = 1] - \Pr[D(H_n^{(0)}) = 1] \right| \\
&= \frac{1}{t(n)} \cdot \left| \Pr[D(X_n) = 1] - \Pr[D(Y_n) = 1] \right| \geq \frac{1}{t(n) \cdot p(n)}
\end{aligned}$$

for some $p()$ and for infinitely many n's ■

> Oded's Note: *One can easily show that computational indistinguishability by small circuits is preserved under multiple samples. Here we don't need to assume probabilistic polynomial-time constructability of the ensembles.*

3 PRG: Definition and amplification of the stretch function

Intuitively, a pseudo-random generator takes a short, truly-random string and stretches it into a long, pseudorandom one. The pseudorandom string should look "random enough" to use it in place of a truly random string.

Definition 9 (PseudoRandom Generator – PRG): *The function $G : \{0,1\}^* \rightarrow \{0,1\}^*$ with stretch function $l(n)$ is a* pseudo-random generator *if:*

- *G is a polynomial time algorithm*
- *for every x, $|G(x)| = l(|x|) > |x|$*
- *$\{G(U_n)\}$ and $\{U_{l(n)}\}$ are computational indistinguishable, where U_m denotes the uniform distribution over $\{0,1\}^m$.*

  Oded's Note: *The above definition is minimalistic regarding its stretch requirement. A generator stretching n bits into $n+1$ bits seems to be of little use. However, as shown next, such minimal stretch generators can be used to construct generators of arbitrary stretch.*

Theorem 10 (amplification of stretch function): *Suppose we have a Pseudo-Random Generator G_1 with a stretch function $n + 1$. Then for all polynome $l(n)$ there exists a Pseudo-Random Generator with stretch function $l(n)$.*

Proof:

Construct G as follows: We take the input seed x ($|x| = n$) and feed it through G_1. Then we save the first bit of the output of G_1 (denote it by y_1), and feed the rest of the bits as input to a new invocation of G_1. We repeat this operation $l(n)$ times, in the i-th step we invoke G_1 on input determined in the previous step, save the first output bit as y_i and use the rest n bits as an input to step $i+1$. The output of G is $y = y_1 y_2 \ldots y_{l(n)}$. See Figure 1.

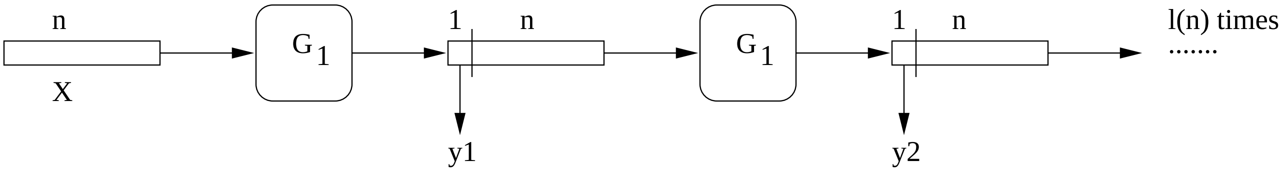

Figure 1.

We claim that G is a Pseudo-Random Generator. The first two requirements for Pseudo-Random Generator are trivial (by construction/definition of G). We will prove the 3rd one. The proof is by contradiction, again using the hybrid method.

Suppose there exists a distinguisher $A : \{0,1\}^* \rightarrow \{0,1\}^{l(n)}$ such that exists polynomial $p()$ and for infinitely namy n's

$$| \Pr[\, A(G(U_n)) = 1 \,] - \Pr\left[\, A(U_{l(n)}) = 1 \,\right] | \geq \frac{1}{p(n)}$$

Let us make the following construction . Define sequence of functions $g^{(i)}$:

$g^{(0)}$ is empty

$g^{(i)} = [G_1(x)]_1 \circ g^{(i-1)}([G_1(x)]_{2\ldots(n+1)})$

Where $[y]_i$ is the notation of i-th bit of y and $[y]_{2\ldots(n+1)}$ denotes substring of y from the second bit up to $n+1$-th bit. It is easy to see that $g^{l(n)} = G(x)$.

Construct the class of hybrid distributions $\{H^i\}_{i=1}^{l(n)}$:

$$H^i = U_{l(n)-i} \circ g^i(U_n)$$

One can obeserve that $H^0 = G(U_n)$, and $H^{l(n)} = U_{l(n)}$.
Now we construct the distinguisher D as follows:

Begin Algorithm Distinguisher
Input α , $|\alpha| = n+1$ (taken from $G_1(U_n)$ on U_{n+1})
(1) Choose $i \in_R \{1 \,..\, l(n)\}$
(2) Choose $Z \sim U_{l(n)-i}$
(3) Construct $y = Z \circ \sigma \circ g^{(i-1)}(S)$, where σ is first bit of α and S its n-bit suffix
Return $A(y)$
end.

We denote by $\Pr[A|i]$ the *conitional probability* of event A if particular i was choosen in step (1) of algorithm D. We see that

$$\begin{aligned}
&\Pr[\, D(G_1(U_n)) = 1 \,] \;-\; \Pr[\, D(U_{n+1}) = 1 \,] \\
&= \frac{1}{l(n)} \sum_{i=1}^{l(n)} (\; \Pr[\, D(G_1(U_n)) = 1 \;|i] \;-\; \Pr[\, D(U_{n+1}) = 1 \;|i] \;). \qquad (**)
\end{aligned}$$

Note that

$$\begin{aligned}
\Pr[\, D(G_1(U_n)) = 1|i] &= \Pr\Big[\, A(Z_{1..(l(n)-i)} \circ [G_1(U_n)]_1 \circ g^{(i-1)}([G_1(U_n)]_{(2..n+1)}) = 1 \,\Big] \\
&= \Pr\Big[\, A(H^i) = 1 \,\Big]
\end{aligned}$$

and

$$\begin{aligned}
\Pr[\, D(U_{n+1}) = 1|i] &= \Pr\Big[\, A(Z_{1..(l(n)-i)} \circ [U_{n+1}]_1 \circ g^{(i-1)}([U_{n+1}]_{2..n+1})) = 1 \,\Big] \\
&= \Pr\Big[\, A(H^{i-1}) = 1 \,\Big]
\end{aligned}$$

So equation $(**)$ is

$$\begin{aligned}
&\frac{1}{l(n)} \sum_{i=1}^{l(n)} \Big(\Pr\Big[\, A(H^i) = 1 \,\Big] \;-\; \Pr\Big[\, A(H^{i-1}) = 1 \,\Big]\Big) \\
&= \frac{1}{l(n)} \Big(\Pr\Big[\, A(H^{l(n)}) = 1 \,\Big] \;-\; \Pr\Big[\, A(H^0) = 1 \,\Big] \Big) \\
&= \frac{1}{l(n)} \Big(\Pr[\, A(G(U_n)) = 1 \,] \;-\; \Pr\Big[\, A(U_{l(n)}) = 1 \,\Big] \Big)
\end{aligned}$$

so

$$\Pr[\, D(G_1(U_n)) = 1 \,] \;-\; \Pr[\, D(U_{n+1}) = 1 \,] \;\geq\; \frac{1}{l(n)p(n)}$$

■

4 On Using Pseudo-Random Generators

Suppose we have a probabilistic polynomial time algorithm A, which on input of length n uses $m(n)$, random bits. Algorithm A may solve either search problem for some relation or decision problem for some language L. Our claim will be that for all $\varepsilon > 0$ there exists a probabilistic polynomial time algorithm A' that uses only n^{ε} random bits and "behaves" in the same way that A does.

The construction of A' bases on assumption that we are given pseudo-random generator $G : \{0,1\}^{n^{\varepsilon}} \rightarrow \{0,1\}^{m(n)}$. Recall that $A(x, R)$ that A is running on input x with coins R.

Algorithm A'
Input $x \in \{0,1\}^n$
Choose $S \in_R \{0,1\}^{n^{\varepsilon}}$
$R \leftarrow G(s)$ (generate the coin tosses)
Return $A(x, R)$ (run A on input x using coins R)
end.

Proposition 4.1 (imformal): *It is infeasible given 1^n to find $x \in \{0,1\}^n$, such that the "behaviour" of $A'(x)$ is substantially diferent from $A(x)$.*

The meaning of this proposition depends on the computational problem solved by A. In case A solves some NP-search problem, the proposition asserts that it is hard (i.e., feasible only with negligable probability) to find large x's such that A can find the solution for x, and $A'(x)$ will fail to do so. In case A computes some function the proposition applies too.

> Oded's Note: *But the proposition may not hold if A solves a search problem in which instances have many possible solutions and are not efficiently verifiable (as in NP-search problems).*

Below we prove the proposition for the case of decision problems (and the proof actually extends to any function computation). We assume that A gives the correct answer with probability bounded away from 1/2.

Proof: Suppose we have a finder F, which works in polynomial time , $F(1^n) = x \in \{0,1\}^n$, such that

$$\Pr\left[A'(x) = X_L(x)\right] \;\leq\; \frac{1}{2}$$

where X_L is the characteristic function of a language decideable by A (i.e., $\Pr[A(x) = X_L(x)] \geq 2/3$ for all x's). Construct a distinguisher D as follows:

Begin Algorithm D
Input $\alpha \in \{0,1\}^{m(n)}$
$x \leftarrow F(1^n)$
$v \leftarrow X_L(x)$ with overwhelmingly high probability (i.e., invoke $A(x)$ polynomially many times and take a majority vote).
$w \leftarrow A(x, \alpha)$
If $v = w$ **Then** **Return** 1
Else **Return** 0

end.

D is contradicts the pseudorandomness of G because

$$A(x,\alpha) = \begin{cases} X_L(x), \ w.p. \ \geq \frac{2}{3}, & \alpha \sim U_{m(n)} \\ X_L(x) = A'(x), \ w.p. \ \leq \frac{1}{2}, & \alpha \sim G(U_{n^\varepsilon}) \end{cases} \tag{1}$$

Furthermore, with probability at least 0.99, the value v found by $D(x)$ equals $X_L(x)$. Thus,

$$\begin{aligned} \Pr[D(U_{m(n)}) = 1] &> 0.66 - 0.01 \ = \ 0.65 \\ \Pr[D(G(U_{n^\epsilon})) = 1] &\leq 0.5 + 0.01 \ < \ 0.55 \end{aligned}$$

which provides the required gap. ■

> Oded's Note: *Note that for a strong notion of pseudorandom generators, where the output is indistinguishable from random by small circuits we can prove a stronger result; that is, that there are only finitely many x's on which A' behaves differently than A. Thus, in case of decision algorithms, by minor modification to A', we can make A' accept the same language as A.*

5 Relation to one-way functions

An important property of a pseudo-random generator $G(S)$ that it turns the seed into the sequence $x = G(S)$ in polynomial time. But the inverse operation of finding the seed S from $G(S)$ would be hard (or else pseudorandomness is violated as shown below). A pseudo-random generator is however, not just a function that hard to invert it also stretches the input into the larger sequence that look random. Still pseudo-random generators can be related to functions which are "only" easy to compute and hard to invert, as defined next.

Definition 11 (One-way functions – OWF): *A function $f : \{0,1\}^* \longrightarrow \{0,1\}^*$ such that $\forall x \ |f(x)| = |x|$ is* one-way *if :*

- *there is exists polynomial time algorithm A, such that $\forall x \ A(x) = f(x)$*
- *for all probabilistic polynomial time A' and for all polynome $p()$ and for all sufficiently large n's :*

$$\Pr\left[A'(f(U_n)) = f^{-1} \circ f(U_n)\right] < \frac{1}{p(n)}$$

In other words this function must be easy computed and hard inverted. Note an important feauture: the inversion algotihm must fail almost always. But the probability distribution used here is not uniform over all $f(x)$; rather, it is the distribution $f(x)$ when x is choosen uniformly.

> Oded's Note: *The requirement that the function be length preserving (i.e., $|f(x)| = |x|$ for all x's) may be relaxed as long as the length of $f(x)$ is polynomially related to $|x|$. In contrast a function like $f(x) \stackrel{\text{def}}{=} |x|$ would be "one-way" for a trivial and useless reason (on input n in binary one cannot print an n-bit string in polynomial (in $\log n$) time).*

Comment: A popular candidate to be one-way function is based on the conjectured intractability of the integer factorization problem. The length of input and output to the function will not be exactly n, only polynomial in n:
The factoring problem. Let $x, y > 1$ be n-bit integers. Define

$$f(x, y) = x * y$$

When x, y are n-bit primes, it is believed that finding x, y from $x * y$ is computationaly difficult.

> Oded's Note: *So the above should be hard to invert in these cases which occur at density $\approx 1/n^2$. This does not satisfy the definition of one-wayness which requires hardness of inversion almost everywhere, but suitable amplification can get us there. Alternatively, we can redefine the function f so that $f(x, y) = prime(x) * prime(y)$, where $prime()$ is a procedure which uses the input string to generate a large prime so that when the input is a random n-bit string the output is a random $n/O(1)$-bit prime. Such efficient procedures are known to exist. Using less sophisticated methods one can easily construct a procedure which uses n-bits to produce a prime of length $\sqrt[3]{n}/O(1)$.*

Theorem 12 *Pseudo-Random Generators exist if and only if One-Way Functions exist.*

So the computational hardness and pseudorandomness are strongly connect each other. If we have the created randomness we can create the hardness, and vice versa. Let us prove one part of the theorem and give hints to special case of other.

PRG $\Longrightarrow$ OWF: Consider pseudo-random generator $G : \{0,1\}^n \to \{0,1\}^{2n}$. Let us define function $f : \{0,1\}^{2n} \to \{0,1\}^{2n}$ as follows:

$$f(xy) = G(x) \qquad (|x| = |y| = n).$$

We claim, that f is one-way function, and the proof is by contradiction :

Suppose probabilistic polynomial time algorithm A' inverts f with success probability greater than $\frac{1}{p(n)}$, where $p(n)$ is polynom.

Consider a distinguisher D :
input: α, $\alpha \in \{0,1\}^{2n}$
$xy \leftarrow A'(\alpha)$
if $f(xy) = \alpha$ return 1
otherwise return 0.

$$\begin{aligned}
\Pr\left[D(G(U_n)) = 1\right] &= \Pr\left[D(f(U_n)) = 1\right] \\
&= \Pr\left[f(A'(f(U_n))) = f(U_n)\right] \\
&= \Pr\left[A'(f(U_n)) \in f^{-1}f(U_n)\right] \\
&> \frac{1}{p(n)}
\end{aligned}$$

where the last inequality is due to the contradiction hypothesis. On the other hand, there are at most 2^n strings of length $2n$ which have a preimage under G (and so under f). Thus, a uniformly selected random string of length $2n$ has a preimage under f with probability at most $2^n/2^{2n}$. It follows that

$$\begin{aligned}
\Pr[D(U_{2n})=1] &= \Pr[f(A'(U_{2n}))=U_{2n}] \\
&\leq \Pr[U_{2n} \text{ is in the image of } f] \\
&\leq \frac{2^n}{2^{2n}} = 2^{-n}
\end{aligned}$$

Thus,

$$\Pr[D(G(U_n))=1] - \Pr[D(U_{2n})=1] > \frac{1}{p(n)} - \frac{1}{2^n} > \frac{1}{q(n)}$$

For some polynome $q()$

OWF $\Longrightarrow$ PRG:

> Oded's Note: *The rest of this section is an overview of what is shown in detail in the next lecture (i.e., Lecture 14).*

Let us demonstrate the other direction and build an Pseudo-Random Generator if we have OWF of special form. Suppose the function $f:\{0,1\}^n \to \{0,1\}^n$ is not only OWF but it is also $1-1$. So it is a permutation of strings of length n. Assume that we can get a random bit b from the input, such that b will be hard to "predict" from the output of f. In this case we can construct a Pseudo-Random Generator as a concatenation of $f(x)$ and b.

Definition 13 (Hardcore):
Let f be one-way function, $b:\{0,1\}^* \longrightarrow \{0,1\}$ is a hardcore of f if:

- *$\exists$ polinomial time algorithm A, such that $\forall t A(t)=b(t)$*
- *$\forall$ probabilistic polynomial time algorithm A' $\forall$ polynom $p(.)$ $\forall$ sufficiently large n's*

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

In other words this function must be easy to compute and hard to predict out of $f(x)$.

The following theorem can be proven:

Theorem 14 *If f is OW, $f'(x,y) = f(x)\circ y$, $(|x|=|y|)$ then $b(x,y)=\sum_{i=1}^{n} x_i y_i \ (mod\ 2)$ is a hardcore of f.*

This theorem would be proven in next lecture. Now we can construct a Pseudo-Random Generator G as follows:

$$G(s) = f'(s)\circ b(s)$$

The two first properties of G (poly-time and stretching) are trivial. The pseudorandomness of G follows from the fact that its first n output bits are uniformly distributed and the last bit is unpredictable. Unpredictability translates to indistinguishability, as will be shown in the next lecture.

Oded's Note: *Being in the process of writing an essay on pseudorandomness, it feels a good idea to augment the notes of the current lecture by a draft of this essay. The lecture notes actually expand on the presentation in Sections 6.2 and 6.3. The other sections in this essay go beyond the lecture notes.*

Appendix: An essay by O.G.

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

6.1 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., [4]), initiated by Shannon [21], is rooted in probability theory and is focused at distributions which 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). 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 seeds.

The second theory (cf., [16, 17]), due to Solomonov [22], Kolmogorov [15] and Chaitin [3], 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) which generates the object. 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 seeds.

The third theory, initiated by Blum, Goldwasser, Micali and Yao [12, 1, 24], is rooted in complexity theory and is the focus of this essay. This approach is explicitly aimed at providing a notion of perfect randomness which nevertheless allows to efficiently generate perfect random strings from shorter random seeds. The heart of this approach is the suggestion to view objects as equal if they cannot be told apart by any efficient procedure. Consequently a distribution which 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 rather relative to an observer (and its computational abilities). To demonstrate this approach, let us consider the following mental experiment.

> Alice and Bob play "head or tail" in one of the following four ways. In all of them Alice flips a coin high in the air, and Bob is asked to guess its outcome *before* the coin hits the floor. The alternative ways differ by the knowledge Bob has before making his guess. In the first alternative, Bob has to announce his guess before Alice flips the coin. Clearly, in this case Bob wins with probability 1/2. In the second alternative,

Bob has to announce his guess while the coin is spinning in the air. Although the outcome is *determined in principle* by the motion of the coin, Bob does not have accurate information on the motion and thus we believe that also in this case Bob wins with probability 1/2. The third alternative is similar to the second, except that Bob has at his disposal sophisticated equipment capable of providing accurate *information* on the coin's motion as well as on the environment effecting the outcome. However, Bob cannot process this information in time to improve his guess. In the fourth alternative, Bob's recording equipment is directly connected to a *powerful computer* programmed to solve the motion equations and output a prediction. It is conceivable that in such a case Bob can improve substantially his guess of the outcome of the coin.

We conclude that the randomness of an event is relative to the information and computing resources at our disposal. Thus, a natural concept of pseudorandomness arises – a distribution is *pseudorandom* if no efficient procedure can distinguish it from the uniform distribution, where efficient procedures are associated with (probabilistic) polynomial-time algorithms.

6.2 The Definition of Pseudorandom Generators

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

1. *Efficiency*: The generator has to be efficient. We associate efficient computations with those conducted within time which is polynomial in the length of the input. Consequently, we postulate that the generator has to be implementable by a deterministic polynomial-time algorithm.

   This algorithm takes as input a *seed*, as mentioned above. The seed captures a bounded amount of randomness used by a device which "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.

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).[1] Thus, the objects in question are infinite

[1] 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.

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 $\{D_n\}_{n\in\mathsf{N}}$, where for some function $\ell : \mathsf{N} \mapsto \mathsf{N}$, the support of each D_n is a subset of $\{0,1\}^{\ell(n)}$. Furthermore, typically ℓ will be a positive polynomial.

> Oded's Note: *In this essay, I've preferred the traditional mathematical notations. Specifically, I have used distributions (over strings) rather than our non-standard "random variables" (which range over strings). For a distribution D, the traditional notation $x\sim D$ means x selected according to distribution D.*

Definition 6.1 (Computational Indistinguishability [12, 24]): *Two probability ensembles, $\{X_n\}_{n\in\mathsf{N}}$ and $\{Y_n\}_{n\in\mathsf{N}}$, are called* computationally indistinguishable *if for any probabilistic polynomial-time algorithm A, for any positive polynomial p, and for all sufficiently large n's*

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

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 which 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 which occurs with negligible probability (as a function of a parameter n), will occur with negligible probability also if the experiment is repeated for poly(n)-many times.

We note that computational indistinguishability is a strictly more liberal notion than statistical indistinguishability (cf., [24, 10]). An important case is the one of distributions generated by a pseudorandom generator as defined next.

Definition 6.2 (Pseudorandom Generators [1, 24]): *A deterministic polynomial-time algorithm G is called a* pseudorandom generator *if there exists a* stretching function, $\ell : \mathsf{N} \mapsto \mathsf{N}$, *so that the following two probability ensembles, denoted $\{G_n\}_{n\in\mathsf{N}}$ and $\{R_n\}_{n\in\mathsf{N}}$, 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

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

Thus, pseudorandom generators are efficient (i.e., polynomial-time) deterministic programs which 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 6.3 (typical application of pseudorandom generators): *Let A be a probabilistic algorithm, and $\rho(n)$ denote a* (polynomial) *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 $\ell:\mathsf{N}\mapsto\mathsf{N}$. Then A_G is a randomized algorithm which 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 *noticeablly different* from the one of A, although A_G may use much fewer coin tosses than A. That is

Theorem 6.4 *Let A and G be as above. 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*

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

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

and the probabilities are taken over the U_m's as well as over the coin tosses of F and D.

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

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

Theorem 6.5 [9]: *Let G be a pseudorandom generator with stretch function $\ell(n)=n+1$, and ℓ' be any polynomially bounded stretch function, which 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*

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

is a pseudorandom generator with stretch function ℓ'.

Proof Sketch: The theorem is proven using the *hybrid technique* (cf., Sec. 3.2.3 in [5]): One considers distributions H_n^i (for $i=0,...,\ell(n)$) defined by $U_i^{(1)} P_{\ell(n)-i}(U_n^{(2)})$, where $U_i^{(1)}$ and $U_n^{(2)}$ are independent uniform distributions (over $\{0,1\}^i$ and $\{0,1\}^n$, respectively), and $P_j(x)$ denotes the j-bit long prefix of $G'(x)$. The extreme hybrids correspond to $G'(U_n)$ and $U_{\ell(n)}$, whereas distinguishability of neighboring hybrids can be worked into distinguishability of $G(U_n)$ and U_{n+1}. Loosely speaking, suppose one could distinguish H_n^i from H_n^{i+1}. Then, using $P_j(s) = G_2(s)P_{j-1}(G_1(s))$ (for $j \geq 1$), this means that one can distinguish $H_n^i \equiv (U_i^{(1)}, G_2(U_n^{(2)}), P_{(\ell(n)-i)-1}(G_1(U_n^{(2)})))$ from $H_n^{i+1} \equiv (U_i^{(1)}, U_1^{(1')}, P_{\ell(n)-(i+1)}(U_n^{(2')}))$. Incorporating the generation of $U_i^{(1)}$ and the evaluation of $P_{\ell(n)-i-1}$ into the distinguisher, one could distinguish $(f(U_n^{(2)}), b(U_n^{(2)})) \equiv G_1(U_n)$ from $(U_n^{(2')}, U_1^{(1')}) \equiv U_{n+1}$, in contradiction to the pseudorandomness of G_1. (For details see Sec. 3.3.3 in [5].) ■

6.3 How to Construct Pseudorandom Generators

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

Definition 6.6 (one-way function): *A* one-way function*, f, is a polynomial-time computable function such that for every probabilistic polynomial-time algorithm A', every positive polynomial $p(\cdot)$, and all sufficiently large n's*

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

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

Popular candidates for one-way functions are based on the conjectured intractability of Integer Factorization (cf., [18] for state of the art), the Discrete Logarithm Problem (cf., [19] analogously), and decoding of random linear code [11]. 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 $x \in \{0,1\}^n$. 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 all efficient algorithm, given $f(x)$, can guess $b(x)$ only with success probability which is negligible better than half.

Definition 6.7 (hard-core predicate [1]): *A polynomial-time computable predicate $b : \{0,1\}^* \mapsto \{0,1\}$ 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*

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

Clearly, if b is a hard-core of a 1-1 polynomial-time computable function f then f must be one-way.[2] It turns out that any one-way function can be slightly modified so that it has a hard-core predicate.

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

Theorem 6.8 (A generic hard-core [8]): *Let f be an arbitrary one-way function, and let g be defined by $g(x,r) \stackrel{\text{def}}{=} (f(x),r)$, where $|x|=|r|$. Let $b(x,r)$ denote the inner-product mod 2 of the binary vectors x and r. Then the predicate b is a hard-core of the function g.*

See proof in Apdx C.2 in [6]. Finally, we get to the construction of pseudorandom generators:

Theorem 6.9 (A simple construction of pseudorandom generators): *Let b be a hard-core predicate of a polynomial-time computable 1-1 function f. Then, $G(s) \stackrel{\text{def}}{=} f(s)\,b(s)$ is a pseudorandom generator.*

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

In a sense, the key point in the proof of the above theorem 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 Construction 6.5 and Proposition 6.9, is analogous to the original construction of pseudorandom generators suggested by by Blum and Micali [1]: Given an arbitrary stretch function $\ell : \mathsf{N} \mapsto \mathsf{N}$, a 1-1 one-way function f with a hard-core b, one defines

$$G(s) \stackrel{\text{def}}{=} b(x_1)b(x_2)\cdots b(x_{\ell(|s|)}) \,,$$

where $x_0 = s$ and $x_i = f(x_{i-1})$ for $i = 1,...,\ell(|s|)$. The pseudorandomness of G is established in two steps, using the notion of (next bit) unpredictability. An ensemble $\{Z_n\}_{n\in\mathsf{N}}$ 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-negligiblly higher than $1/2$.

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

Loosely speaking, if one can predict $b(x_i)$ from $b(x_{i+1})\cdots b(x_{\ell(|s|)})$ then one can predict $b(x_i)$ given $f(x_i)$ (i.e., by computing $x_{i+1},...,x_{\ell(|s|)}$ and so obtaining $b(x_{i+1})\cdots b(x_{\ell(|s|)})$), in contradiction to 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., Sec. 3.3.4 in [5]).

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

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 6.10 (On the existence of pseudorandom generators [13]): *Pseudorandom generators exist if and only if one-way functions exist.*

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

The interesting direction is the construction of pseudorandom generators based on any one-way function. In general (when f may not be 1-1) the ensemble $f(U_n)$ may not be pseudorandom, and so Construction 6.9 (i.e., $G(s) = f(s)b(s)$, where b is a hard-core of f) cannot be used *directly*. One idea of [13] is to hash $f(U_n)$ to an almost uniform string of length related to its entropy, using Universal Hash Functions [2]. (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 [13] 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.

6.4 Pseudorandom Functions

Pseudorandom generators allow to efficiently generate long pseudorandom sequences from short random seeds. Pseudorandom functions (defined below) are even more powerful: They allow efficient direct access to a huge pseudorandom sequence (which is infeasible to scan bit-by-bit). Put in other words, pseudorandom functions can replace truly random functions in any efficient application (e.g., most notably in cryptography). That is, pseudorandom functions are indistinguishable from random functions by efficient machines which 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 6.11 (pseudorandom functions [7]): *A* pseudorandom function (ensemble), *with length parameters* $\ell_{\rm D}, \ell_{\rm R} : \mathsf{N} \mapsto \mathsf{N}$, *is a collection of functions* $F \stackrel{\text{def}}{=} \{f_s : \{0,1\}^{\ell_{\rm D}(|s|)} \mapsto \{0,1\}^{\ell_{\rm R}(|s|)}\}_{s \in \{0,1\}^*}$ *satisfying*

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

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

- (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 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 which 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, which 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 6.12 (How to construct pseudorandom functions [7]): *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 $\{f_s:\{0,1\}^{|s|}\mapsto\{0,1\}^{|s|}\}_{s\in\{0,1\}^*}$, 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 $\ell_{\mathrm{D}},\ell_{\mathrm{R}}:\mathsf{N}\mapsto\mathsf{N}$.

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 $\{0,1\}^n\mapsto\{0,1\}^n$, each defined by 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 $|\alpha|=i$, is $G_\beta(s_\alpha)$. The extreme hybrids correspond to our indistinguishability claim (i.e., $H_n^0\equiv f_{U_n}$ and $H_n^n\equiv F_n$), and neighboring hybrids correspond to our indistinguishability hypothesis (specifically, to the indistinguishability of $G(U_n)$ and U_{2n} under multiple samples). ■

We mention that pseudorandom functions have been used to derive negative results in computational learning theory [23] and in complexity theory (cf., Natural Proofs [20]).

6.5 The Applicability of Pseudorandom Generators

Randomness is playing an increasingly important role in computation: It is frequently used in the design of sequential, parallel and distributed algorithms, and is of course central to cryptography. Whereas it is convenient to design such algorithms making free use of randomness, it is also desirable to minimize the usage of randomness in real implementations. 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 which make an economical use of randomness.

Indeed, "pseudo-random number generators" have appeared with the first computers. However, typical implementations use generators which are not pseudorandom according to the above definition. Instead, at best, these generators are shown to pass SOME ad-hoc statistical test (cf., [14]).

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 6.2 aims at such generality, and in fact is tailored to obtain it: The notion of computational indistinguishability, which underlines Definition 6.2, covers all possible efficient applications postulating that for all of them pseudorandom sequences are as good as truly random ones.

6.6 The Intelectual Contents of Pseudorandom Generators

We shortly discuss some intelectual 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 which 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 which are not uniform (and are not even statistically close to a uniform distribution) that nevertheless are indistinguishable from a uniform distribution by any efficient method [24, 10]. Thus, distributions which 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 which 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 powerful (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 which are not uniform and still cannot be distinguished from uniform. Furthermore, the construction of pseudorandom generators rely on conjectures regarding computational difficulty (i.e., the existence of one-way functions), and this is inevitable: given a pseudorandom generator, we can construct one-way functions. Thus, (non-trivial) pseudorandomness and computational hardness can be converted back and forth.

6.7 A General Paradigm

Pseudorandomness as surveyed above can be viewed as an important special case of a general paradigm. A general treatment is provided in [6].

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) which fool any probabilistic polynomial-time observers. A variety of other cases are of interest too, and we briefly discuss some of them.

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 able to do better. Similarly, if we know that the analysis of the algorithm only depends 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 which 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 that we advocate a rigorous formulation of the characteristics of such applications and rigorous construction of generators which fool the type of tests which emerge.

Alternative notions of generator efficiency. The above discussion has focused on one aspect of the pseudorandomness question – the resources or type of the observer (or potential distinguisher). Another important question is whether such pseudorandom sequences can be generated from much shorter ones, and at what cost (or complexity). Throughout this essay we've required the generation process to be at least as efficient as the efficiency limitations of the distinguisher.[4] 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 classe. For example, one may consider generators working in time exponential in the length of the seed. In some cases we loose nothing by being more liberal (i.e., allowing exponential-time generators). To see why, we consider a typical derandomization argument, proceediing in two steps: First one replaces the true randomness of the algorithm by pseudorandom sequences generated from much shorter seeds, and next one goes deterministically over all possible seeds and looks for the most frequent behavior of the modified algorithm. Thus, in such a case the deterministic complexity is anyhow exponential in the seed length. However, constructing exponential time generators may be easier than constructing polynomial-time ones.

References

[4] If fact, we have require 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.

1. 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.

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

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

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

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

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

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

8. 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.

9. O. Goldreich and S. Micali. Increasing the Expansion of Pseudorandom Generators. Manuscript, 1984. Available from `http://theory.lcs.mit.edu/~oded/papers.html`

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

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

12. 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.

13. J. Håstad, R. Impagliazzo, L.A. Levin and M. Luby. Construction of Pseudorandom Generator from any One-Way Function. To appear in *SIAM Journal on Computing.* Preliminary versions by Impagliazzo et. al. in *21st ACM Symposium on the Theory of Computing* (1989) and Håstad in *22nd ACM Symposium on the Theory of Computing* (1990).

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

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

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

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

18. A.M. Odlyzko. The future of integer factorization. *CryptoBytes* (The technical newsletter of RSA Laboratories), Vol. 1 (No. 2), pages 5-12, 1995.

19. 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.

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

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

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

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

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