Chapter 3

Pseudorandom Generators

In this chapter we discuss pseudorandom generators. Loosely speaking, these are efficient deterministic programs which expand short randomly selected seeds into much longer "pseudorandom" bit sequences. Pseudorandom sequences are defined as computationally indistinguishable from truly random sequences by efficient algorithms. Hence, the notion of computational indistinguishability (i.e., indistinguishability by efficient procedures) plays a pivotal role in our discussion of pseudorandomness. Furthermore, the notion of computational indistinguishability, plays a key role also in subsequent chapters, and in particular in the discussion of secure encryption, zero-knowledge proofs, and cryptographic protocols.

In addition to definitions of pseudorandom distributions, pseudorandom generators, and pseudorandom functions, the current chapter contains constructions of pseudorandom generators (and pseudorandom functions) based on various types of one-way functions. In particular, very simple and efficient pseudorandom generators are constructed based on the existence of one-way permutations.

Organization: Basic discussions, definitions and constructions of pseudorandom generators appear in Sections 3.1–3.4: we start with a motivating discussion (Section 3.1), proceed with a general definition of computational indistinguishability (Section 3.2), next present and discuss definitions of pseudorandom generators (Section 3.3), and finally present some simple constructions (Section 3.4). More general constructions are discussed in Section 3.5. Pseudorandom functions are defined and constructed (out of pseudorandom generators) in Section 3.6. (Pseudorandom permutations are discussed in Section 3.7.)

3.1 Motivating Discussion

The nature of randomness has attracted the attention of many people and in particular of scientists in various fields. We believe that the notion of computation, and in particular of efficient computation, provides a good basis for understanding the nature of randomness.
3.1.1 Computational Approaches to Randomness

One computational approach to randomness has been initiated by Solomonov and Kolmogorov in the early 1960’s (and rediscovered by Chaitin in the early 1970’s). This approach is “ontological” in nature. Loosely speaking, a string, $s$, is considered Kolmogorov-random if its length (i.e., $|s|$) equals the length of the shortest program producing $s$. This shortest program may be considered the “simplest” “explanation” to the phenomenon described by the string $s$. Hence, the string, $s$, is considered Kolmogorov-random if it does not possess a simple explanation (i.e., an explanation which is substantially shorter than $|s|$). We stress that one cannot determine whether a given string is Kolmogorov-random or not (and more generally Kolmogorov-complexity is a function that cannot be computed). Furthermore, this approach seems to have no application to the issue of “pseudorandom generators”.

An alternative computational approach to randomness is presented in the rest of this chapter. In contrast to the approach of Kolmogorov, the new approach is behavioristic in nature. Instead of considering the “explanation” to a phenomenon, we consider the phenomenon’s effect on the environment. Loosely speaking, a string is considered pseudorandom if no efficient observer can distinguish it from a uniformly chosen string of the same length. The underlying postulate is that objects that cannot be told apart by efficient procedures are considered equivalent, although they may be very different in nature (e.g., have fundamentally different (Kolmogorov) complexity). Furthermore, the new approach naturally leads to the concept of a pseudorandom generator, which is a fundamental concept with lots of practical applications (and in particular to the area of cryptography).

3.1.2 A Rigorous Approach to Pseudorandom Generators

The approach to pseudorandom generators, presented in this book, stands in contrast to the heuristic approach which is still common in discussions concerning “pseudorandom generators” which are being used in real computers. The heuristic approach consider “pseudorandom generators” as programs which produce bit sequences “passing” several specific statistical tests. The choice of statistical tests, to which these programs are subjected, is quite arbitrary and lacks a systematic foundation. Furthermore, it is possible to construct efficient statistical tests which foil the “pseudorandom generators” commonly used in practice (and in particular distinguish their output from a uniformly chosen string of equal length). Consequently, before using a “pseudorandom generator”, in a new application (which requires “random” sequences), extensive tests have to be conducted in order to detect whether the behaviour of the application when using the “pseudorandom generator” preserves its behaviour when using a “true source of randomness”. Any modification of the application requires new comparison of the “pseudorandom generator” against the “random source”, since the non-randomness of the “pseudorandom generator” may badly effect the modified application (although it did not effect the original application). Furthermore, using such a “pseudorandom generator” for “cryptographic purposes” is highly risky, since the adversary may try to exploit the known weaknesses of the “pseudorandom generator”.
3.2. COMPUTATIONAL INDISTINGUISHABILITY

In contrast the concept of pseudorandom generators, presented below, is a robust one. By definition these pseudorandom generators produce sequences which look random to any efficient observer. It follows that the output of a pseudorandom generator may be used instead of "random sequences" in any efficient application requiring such (i.e., "random") sequences.

3.2 Computational Indistinguishability

The concept of efficient computation leads naturally to a new kind of equivalence between objects. Objects are considered to be computationally equivalent if they cannot be told apart by any efficient procedure. Considering indistinguishable objects as equivalent is one of the basic paradigms of both science and real-life situations. Hence, we believe that the notion of computational indistinguishability is fundamental.

Formulating the notion of computational indistinguishability is done, as standard in computational complexity, by considering objects as infinite sequences of strings. Hence, the sequences \( \{x_n\}_{n \in \mathbb{N}} \) and \( \{y_n\}_{n \in \mathbb{N}} \), are said to be computational indistinguishable if no efficient procedure can tell them apart. In other words, no efficient algorithm, \( D \), can accept infinitely many \( x_n \)'s while rejecting their \( y \)-counterparts (i.e., for every efficient algorithm \( D \) and all sufficiently large \( n \)'s it holds that \( D \) accepts \( x_n \) iff \( D \) accepts \( y_n \)). Objects which are computationally indistinguishable in the above sense may be considered equivalent as far as any practical purpose is concerned (since practical purposes are captured by efficient algorithms and those cannot distinguish these objects).

The above discussion is naturally extended to the probabilistic setting. Furthermore, as we shall see, this extension yields very useful consequences. Loosely speaking, two distributions are called computationally indistinguishable if no efficient algorithm can tell them apart. Given an efficient algorithm, \( D \), we consider the probability that \( D \) accepts (e.g., outputs 1 on input) a string taken from the first distribution. Likewise, we consider the probability that \( D \) accepts a string taken from the second distribution. If these two probabilities are close, we say that \( D \) does not distinguish the two distributions. Again, the formulation of this discussion is with respect to two infinite sequences of distributions (rather than with respect to two fixed distributions). Such sequences are called probability ensembles.

3.2.1 Definition

Definition 3.2.1 (ensembles): Let \( I \) be a countable index set. An ensemble indexed by \( I \) is a sequence of random variables indexed by \( I \). Namely, \( X = \{X_i\}_{i \in I} \), where the \( X_i \)'s are random variables, is an ensemble indexed by \( I \).

We will use either \( \mathbb{N} \) or a subset of \( \{0,1\}^* \) as the index set. Typically, in our applications, an ensemble of the form \( X = \{X_n\}_{n \in \mathbb{N}} \) has each \( X_n \) ranging over strings of length \( n \), whereas an ensemble of the form \( X = \{X_w\}_{w \in \{0,1\}^*} \) will have each \( X_w \) ranging over strings of length
[w]. In the rest of this chapter, we will deal with ensembles indexed by \( \mathbb{N} \), whereas in other chapters (e.g., in the definition of secure encryption and zero-knowledge) we will deal with ensembles indexed by strings. To avoid confusion, we present variants of the definition of computational indistinguishability for each of these two cases. The two formulations can be unified if one associates the natural numbers with their unary representation (i.e., associate \( \mathbb{N} \) and \( \{1^n : n \in \mathbb{N}\} \).

**Definition 3.2.2** (polynomial-time indistinguishability):

1. variant for ensembles indexed by \( \mathbb{N} \): Two ensembles, \( X \overset{\text{def}}{=} \{X_n\}_{n \in \mathbb{N}} \) and \( Y \overset{\text{def}}{=} \{Y_n\}_{n \in \mathbb{N}} \), are indistinguishable in polynomial-time if for every probabilistic polynomial-time algorithm, \( D \), every polynomial \( p(\cdot) \), and all sufficiently large \( n \)'s

\[
|\Pr(D(X_n; 1^n) = 1) - \Pr(D(Y_n; 1^n) = 1)| < \frac{1}{p(n)}
\]

2. variant for ensembles indexed by a set of strings \( S \): Two ensembles, \( X \overset{\text{def}}{=} \{X_w\}_{w \in S} \) and \( Y \overset{\text{def}}{=} \{Y_w\}_{w \in S} \), are indistinguishable in polynomial-time if for every probabilistic polynomial-time algorithm, \( D \), every polynomial \( p(\cdot) \), and all sufficiently long \( w \)'s

\[
|\Pr(D(X_w; w) = 1) - \Pr(D(Y_w; w) = 1)| < \frac{1}{p(|w|)}
\]

The probabilities in the above definition are taken over the corresponding random variables \( X_i \) (or \( Y_i \)) and the internal coin tosses of algorithm \( D \) (which is allowed to be a probabilistic algorithm). The second variant of the above definition will play a key role in subsequent chapters, and further discussion of it is postponed to these places. In the rest of this chapter we refer only to the first variant of the above definition. The string \( 1^n \) is given as auxiliary input to algorithm \( D \) in order to make the first variant consistent with the second one, and in order to make it more intuitive. However, in typical cases, where the length of \( X_n \) (resp. \( Y_n \)) and \( n \) are polynomially related (i.e., \( |X_n| < \text{poly}(n) \) and \( n < \text{poly}(|X_n|) \)) and can be computed one from the other in \( \text{poly}(n) \)-time, giving \( 1^n \) as auxiliary input is redundant.

The following mental experiment may be instructive. For each \( \alpha \in \{0, 1\}^* \), consider the probability, hereafter denoted \( d(\alpha) \), that algorithm \( D \) outputs 1 on input \( \alpha \). Consider the expectation of \( d \) taken over each of the two ensembles. Namely, let \( d_1(n) = E(d(X_n)) \) and \( d_2(n) = E(d(Y_n)) \). Then, \( X \) and \( Y \) are said to be indistinguishable by \( D \) if the difference (function) \( \Delta(n) \overset{\text{def}}{=} |d_1(n) - d_2(n)| \) is negligible in \( n \). A few examples may help to further clarify the definition.

Consider an algorithm, \( D_1 \), which obliviously of the input, flips a 0–1 coin and outputs its outcome. Clearly, on every input, algorithm \( D_1 \) outputs 1 with probability exactly one half, and hence does not distinguish any pair of ensembles. Next, consider an algorithm,
3.2. COMPUTATIONAL INDISTINGUISHABILITY

$D_2$, which outputs 1 if and only if the input string contains more zeros than ones. Since $D_2$ can be implemented in polynomial-time, it follows that if $X$ and $Y$ are polynomial-time indistinguishable then the difference $|\Pr(\omega(X_n) < \frac{n}{2}) - \Pr(\omega(Y_n) < \frac{n}{2})|$ is negligible (in $n$), where $\omega(\alpha)$ denotes the number of 1's in the string $\alpha$. Similarly, polynomial-time indistinguishable ensembles must exhibit the same "profile" (up to negligible error) with respect to any "string statistics" which can be computed in polynomial-time. However, it is not required that polynomial-time indistinguishable ensembles have similar "profiles" with respect to quantities which cannot be computed in polynomial-time (e.g., Kolmogorov Complexity or the function presented right after Proposition 3.2.3).

3.2.2 Relation to Statistical Closeness

Computational indistinguishability is a refinement of a traditional notion from probability theory. We call two ensembles $X \overset{\Delta}{=} \{X_n\}_{n \in \mathbb{N}}$ and $Y \overset{\Delta}{=} \{Y_n\}_{n \in \mathbb{N}}$ 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

$$\Delta(n) \overset{\Delta}{=} \frac{1}{2} \cdot \sum_{\alpha} |\Pr(X_n = \alpha) - \Pr(Y_n = \alpha)|$$

Clearly, if the ensembles $X$ and $Y$ are statistically close then they are also polynomial-time indistinguishable (see Exercise 6). The converse, however, is not true. In particular

Proposition 3.2.3 There exist an ensemble $X = \{X_n\}_{n \in \mathbb{N}}$ so that $X$ is not statistically close to the uniform ensemble, $U \overset{\Delta}{=} \{U_n\}_{n \in \mathbb{N}}$, yet $X$ and $U$ are polynomial-time indistinguishable. Furthermore, $X_n$ assigns all its probability mass to at most $2^{n/2}$ strings (of length $n$).

Recall that $U_n$ is uniformly distributed over strings of length $n$. Although $X$ and $U$ are polynomial-time indistinguishable, one can define a function $f : \{0,1\}^* \rightarrow \{0,1\}$ so that $f$ has average 1 over $X$ while having average almost 0 over $U$ (e.g., $f(x) = 1$ if and only if $x$ is in the range of $X$). Hence, $X$ and $U$ have different "profile" with respect to the function $f$; yet $f$ is (necessarily) impossible to compute in polynomial-time.

Proof: 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

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

The proposition follows from this claim, since polynomial-time distinguishers (even probabilistic ones - see Exercise 7) yield polynomial-size circuits with at least as big a distinguishing gap.
CHAPTER 3. PSEUDORANDOM GENERATORS

The claim is proven using a probabilistic argument (i.e., a counting argument). Let $C_n$ be some fixed circuit with $n$ inputs, and let $p_n \stackrel{\text{def}}{=} \Pr(C_n(U_n) = 1)$. We select, independently and uniformly $2^{n/2}$ strings, denoted $\varepsilon_1, \ldots, \varepsilon_{2^{n/2}}$, in $\{0, 1\}^n$. Define random variables $\zeta_i$ so that $\zeta_i = C_n(\varepsilon_i)$ (these random variables depend on the random choices of the corresponding $\varepsilon_i$'s). Using Chernoff Bound, we get that

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

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 $\varepsilon_1, \ldots, \varepsilon_{2^{n/2}} \in \{0, 1\}^n$, so that for every circuit $C_n$ of size $2^{n/8}$ it holds that

$$|\Pr(C_n(U_n) = 1) - \frac{1}{2^{n/2}} \sum_{i=1}^{2^{n/2}} C_n(\varepsilon_i)| < 2^{-n/8}$$

Letting $X_n$ equal $\varepsilon_i$ with probability $2^{-n/2}$, for every $1 \leq i \leq 2^{n/2}$, the claim follows. □

Proposition 3.2.3 presents a pair of ensembles which are computationally indistinguishable although they are statistically far apart. One of the two ensembles is not constructible in polynomial-time (see Definition 3.2.5 below). Interestingly, a pair of polynomial-time constructible ensembles having the above property (i.e., being both computationally indistinguishable and having a non-negligibly statistical difference) exists only if one-way functions exist. Jumping ahead, we note that this necessary condition is also sufficient. (This follows from the fact that pseudorandom generators give rise to a polynomial-time constructible ensemble which is computationally indistinguishable from the uniform ensemble and yet statistically very far from it.)

3.2.3 Indistinguishability by Repeated Experiments

By Definition 3.2.2, two ensembles are considered computationally indistinguishable if no efficient procedure can tell them apart based on a single sample. We shall now show that "efficiently constructible" computational indistinguishable ensembles cannot be (efficiently) distinguished even by examining several samples. We start by presenting definitions of "indistinguishability by sampling" and "efficiently constructible ensembles".

Definition 3.2.4 (Indistinguishability by sampling): Two ensembles, $X \stackrel{\text{def}}{=} \{X_n\}_{n \in \mathbb{N}}$ and $Y \stackrel{\text{def}}{=} \{Y_n\}_{n \in \mathbb{N}}$, are indistinguishable by polynomial-time sampling if for every probabilistic polynomial-time algorithm, $D$, every two polynomials $m(\cdot)$ and $p(\cdot)$, and all sufficiently large $n$'s

$$|\Pr\left(D(X_n^{(1)}, \ldots, X_n^{(m(n))}) = 1\right) - \Pr\left(D(Y_n^{(1)}, \ldots, Y_n^{(m(n))}) = 1\right)| < \frac{1}{p(n)}$$
where $X_n^{(1)}$ through $X_n^{(m)}$ and $Y_n^{(1)}$ through $Y_n^{(m)}$, are independent random variables with each $X_n^{(i)}$ identical to $X_n$ and each $Y_n^{(i)}$ identical to $Y_n$.

**Definition 3.2.5 (efficiently constructible ensembles):** An ensemble, $X \overset{\text{def}}{=} \{X_n\}_{n \in \mathbb{N}}$, 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 3.2.6** Let $X \overset{\text{def}}{=} \{X_n\}_{n \in \mathbb{N}}$ and $Y \overset{\text{def}}{=} \{Y_n\}_{n \in \mathbb{N}}$ be two polynomial-time constructible ensembles, and suppose that $X$ and $Y$ are indistinguishable in polynomial-time. Then $X$ and $Y$ are indistinguishable by polynomial-time sampling.

An alternative formulation of Theorem 3.2.6 proceeds as follows. For every ensemble $Z \overset{\text{def}}{=} \{Z_n\}_{n \in \mathbb{N}}$ and every polynomial $m(\cdot)$ define the $m(\cdot)$-product of $Z$ as the ensemble $\{(Z_n^{(1)}, \ldots, Z_n^{(m(n))})\}_{n \in \mathbb{N}}$, where the $Z_n^{(i)}$'s are independent copies of $Z_n$. Theorem 3.2.6 asserts that if the ensembles $X$ and $Y$ are polynomial-time 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 $Y$ are polynomial-time indistinguishable.

The information theoretic analogue of the above theorem is quite obvious: if two ensembles are statistically close then also their polynomial-products must be statistically close (see Exercise 5). Adapting the proof to the computational setting requires, as usual, a “reducibility argument”. This argument uses, for the first time in this book, the hybrid technique. The hybrid technique plays a central role in demonstrating the computational indistinguishability of complex ensembles, constructed based on simpler (computational indistinguishable) ensembles. Subsequent application of the hybrid technique will involve more technicalities. Hence, the reader is urged not to skip the following proof.

**Proof:** The proof is by a “reducibility argument”. We show that the existence of an efficient algorithm that distinguishes the ensembles $X$ and $Y$ using several samples, implies the existence of an efficient algorithm that distinguishes the ensembles $X$ and $Y$ using a single sample. The implication is proven using the following argument, which will be latter 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

$$
\Delta(n) \overset{\text{def}}{=} |\Pr(D(X_n^{(1)}, \ldots, X_n^{(m)}) = 1) - \Pr(D(Y_n^{(1)}, \ldots, Y_n^{(m)}) = 1)| > \frac{1}{p(n)}
$$

where $m \overset{\text{def}}{=} m(n)$, and the $X_n^{(i)}$'s and $Y_n^{(i)}$'s are as in Definition 3.2.4. 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 3.2.2).
CHAPTER 3. PSEUDORANDOM GENERATORS

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 $X_n$ and $m - k$ independent copies of $Y_n$. Namely,

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

where $X_n^{(i)}$ 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, $H_n^m = X_n^{(1)}, \ldots, X_n^{(m)}$; whereas $H_n^0 = Y_n^{(1)}, \ldots, Y_n^{(m)}$.

By our hypothesis, algorithm $D$ can distinguish the extreme hybrids (i.e., $H_n^0$ and $H_n^m$). As 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 neighbouring hybrids. It follows that $D$, although not "designed to work on general hybrids," can distinguish a pair of neighbouring hybrids. The punch-line is that, algorithm $D$ can be easily modified into an algorithm $D'$ which distinguishes $X$ and $Y$. Details follow.

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

Clearly, $D'$ can be implemented in probabilistic polynomial time. It is also easy to verify the following claims.

Claim 3.2.6.1:

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

and

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

Proof: By construction of algorithm $D'$, we have

$$D'(\alpha) = D(X_n^{(1)}, \ldots, X_n^{(k)}, \alpha, Y_n^{(k+2)}, \ldots, Y_n^{(m)})$$

Using the definition of the hybrids $H_n^k$, the claim follows. □

Claim 3.2.6.2:

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

Proof: Using Claim 3.2.6.1 for the first equality, we get

$$|\Pr(D'(X_n)=1) - \Pr(D'(Y_n)=1)|$$
\[ \frac{1}{m} \cdot \left| \sum_{k=0}^{m-1} \Pr(D(H_{n}^{k+1}) = 1) - \Pr(D(H_{n}^{k}) = 1) \right| \\
= \frac{1}{m} \cdot \left| \Pr(D(H_{n}^{\mu}) = 1) - \Pr(D(H_{n}^{\nu}) = 1) \right| \\
= \Delta(n) \]

The last equality follows by observing that \( H_{n}^{\mu} = X_{n}^{(1)}; \ldots; X_{n}^{(m)} \) and \( H_{n}^{\nu} = Y_{n}^{(1)}; \ldots; Y_{n}^{(m)} \), and using the definition of \( \Delta(n) \). \( \Box \)

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. \( \blacksquare \)

It is worthwhile to give some thought to the hybrid technique (used for the first time in the above proof). 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. **Neighbouring 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 (specifically computational indistinguishability) of the basic ensembles to knowledge (specifically computational indistinguishability) of any pair of neighbouring hybrids. Typically, it is required to efficiently transform strings in the range of a basic hybrid 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 neighbouring hybrid. (In the proof of Theorem 3.2.8, the hypothesis that both \( X \) and \( Y \) are polynomial-time constructible is instrumental for such 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 neighbouring 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" and once it exists we can apply it to any input of our choice and analyze its performance on arbitrary input distributions.

3.2.4 Pseudorandom Ensembles

A special, yet important, case of computationally indistinguishable ensembles is the case in which one of the ensembles is uniform. Ensembles which are computational indistinguishable from the a uniform ensemble are called pseudorandom. Recall that \( U_m \) denotes a random variable uniformly distributed over the set of strings of length \( m \). The ensemble \( \{ U_n \}_{n \in \mathbb{N}} \) is called the

**Definition 3.2.7 (pseudorandom ensembles):** Let \( U \overset{\text{def}}{=} \{ U_n \}_{n \in \mathbb{N}} \) be a uniform ensemble, and \( X \overset{\text{def}}{=} \{ X_n \}_{n \in \mathbb{N}} \) be an ensemble. The ensemble \( X \) is called pseudorandom if \( X \) and \( U \) are indistinguishable in polynomial-time.

We stress that \(|X_n|\) is not necessarily \( n \) (whereas \(|U_n| = n\)). In fact, with high probability \(|X_n| = I(n)\).

In the above definition, as in the rest of this book, pseudorandomness is a shorthand for "pseudorandomness with respect to polynomial-time".

3.3 Definitions of Pseudorandom Generators

Pseudorandom ensembles, defined above, can be used instead of uniform ensemble in any efficient application without noticeable degradation in performance (otherwise the efficient application can be transformed into an efficient distinguisher of the supposedly-pseudorandom ensemble from the uniform one). Such a replacement is useful only if we can generate pseudorandom ensembles at a cheaper cost than required to generate a uniform ensemble. The cost of generating an ensemble has several aspects. Standard cost considerations are reflected by the time and space complexities. However, in the context of randomized algorithms, and in particular in the context of generating probability ensembles, a major cost consideration is the quantity and quality of the randomness source used by the algorithm. In particular, in many applications (and especially in cryptography), it is desirable to generate pseudorandom ensembles using as little randomness as possible. This leads to the definition of a pseudorandom generator.

3.3.1 * A General Definition of Pseudorandom Generators

**Definition 3.3.1 (pseudorandom generator):** A pseudorandom generator is a deterministic polynomial-time algorithm, \( G \), satisfying the following two conditions:
3.3. DEFINITIONS OF PSEUDORANDOM GENERATORS

1. expansion: for every \( s \in \{0, 1\}^* \) it holds that \( |G(s)| > |s| \).

2. pseudorandomness: the ensemble \( \{G(U_n)\}_{n \in \mathbb{N}} \) is pseudorandom.

The input, \( s \), to the generator is called its seed. It is required that a pseudorandom generator \( G \) always outputs a string longer than its seed, and that \( G \)'s output, on a uniformly chosen seed, is pseudorandom. In other words, the output of a pseudorandom generator, on a uniformly chosen seed, must be polynomial-time indistinguishable from uniform, although it cannot be uniform (or even statistically close to uniform). To justify the last statement consider a uniform ensemble \( \{U(n)\}_{n \in \mathbb{N}} \) that is polynomial-time indistinguishable from the ensemble \( \{G(U_n)\}_{n \in \mathbb{N}} \) (such a uniform ensemble must exist by the pseudorandom property of \( G \)). We first claim that \( l(n) > n \), since otherwise an algorithm that on input \( 1^n \) and a string \( \alpha \) outputs 1 if and only if \( |\alpha| > n \) will distinguish \( G(U_n) \) from \( U(n) \) (as \( |G(U_n)| > n \) by the expansion property of \( G \)). It follows that \( l(n) \geq n + 1 \). We next bound from below the statistical difference between \( G(U_n) \) and \( U(n) \); as follows

\[
\sum_x |\Pr(U(n) = x) - \Pr(G(U_n) = x)| \geq \sum_{x \in \{0, 1\}^*} |\Pr(U(n) = x) - \Pr(G(U_n) = x)|
= \left( 2^{-n} - 2^{-n} \right) \cdot 2^{-n} \\
\geq \frac{1}{2}
\]

It can be shown, see Exercise 10, that all the probability mass of \( G(U_n) \), except for a negligible (in \( n \)) amount, is concentrated on strings of the same length and that this length equals \( l(n) \), where \( \{G(U_n)\}_{n \in \mathbb{N}} \) is polynomial-time indistinguishable from \( \{U(n)\}_{n \in \mathbb{N}} \). For simplicity, we consider in the sequel, only pseudorandom generators \( G \) satisfying \( |G(s)| = l(|s|) \) for all \( x \)'s.

3.3.2 Standard Definition of Pseudorandom Generators

**Definition 3.3.2 (pseudorandom generator - standard definition):** A pseudorandom generator is a deterministic polynomial-time algorithm, \( G \), satisfying the following two conditions:

1. expansion: there exists a function \( l : \mathbb{N} \rightarrow \mathbb{N} \) so that \( l(n) > n \) for all \( n \in \mathbb{N} \), and \( |G(s)| = l(|s|) \) for all \( s \in \{0, 1\}^* \).

The function \( l \) is called the expansion factor of \( G \).

2. pseudorandomness (as above): the ensemble \( \{G(U_n)\}_{n \in \mathbb{N}} \) is pseudorandom.

Again, we call the input to the generator a seed. The expansion condition requires that the algorithm \( G \) maps \( n \)-bit long seeds into \( l(n) \)-bit long strings, with \( l(n) > n \). The pseudorandomness condition requires that the output distribution, induced by applying algorithm \( G \) to a uniformly chosen seed, is polynomial-time indistinguishable from uniform (although it is not statistically close to uniform - see justification in previous subsection).
The above definition says little about the expansion factor \( l: \mathbb{N} \rightarrow \mathbb{N} \). We merely know that for every \( n \) it holds that \( l(n) \geq n + 1 \), that \( l(n) \leq \text{poly}(n) \), and that \( l(n) \) can be computed in time polynomial in \( n \). Clearly, a pseudorandom generator with expansion factor \( l(n) = n + 1 \) is of little value in practice, since it offers no significant saving in coin tosses. Fortunately, as shown in the subsequent subsection, even pseudorandom generators with such small expansion factor can be used to construct pseudorandom generators with any polynomial expansion factor. Hence, for every two expansion factors \( l_1: \mathbb{N} \rightarrow \mathbb{N} \) and \( l_2: \mathbb{N} \rightarrow \mathbb{N} \), that can be computed in \( \text{poly}(n) \)-time, there exists a pseudorandom generator with expansion factor \( l_1 \), if and only if there exists a pseudorandom generator with expansion factor \( l_2 \). This statement is proven by using a pseudorandom generator with expansion factor \( l_1(n) \overset{\text{def}}{=} n + 1 \) to construct, for every polynomial \( p(\cdot) \), a pseudorandom generator with expansion factor \( p(n) \). Note that a pseudorandom generator with expansion factor \( l_1(n) \overset{\text{def}}{=} n + 1 \) can be derived from any pseudorandom generator (even from one in the general sense of Definition 3.3.1).

3.3.3 Increasing the Expansion Factor of Pseudorandom Generators

Given a pseudorandom generator, \( G_1 \), with expansion factor \( l_1(n) = n + 1 \), we construct a pseudorandom generator \( G \) with polynomial expansion factor, as follows.

Construction 3.3.3 Let \( G_1 \) a deterministic polynomial-time algorithm mapping strings of length \( n \) into strings of length \( n + 1 \), and let \( p(\cdot) \) be a polynomial. Define \( G(s) = \sigma_1 \cdots \sigma_{p(|s|)} \), where \( s_0 \overset{\text{def}}{=} s \), the bit \( \sigma_1 \) is the first bit of \( G_1(s_{i-1}) \), and \( s_i \) is the \( |s| \)-bit long suffix of \( G_1(s_{i-1}) \), for every \( 1 \leq i \leq p(|s|) \). (i.e., \( \sigma_i s_i = G_1(s_{i-1}) \))

Hence, on input \( s \), algorithm \( G \) applies \( G_1 \) for \( p(|s|) \) times, each time on a new seed. Applying \( G_1 \) to the current seed yields a new seed (for the next iteration) and one extra bit (which is being output immediately). The seed in the first iteration is \( s \) itself. The seed in the \( i \)-th iteration is the \( |s| \)-long suffix of the string obtained from \( G_1 \) in the previous iteration. Algorithm \( G \) outputs the concatenation of the "extra bits" obtained in the \( p(|s|) \) iterations. Clearly, \( G \) is polynomial-time computable and expands inputs of length \( n \) into output strings of length \( p(n) \).

Theorem 3.3.4 Let \( G_1 \), \( p(\cdot) \), and \( G \) be as in Construction 3.3.3 (above). Then, if \( G_1 \) is a pseudorandom generator then so is \( G \).

Intuitively, the pseudorandomness of \( G \) follows from that of \( G_1 \) by replacing each application of \( G_1 \) by a random process which on input \( s \) outputs \( \sigma s \), where \( \sigma \) is uniformly chosen in \( \{0,1\} \). Loosely speaking, the indistinguishability of a single application of the random process from a single application of \( G_1 \) implies that polynomially many applications of the random process are indistinguishable from polynomially many applications of \( G_1 \). The actual proof uses the hybrid technique.
3.3. DEFINITIONS OF PSEUDORANDOM GENERATORS

Using Claim 3.3.4.1, our claim follows. □

Let \( d^k(n) \) denote the probability that \( B \) outputs 1 on input taken from the hybrid \( H^k_{p(n)} \) (i.e., \( d^k(n) \triangleq \Pr(B(H^k_{p(n)} = 1)) \)). Recall that \( H^0_{p(n)} \) equals \( G(U_n) \), whereas \( H^{p(n)}_{p(n)} \) equals \( U^{p(n)}_n \). Hence, \( d^0(n) = \Pr(B(G(U_n)) = 1) \), \( d^{p(n)}(n) = \Pr(B(U^{p(n)}_n) = 1) \), and \( \Delta(n) = |d^0(n) - d^{p(n)}(n)| \). Combining these facts with Claim 3.3.4.2, we get:

\[
\frac{\Pr(B'(G_1(U_n)) = 1) - \Pr(B'(U_{n+1}) = 1)}{p(n)} = \frac{1}{p(n)} \sum_{k=0}^{p(n)-1} [d^k(n) - d^{k+1}(n)]
\]

\[
= \frac{\Delta(n)}{p(n)}
\]

Recall that by our (contradiction) hypothesis \( \Delta(n) \geq \frac{1}{q(n)} \); for infinitely many \( n \)'s. Contradiction to the pseudorandomness of \( G_1 \) follows. □

3.3.4 The Significance of Pseudorandom Generators

Pseudorandom generators have the remarkable property of being efficient "amplifiers/expander of randomness". Using very little randomness (in form of a randomly chosen seed) they produce very long sequences which look random with respect to any efficient observer. Hence, the output of a pseudorandom generator may be used instead of "random sequences" in any efficient application requiring such (i.e., "random") sequences. The reason being that such an application may be viewed as a distinguisher. In other word, if some efficient algorithm suffers noticeable degradation in performance when replacing the random sequences it uses by pseudorandom one, then this algorithm can be easily modified into a distinguisher contradicting the pseudorandomness of the later sequences.

The generality of the notion of a pseudorandom generator is of great importance in practice. Once you are guaranteed that an algorithm is a pseudorandom generator you can use it in every efficient application requiring "random sequences" without testing the performance of the generator in the specific new application.

The benefits of pseudorandom generators to cryptography are innumerable (and only the most important ones will be presented in the subsequent chapters). The reason that pseudorandom generators are so useful in cryptography is that the implementation of all cryptographic tasks requires a lot of "high quality randomness". Thus, producing, exchanging and sharing large amounts of "high quality random bits" at low cost is of primary importance. Pseudorandom generators allow to produce (resp., exchange and/or share) \( \poly(n) \) pseudorandom bits at the cost of producing (resp., exchanging and/or sharing) only \( n \) random bits.

A key property of pseudorandom sequences, that is used to justify the use of such sequences in cryptography, is the unpredictability of the sequence. Loosely speaking, a
sequence is unpredictable if no efficient algorithm, given a prefix of the sequence, can guess its next bit with an advantage over one half that is not negligible. Namely,

**Definition 3.3.5 (unpredictability):** An ensemble \( \{X_n\}_{n \in \mathbb{N}} \) is called unpredictable in polynomial-time if for every probabilistic polynomial-time algorithm \( A \) and every polynomial \( p(\cdot) \) and for all sufficiently large \( n \)'s

\[
\Pr(A(1^n, X_n) = \text{next}_A(1^n, X_n)) < \frac{1}{2} + \frac{1}{p(n)}
\]

where \( \text{next}_A(x) \) returns the \( i + 1 \)st bit of \( x \) if \( A \) on input \( (1^n, x) \) reads only \( i < |x| \) of the bits of \( x \), and returns a uniformly chosen bit otherwise (i.e. in case \( A \) read the entire string \( x \)).

Clearly, pseudorandom ensembles are unpredictable in polynomial-time (see Exercise 16). It turns out that the converse holds as well. Namely, only pseudorandom ensembles are unpredictable in polynomial-time (see Exercise 17).

### 3.3.5 Pseudorandom Generators imply One-Way Functions

Up to this point we have avoided the question of whether pseudorandom generators exist at all. Before saying anything positive, we remark that a necessary condition to the existence of pseudorandom generators is the existence of one-way function. Jumping ahead, we wish to reveal that this necessary condition is also sufficient: hence, pseudorandom generators exist if and only if one-way functions exist. At this point we only prove that the existence of pseudorandom generators implies the existence of one-way function. Namely,

**Proposition 3.3.6** Let \( G \) be a pseudorandom generator with expansion factor \( l(n) = 2n \). Then the function \( f: \{0, 1\}^* \to \{0, 1\}^* \) defined by letting \( f(x, y) \) \( \equiv \) \( G(x) \), for every \( |x| = |y| \), is a strongly one-way function.

**Proof:** Clearly, \( f \) is polynomial-time computable. It is left to show that each probabilistic polynomial-time algorithm invert \( f \) with only negligible probability. We use a "reducibility argument". Suppose, on the contrary, that \( A \) is a probabilistic polynomial-time algorithm which for infinitely many \( n \)'s inverts \( f \) on \( f(U_{2^n}) \) with success probability at least \( \frac{1}{\text{poly}(n)} \).

We will construct a probabilistic polynomial-time algorithm, \( D \), that distinguishes \( U_{2^n} \) and \( G(U_n) \) on these \( n \)'s and reach a contradiction.

The distinguisher \( D \) uses the inverting algorithm \( A \) as a subroutine. On input \( \alpha \in \{0, 1\}^* \), algorithm \( D \) uses \( A \) in order to try to get a preimage of \( \alpha \) under \( f \). Algorithm \( D \) then checks whether the string it obtained from \( A \) is indeed a preimage and halts outputting \( 1 \) in case it is (otherwise it outputs \( 0 \)). Namely, algorithm \( A \) computes \( \beta \leftarrow A(\alpha) \), and outputs \( 1 \) if \( f(\beta) = \alpha \) and \( 0 \) otherwise.

By our hypothesis, for some polynomial \( p(\cdot) \) and infinitely many \( n \)'s,

\[
\Pr(f(A(f(U_{2^n}))) = f(U_{2^n})) > \frac{1}{p(n)}
\]
3.4 Constructions Based on One-Way Permutations

By f's construction the random variable f(U_{2n}) equals G(U_n), and therefore \( \Pr(D(G(U_n)) = 1) > \frac{1}{p(n)} \). On the other hand, by f's construction at most \( 2^n \) different 2n-bit long strings have a preimage under f. Hence, \( \Pr(f(A(U_{2n})) = U_{2n}) \leq 2^{-n} \). It follows that for infinitely many n's

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

which contradicts the pseudorandomness of G. ■

3.4 Constructions based on One-Way Permutations

In this section, we present constructions of pseudorandom generator based on one-way permutations. The first construction has a more abstract flavor, as it uses a single length preserving 1-1 one-way function (i.e., a single one-way permutation). The second construction utilizes the same underlying ideas to present practical pseudorandom generators based on collections of one-way permutations.

3.4.1 Construction based on a Single Permutation

By Theorem 3.3.4 (see Subsection 3.3.3), it suffices to present a pseudorandom generator expanding n-bit long seeds into n + 1-bit long strings. Assuming that one-way permutations (i.e., 1-1 length preserving functions) exist, such pseudorandom generators can be constructed easily. We remind the reader that the existence of one-way permutation implies the existence of one-way permutation with corresponding hard-core predicates. Thus, it suffices to prove the following:

**Theorem 3.4.1** Let f be a length-preserving 1-1 (strongly one-way) function, and let b be a hard-core predicate for f. Then the algorithm G, defined by G(s) \( \triangleq f(s)b(s) \), is a pseudorandom generator.

Intuitively, the ensemble \( \{f(U_n)b(U_n)\}_{n \in \mathbb{N}} \) is pseudorandom since otherwise b(U_n) can be efficiently predicted from f(U_n). The proof merely formalizes this intuition.

**Proof:** We use a "reducibility argument". Suppose, on the contrary, that there exists an efficient algorithm D which distinguishes G(U_n) from U_{n+1}. Recalling that G(U_n) = f(U_n)b(U_n) and using the fact that f induces a permutation on \( \{0, 1\}^n \), we deduce that algorithm D distinguishes f(U_n)b(U_n) from f(U_n)U_1. It follows that D distinguishes f(U_n)b(U_n) from f(U_n)b(U_n), where b(x) is the complement bit of b(x) (i.e., \( \overline{b}(x) \triangleq \{0, 1\} - b(x) \)). Hence, algorithm D provides a good indication of b(U_n) from f(U_n), and can be easily modified into an algorithm guessing b(U_n) from f(U_n), in contradiction to the hypothesis that b is a hard-core predicate of f. Details follows.
Furthermore, if $R^1_i \neq R^1_{m+1}$ then
\[
\Pr \left( R^1_i = R^2_{m+1} \right) = \Pr \left( H^{(2)}(R^1_i) \oplus H^{(2)}(R^1_{m+1}) = R^0_i \oplus R^0_{m+1} \right) = 2^{-m}
\]

Hence, both claims follow. □

Combining the above claims, we conclude that $\Pr(\zeta_m) < \frac{a^d}{2^n}$; and furthermore, given that $\zeta_m$ is false, the answers of $P_{2m}$ have left half uniformly chosen among all $n$-bit strings not appearing as left halves in previous answers, whereas the right half uniformly distributed among all $n$-bit strings. On the other hand, the answers of $K_{2m}$ are uniformly distributed among all $2n$-bit strings not appearing as previous answers. Hence, the statistical difference between the distribution of answers in the two cases (i.e., answers by $P_{2m}$ or by $K_{2m}$) is bounded by $\frac{a^d}{2^n}$. The first part of the proposition follows.

The proof that \{DES_{K_z}^4\}_{z \in \mathbb{N}} is strongly pseudorandom is more complex, yet uses essentially the same ideas. In particular, the event corresponding to $\zeta_m$ is the disjunction of four types of events. Events of the first type are of the form $R^h_k = R^h_j$ for $k \in \{2, 3\}$, where $q_l = (L^0_l, R^0_l)$ and $q_j = (L^0_j, R^0_j)$ are queries of the forward direction. Similarly, events of the second type are of the form $R^h_k = R^h_j$ for $k \in \{2, 1\}$, where $q_i = (L^0_i, R^0_i)$ and $q_j = (L^0_j, R^0_j)$ are queries of the backwards direction. Events of the third type are of the form $R^h_k = R^h_j$ for $k \in \{2, 3\}$, where $q_i = (L^0_i, R^0_i)$ is of the forward direction, $q_j = (L^0_j, R^0_j)$ is of the backward direction, and $j < i$. Similarly, events of the fourth type are of the form $R^h_k = R^h_j$ for $k \in \{2, 1\}$, where $q_i = (L^0_i, R^0_i)$ is of the forward direction, $q_j = (L^0_j, R^0_j)$ is of the backward direction, and $j < i$. As before, one bounds the probability of event $\zeta_m$, and bounds the statistical distance between answers by $K_{2m}$ and answers by \{DES_{K_z}^4\}_{z \in \mathbb{N}} given that $\zeta_m$ is false. □

### 3.8 Miscellaneous

#### 3.8.1 Historical Notes

The notion of computational indistinguishable ensembles was first presented by Goldwasser and Micali (in the context of encryption schemes) [GM82]. In the general setting, the notion first appears in Yao's work which is also the origin of the definition of pseudorandomness [Y82]. Yao also observed that pseudorandom ensembles can be very far from uniform, yet our proof of Proposition 3.2.3 is taken from [GM89a].

Pseudorandom generators were introduced by Blum and Micali [BM82], who defined such generators as producing sequences which are unpredictable. Blum and Micali proved that such pseudorandom generators do exist assuming the intractability of the discrete logarithm problem. Furthermore, they presented a general paradigm for constructing pseudorandom generators, which has been used explicitly or implicitly in all subsequent developments. Other suggestions for pseudorandom generators were made soon after by Goldwasser et. al. [GNT82] and Blum et. al. [BBS82]. Consequently, Yao proved that
the existence of any one-way permutation implies the existence of pseudorandom generators [Y82]. Yao was the first to characterize pseudorandom generators as producing sequences which are computationally indistinguishable from uniform. He also proved that this characterization of pseudorandom generators is equivalent to the characterization of Blum and Micali [BM82].

Generalizations to Yao's result, that one-way permutations imply pseudorandom generators, were proven by Levin [L85] and by Goldreich et. al. [GKL88], culminating with the result of Hastad et. al. [H90, ILL89] which asserts that pseudorandom generators exist if and only if one-way functions exist. The constructions presented in Section 3.5 follow the ideas of [GKL88] and [ILL89]. These constructions make extensive use of universal, hashing functions, which were introduced by Carter and Wegman [CW] and first used in complexity theory by Sipser [S82].

Pseudorandom functions were introduced and investigated by Goldreich et. al. [GGM84]. In particular, the construction of pseudorandom functions based on pseudorandom generators is taken from [GGM84].

Pseudorandom permutations were defined and constructed by Luby and Rackoff [LR86], and our presentation follows their work. However, a better presentation which distills the real structure of the proof (as well as provides related results) has been recently given by Naor and Reingold [NR97].

Author's Note: See proceedings of STOC97.

The hybrid method originates from the work of Goldwasser and Micali [GM82]. The terminology is due to Leonid Levin.

3.8.2 Suggestion for Further Reading

Section 3.5 falls short of presenting the construction of Hastad et. al. [HILL], not to mention proving its validity. Unfortunately, the proof of this fundamental theorem, asserting that pseudorandom generators exist if one-way functions exist, is too complicated to fit in a book of the current nature. The interested reader is thus referred to the original paper of Hastad et. al. [HILL] (which combines the results in [H90, ILL89]) and to Luby's book [L94book].

Author's Note: Pseudorandom generators and functions have many applications to cryptography, some of them were to be presented in other chapters of the book (e.g., on signatures and encryption). The annotated list of references (dating 1999) contains several pointers to works which present applications of pseudorandom functions; e.g., [GGM84b], [G86] and [G87b, 889]. However, in recent years the list of applications has grown considerably.

Simple pseudorandom generators based on specific intractability assumptions are presented in [BM82, BBS82, ACGS84, VV84, KB88]. In particular, [ACGS84] presents pseudoran-
dom generators based on the intractability of factoring, whereas \[K88\] presents pseudorandom generators based on the intractability of discrete logarithm problems. In both cases, the major step is the construction of hard-core predicates for the corresponding collections of one-way permutations.

Proposition 3.2.3 presents a pair of ensembles which are computational indistinguishable although they are statistically far apart. One of the two ensembles is not constructible in polynomial-time. Goldreich showed that a pair of polynomial-time constructible ensembles having the above property (i.e., being both computationally indistinguishable and having a non-negligibly statistical difference) exists if and only if one-way functions exist [G90ipl].

Author's Note: G90ipl has appeared in IPL, Vol. 34, pp. 277–281.

Readers interested in Kolmogorov complexity are referred to [LiVitanyi].


3.8.3 Open Problems

Although Håstad et. al. [HILL] showed how to construct pseudorandom generators given any one-way function, their construction is not practical. The reason being that the “quality” of the generator on seeds of length \( n \) is related to the hardness of inverting the given function on inputs of length \( < \sqrt{n} \). We believe that presenting an efficient transformation of arbitrary one-way functions to pseudorandom generators is one of the most important open problems of the area.

An open problem of more practical importance is to try to present even more efficient pseudorandom generators based on the intractability of specific computational problems like integer factorization. For further details see Subsection 2.7.3.

3.8.4 Exercises

Exercise 1: computational indistinguishability is preserved by efficient algorithms: Let \( \{X_n\}_{n \in \mathbb{N}} \) and \( \{Y_n\}_{n \in \mathbb{N}} \) be two ensembles that are polynomial-time indistinguishable, and let \( A \) be a probabilistic polynomial-time algorithm. Prove that the ensembles \( \{A(X_n)\}_{n \in \mathbb{N}} \) and \( \{A(Y_n)\}_{n \in \mathbb{N}} \) are polynomial-time indistinguishable.

Exercise 2: statistical closeness is preserved by any function: Let \( \{X_n\}_{n \in \mathbb{N}} \) and \( \{Y_n\}_{n \in \mathbb{N}} \) be two ensembles that are statistically close, and let \( f: \{0,1\}^* \rightarrow \{0,1\}^* \) be a function. Prove that the ensembles \( \{f(X_n)\}_{n \in \mathbb{N}} \) and \( \{f(Y_n)\}_{n \in \mathbb{N}} \) are statistically close.

Exercise 3: Prove that for every \( L \in \mathcal{BPP} \) and every pair of polynomial-time indistinguishable ensembles, \( \{X_n\}_{n \in \mathbb{N}} \) and \( \{Y_n\}_{n \in \mathbb{N}} \), it holds that the function

\[
\Delta_L(n) \overset{\text{def}}{=} |\Pr(X_n \in L) - \Pr(Y_n \in L)|
\]
is negligible in \( n \).
It is tempting to think that the converse holds as well, but we don’t know if it does; note that \( \{X_n\} \) and \( \{Y_n\} \) may be distinguished by a probabilistic algorithm, but not by a deterministic one. In such a case, which language should we define? For example, suppose that \( A \) is a probabilistic polynomial-time algorithm and let
\[
L \overset{\text{def}}{=} \{ x : \Pr(A(x) = 1) \geq \frac{1}{2} \},
\]
then \( L \) is not necessarily in \( \mathbf{BPP} \).

**Exercise 4:** An equivalent formulation of statistical closeness: In the non-computational setting both the above and its converse are true and can be easily proven. Namely, prove that two ensembles, \( \{X_n\}_{n \in \mathbb{N}} \) and \( \{Y_n\}_{n \in \mathbb{N}} \), are statistically close if and only if for every set \( S \subseteq \{0,1\}^* \)
\[
\Delta_S(n) \overset{\text{def}}{=} |\Pr(X_n \in S) - \Pr(Y_n \in S)|
\]
is negligible in \( n \).

**Exercise 5:** An information theoretic analogue of Theorem 3.2.6: prove that if two ensembles are statistically close then also their polynomial-products must be statistically close.

**Guideline:** Show that the statistical difference between the \( m \)-products of two distributions is bounded by \( m \) times the distance between the individual distributions.

**Exercise 6:** statistical closeness implies computational indistinguishability: Prove that if two ensembles are statistically close then they are polynomial-time indistinguishable. (Hint: use the result of the previous exercise, and define for every function \( f : \{0,1\}^* \rightarrow \{0,1\} \) a set \( S_f \overset{\text{def}}{=} \{ x : f(x) = 1 \} \).

**Exercise 7:** computational indistinguishability by circuits - probabilism versus determinism: Let \( \{X_n\}_{n \in \mathbb{N}} \) and \( \{Y_n\}_{n \in \mathbb{N}} \) be two ensembles, and \( C \overset{\text{def}}{=} \{C_n\}_{n \in \mathbb{N}} \) be a family of probabilistic polynomial-size circuits. Prove that there exists a family of (deterministic) polynomial-size circuits, \( D \overset{\text{def}}{=} \{D_n\}_{n \in \mathbb{N}} \), so that for every \( n \)
\[
\Delta_D(n) \geq \Delta_C(n)
\]
where
\[
\Delta_D(n) \overset{\text{def}}{=} |\Pr(D_n(X_n) = 1) - \Pr(D_n(Y_n) = 1)|
\]
\[
\Delta_C(n) \overset{\text{def}}{=} |\Pr(C_n(X_n) = 1) - \Pr(C_n(Y_n) = 1)|
\]

**Exercise 8:** computational indistinguishability by circuits - single sample versus several samples: We say that the ensembles \( X = \{X_n\}_{n \in \mathbb{N}} \) and \( Y = \{Y_n\}_{n \in \mathbb{N}} \) are indistinguishable by polynomial-size circuits if for every family, \( \{C_n\}_{n \in \mathbb{N}} \), of (deterministic) polynomial-size circuits, for every polynomial \( p(\cdot) \) and all sufficiently large \( n \)’s
\[
|\Pr(C_n(X_n) = 1) - \Pr(C_n(Y_n) = 1)| < \frac{1}{p(n)}
\]
CHAPTER 3. PSEUDORANDOM GENERATORS

Prove that $X$ and $Y$ are indistinguishable by polynomial-size circuits if and only if their $m(\cdot)$-products are indistinguishable by polynomial-size circuits, for every polynomial $m(\cdot)$. (Hint: $X$ and $Y$ need not be polynomial-time constructible! Yet, a "good choice" of $x^1, ..., x^k$ and $y^{k+1}, ..., y^m$ may be "hard-wired" into the circuit.)

Exercise 9: computational indistinguishability - circuits vs algorithms:

1. (Easy) Suppose that the ensembles $X = \{X_n\}_{n \in \mathbb{N}}$ and $Y = \{Y_n\}_{n \in \mathbb{N}}$ are indistinguishable by polynomial-size circuits. Prove that they are computationally indistinguishable (by probabilistic polynomial-time algorithms).

2. (Hard) Show that there exist ensembles which are computationally indistinguishable (by probabilistic polynomial-time algorithms), but are distinguishable by polynomial-size circuits.

Guideline (for Part 2): Given any function $f : \{0, 1\}^* \mapsto \{0, 1\}$, construct an ensemble $X = \{X_n\}_{n \in \mathbb{N}}$ such that each $X_n$ has support of size at most 2 and $\Pr(f(X_n) = 1) \leq \Pr(f(U_n) = 1)$, where $U_n$ is uniformly distributed over $\{0, 1\}^n$. Generalize the argument so that given $t$ such functions, $f^1, ..., f^t$, each $X_n$ has support of size at most $t + 1$ and $\Pr(f^i(X_n) = 1) \leq \Pr(f^i(U_n) = 1)$, for each $i = 1, ..., t$. (Extra hint - consider the $t$-dimensional vectors $(f^1(x), ..., f^t(x))$, for each $x \in \{0, 1\}^n$ and think of convex hulls.) A standard diagonalization argument will finish the job. (In case you did not get it, consult [GM97 indist].)

Author's Note: See ECCC, TR96-067, 1996.

Exercise 10: On the general definition of a pseudorandom generator: Let $G$ be a pseudorandom generator (by Definition 3.3.1), and let $\{U_n\}_{n \in \mathbb{N}}$ be polynomial-time indistinguishable from $\{G(U_n)\}_{n \in \mathbb{N}}$. Prove that the probability that $G(U_n)$ has length not equal to $l(n)$ is negligible (in $n$).

Guideline: Consider an algorithm that for some polynomial $p(\cdot)$ proceeds as follows. On input $1^n$ and a string to be tested $\sigma$, the algorithm first samples $G(U_n)$ for $p(n)$ times and records the length of the shortest string found. Next the algorithm outputs 1 if and only if $\sigma$ is longer than the length recorded.

Exercise 11: Consider a modification of Construction 3.3.3, where $s \oplus \sigma = G_1(s_{n-1})$ is used instead of $s \sigma = G_1(s_{n-1})$. Provide a simple proof that the resulting algorithm is also pseudorandom. (Hint: don't modify the proof of Theorem 3.3.4, but rather modify $G_1$ itself.)

Exercise 12: Analogously to Exercise 7 in Chapter 3, refute the following conjecture.

For every pseudorandom generator $G$, the function $G''(s) \overset{\text{def}}{=} G(s) \oplus 0^{l(G(s))-|s|}$ is also a pseudorandom generator.

Guideline: Let $g$ be a pseudorandom generator, and consider $G$ defined on pairs of strings of the same length so that $G(r, s) = (r, g(s))$. 
3.8. MISCELLANEOUS

Exercise 13: Let \( G \) be a pseudorandom generator, and \( h \) be a polynomial-time computable permutation (over strings of the same length). Prove that \( G' \) and \( G'' \) defined by 
\[
G'(s) \overset{\text{def}}{=} h(G(s)) \text{ and } G''(s) \overset{\text{def}}{=} G(h(s))
\]
are both pseudorandom generators.

Exercise 14: Alternative construction of pseudorandom generators with large expansion factor: Let \( G_1 \) be a pseudorandom generator with expansion factor \( l(n) = n + 1 \), and let \( p(\cdot) \) be a polynomial. Define \( G(s) \) to be the result of applying \( G_1 \) iteratively \( p(|s|) \) times on \( s \) (i.e., \( G(s) \overset{\text{def}}{=} G_1^{p(|s|)}(s) \) where \( G_1^{0}(s) \overset{\text{def}}{=} s \) and \( G_1^{i+1} \overset{\text{def}}{=} G_1(G_1^{i}(s)) \)). Prove that \( G \) is a pseudorandom generator. What are the advantages of using Construction 3.3.3?

Exercise 15: Sequential Pseudorandom Generator: A oracle machine is called a sequential observer if its queries constitute a prefix of the natural numbers. Namely, on input \( 1^n \), the sequential observer makes queries \( 1, 2, 3, \ldots \). Consider the following two experiments with a sequential observer having input \( 1^n \):

1. The observer's queries are answered by independent flips of an unbiased coin.
2. The observer's queries are answered as follows. First a random seed, \( s_0 \), of length \( n \) is uniformly chosen. The \( i \)-th query is answered by the leftmost bit of \( G(s_{i-1}) \) and \( s_i \) is set to equal the \( n \) rightmost bits of \( G(s_{i-1}) \), where \( G \) is a fixed pseudorandom generator.

Prove that a probabilistic polynomial-time observer cannot distinguish the two experiments. Namely, the difference between the probability that the observer outputs 1 in the first experiment and the probability that the observer outputs 1 in the second experiment is a negligible function (in \( n \)).

Exercise 16: Pseudorandomness implies unpredictability: Prove that all pseudorandom ensembles are unpredictable (in polynomial-time). (Hint: Given an efficient predictor show how to construct an efficient distinguisher of the pseudorandom ensemble from the uniform one.)

Exercise 17: Unpredictability implies pseudorandomness: Let \( X = \{X_n\}_{n \in \mathbb{N}} \) be an ensemble such that there exists a function \( i: \mathbb{N} \to \mathbb{N} \) so that \( X_n \) ranges over string of length \( l(n) \), and \( l(n) \) can be computed in time \( \text{poly}(n) \). Prove that if \( X \) is unpredictable (in polynomial-time) then it is pseudorandom.

Guideline: The main part of the argument is analogous to the one used in the proof of Theorem 3.3.4. That is, given an efficient distinguisher of \( X \) from the uniform ensemble \( \{U_{n,i}\}_{i \in \mathbb{N}} \), one shows how to construct an efficient predictor. The predictor randomly selects \( k \in \{0, \ldots, l(n) - 1\} \) reads only the first \( k \) bits of the input, and applies \( D \) to the string resulting by augmenting the \( k \)-bit long prefix of the input with \( l(n) - k \) uniformly chosen bits. If \( D \) answers 1 then the predictor outputs the first of these random bits else the predictor outputs the complementary value. The hybrid technique is used to evaluate the performance of the predictor. Extra hint: argument analogous to that of the proof of Theorem 3.4.1 has to be used as well.
CHAPTER 3. PSEUDORANDOM GENERATORS

Exercise 18: Construction of Hashing Families:

1. Consider the set $S_n^m$ of functions mapping $n$-bit long strings into $m$-bit strings as follows. A function $h$ in $S_n^m$ is represented by an $n$-by-$m$ binary matrix $A$, and an $m$-dimensional binary vector $b$. The $n$-dimensional binary vector $x$ is mapped by the function $h$ to the $m$-dimensional binary vector resulting by multiplying $x$ by $A$ and adding the vector $b$ to the resulting vector (i.e., $h(x) = xA + b$). Prove that $S_n^m$ so defined constitutes a hashing family (as defined in Section 3.5).

2. Repeat the above item when the $n$-by-$m$ matrices are restricted to be Toeplitz matrices. An $n$-by-$m$ Toeplitz matrix, $T = \{T_{i,j}\}$, satisfies $T_{i,j} = T_{i+1,j+1}$ for all $i,j$.

Note that binary $n$-by-$m$ Toeplitz matrices can be represented by strings of length $n + m - 1$, whereas representing arbitrary $n$-by-$m$ binary matrices requires strings of length $n \cdot m$.

Exercise 19: Another Hashing Lemma: Let $m, n, S_n^m, b, X_n$ and $\delta$ be as in Lemma 3.5.1. Prove that, for every set $S \subseteq \{0,1\}^n$, and for all but a $2^{-(t - m + \log_2 |S|)} \delta^2$ fraction of the $h$'s in $S_n^m$, it holds that

$$\Pr(h(X_n) \in S) \in (1 \pm \delta) \cdot \frac{|S|}{2^m}$$

(Hint: Follow the proof of Lemma 3.5.1, defining $\zeta(h) = 1$ if $h(x) \in S$ and 0 otherwise.)

Exercise 20: Yet another Hashing Lemma: Let $m, n$, and $S_n^m$ be as above. Let $B \subseteq \{0,1\}^n$ and $S \subseteq \{0,1\}^n$ be sets, and let $b \overset{\text{def}}{=} \log_2 |B|$ and $s \overset{\text{def}}{=} \log_2 |S|$. Prove that, for all but a $\frac{2^m}{|B||S|} \cdot \delta^2$ fraction of the $h$'s in $S_n^m$, it holds that

$$|\{x \in B : h(x) \in S\}| \in (1 \pm \delta) \cdot (|B| \cdot |S|)$$

(Hint: Define a random variable $X_n$ that is uniformly distributed over $B$.)

Exercise 21: Failure of an alternative construction of pseudorandom functions: Consider a construction of a function ensemble where the functions in $F_n$ are defined as follows. For every $s \in \{0,1\}^n$, the function $f_s$ is defined so that

$$f_s(x) \overset{\text{def}}{=} G_{s_1}(\cdots(G_{s_1}(G_{s_1}(x))\cdots)$$

where $s = s_1 \cdots s_n$; and $G_s$ is as in Construction 3.6.4. Namely the roles of $s$ and $t$ in Construction 3.6.4 are switched (i.e., the root is labelled $s$ and the value of $f_s$ on $x$ is obtained by following the path corresponding to the index $s$). Prove that the resulting function ensemble is not necessarily pseudorandom (even if $C$ is a pseudorandom generator).
3.8. MISCELLANEOUS

Guideline: Show, first, that if pseudorandom generators exist then there exists a pseudorandom generator \( G \) satisfying \( G(0^n) = 0^n \).

Exercise 22: Pseudorandom Generators with Direct Access: A direct access pseudorandom generator is a deterministic polynomial-time algorithm, \( G \), for which no probabilistic polynomial-time oracle machine can distinguish the following two cases:

1. New queries of the oracle machine are answered by independent flips of an unbiased coin. (Repeating the same query yields the same answer.)
2. First, a random “seed”, \( s \), of length \( n \) is uniformly chosen. Next, each query, \( q \), is answered by \( G(s, q) \).

The bit \( G(s, i) \) may be thought of as the \( i^{th} \) bit in a bit sequence corresponding to the seed \( s \), where \( i \) is represented in binary. Prove that the existence of (regular) pseudorandom generators implies the existence of pseudorandom generators with direct access. Note that modifying the current definition, so that only unary queries are allowed, yields an alternative definition of a sequential pseudorandom generator (presented in Exercise 15 above). Evaluate the advantage of direct access pseudorandom generators over sequential pseudorandom generators in settings requiring direct access only to bits of a polynomially long pseudorandom sequence.

Exercise 23: other types of pseudorandom functions: Define pseudorandom predicate ensembles so that the random variable \( F_n \) ranges over arbitrary Boolean predicates (i.e., functions in the range of \( F_n \) are defined on all strings and have the form \( f : \{0, 1\}^n \rightarrow \{0, 1\} \)). Assuming the existence of pseudorandom generators, construct efficiently computable ensembles of pseudorandom Boolean functions. Same for ensembles of functions in which each function in the range of \( F_n \) operates on the set of all strings (i.e., has the form \( f : \{0, 1\}^* \rightarrow \{0, 1\}^* \)).

Guideline: Use a modification of Construction 3.6.4 in which the building block is a pseudorandom generator expanding strings of length \( n \) into strings of length \( 3n \).

Exercise 24: An alternative definition of pseudorandom functions: For sake of simplicity this exercise is stated in terms of ensembles of Boolean functions as presented in the previous exercise. We say that a Boolean function ensemble, \( F = \{ F_n \}_{n \in \mathbb{N}} \), is unpredictable if for every probabilistic polynomial-time oracle machine, \( M \), for every polynomial \( p(\cdot) \) and for all sufficiently large \( n \)’s

\[
\Pr(\text{corr}^F_x(M^{F_n}(1^n))) < \frac{1}{2} + \frac{1}{p(n)}
\]

where \( M^{F_n} \) assumes values of the form \( (x, \sigma) \in \{0, 1\}^{n+1} \) so that \( x \) is not a query appearing in the computation \( M^{F_n} \), and \( \text{corr}^F(x, \sigma) \) is defined as the predicate “\( f(x) = \sigma \)”. Intuitively, after getting the value of \( f \) on points of its choice, the machine \( M \)
outputs a new point and tries to guess the value of \( f \) on this point. Assuming that \( F = \{ F_n \}_{n \in \mathbb{N}} \) is efficiently computable, prove that \( F \) is pseudorandom if and only if \( F \) is unpredictable.

**Guideline:** A pseudorandom function ensemble is unpredictable since the uniform function ensemble is unpredictable. For the other direction use ideas analogous to those used in Exercise 16.

**Exercise 25:** A mistaken “alternative” definition of pseudorandom functions: Consider the following definition of unpredictability of function ensembles. The predicting oracle machine, \( M \), is given a uniformly chosen \( x \in \{0, 1\}^n \) as input and should output a guess to \( f(x) \), after querying the oracle \( f \) on polynomially-many other (than \( x \)) points of its choice. We require that for every probabilistic polynomial-time oracle machine, \( M \), that does not query the oracle on its own input, for every polynomial \( p(\cdot) \), and for all sufficiently large \( n \)’s

\[
\Pr(M^{F^*}(U_n) = F_n(U_n)) < \frac{1}{2} + \frac{1}{p(n)}
\]

Show that a pseudorandom function ensemble meets this requirement but that, provided pseudorandom functions ensembles exists, there exists a function ensemble which is unpredictable in the sense defined here although it is not pseudorandom. This exercise contradicts a flawed claim (which appeared in earlier versions of this manuscript). The flaw (which constitutes an answer to the current exercise) was pointed out by Omer Reingold.

**Exercise 26:** An unsuccessful attempt to make the above definition equivalent to pseudorandomness function: Suppose that we strengthen the requirement of the previous exercise by allowing the input, \( x \), to be chosen from any polynomial-time constructable ensemble. Namely, here we say that a function ensemble, \( F = \{ F_n \}_{n \in \mathbb{N}} \), is unpredictable if for every probabilistic polynomial-time oracle machine, \( M \), that does not query the oracle on its own input, for every polynomial-time ensemble \( \{ X_n \}_{n \in \mathbb{N}} \); every polynomial \( p(\cdot) \), and for all sufficiently large \( n \)’s

\[
\Pr(M^{F^*}(X_n) = F_n(X_n)) < \frac{1}{2} + \frac{1}{p(n)}
\]

Again, show that this definition is a necessary but insufficient definition of pseudorandom function ensembles.

**Exercise 27:** Let \( F_\ast \) and \( \text{DES}_{F_\ast}^{i} \) be as in Construction 3.7.6. Prove that, regardless of the choice of the ensemble \( F = \{ F_n \}_{n \in \mathbb{N}} \), the ensemble \( \text{DES}_{F_\ast}^{i} \) is not pseudorandom. Similarly, prove that the ensemble \( \text{DES}_{F_\ast}^{i} \) is not strongly pseudorandom.

**Guideline:** Start by showing that the ensemble \( \text{DES}_{F_\ast}^{i} \) is not pseudorandom.

**Author's Note:** First draft written mainly in 1991.