Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

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.

Organizaton: Basic discussions, defintions 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.

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

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 posses 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".

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

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, $\{x_n\}_{n\in N}$ and $\{y_n\}_{n\in N}$, 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 can not 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

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

$|w|$. 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, $X \stackrel{\text{def}}{=} \{X_n\}_{n\in\mathbb{N}}$ and $Y \stackrel{\text{def}}{=} \{Y_n\}_{n\in\mathbb{N}}$, 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, $X \stackrel{\text{def}}{=} \{X_w\}_{w\in S}$ and $Y \stackrel{\text{def}}{=} \{Y_w\}_{w\in S}$, 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 polynomialy 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 α. Consider the expectation of d taken over each of the two ensembles. Namely, let $d_1(n) = \mathsf{E}(d(X_n))$ and $d_2(n) = \mathsf{E}(d(Y_n))$. Then, X and Y are said to be indistinguishable by D if the difference (function) $\Delta(n) \stackrel{\text{def}}{=} |d_1(n) - d_2(n)|$ 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,

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

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 $|\mathsf{Pr}(\omega(X_n) < \frac{n}{2}) - \mathsf{Pr}(\omega(Y_n) < \frac{n}{2})|$ is negligible (in n), where $\omega(\alpha)$ denotes the number of 1's in the string α. 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 $X \stackrel{\text{def}}{=} \{X_n\}_{n\in\mathbb{N}}$ and $Y \stackrel{\text{def}}{=} \{Y_n\}_{n\in\mathbb{N}}$, *statistically close* if their statistical difference is negligible, where the *statistical difference* (also known as *variation distance*) of X and Y is defined as the function

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

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, $U \stackrel{\text{def}}{=} \{U_n\}_{n\in\mathbb{N}}$, 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 $f : \{0,1\}^* \mapsto \{0,1\}$ 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

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

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.

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

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

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

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 $s_1, ..., s_{2^{n/2}} \in \{0,1\}^n$, so that for every circuit C_n of size $2^{n/8}$ it holds that

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

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

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

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

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, $X \stackrel{\text{def}}{=} \{X_n\}_{n\in\mathbb{N}}$, 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 $X \stackrel{\text{def}}{=} \{X_n\}_{n\in\mathbb{N}}$ and $Y \stackrel{\text{def}}{=} \{Y_n\}_{n\in\mathbb{N}}$, 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 $Z \stackrel{\text{def}}{=} \{Z_n\}_{n\in\mathbb{N}}$ and every polynomial $m(\cdot)$ define the $m(\cdot)$-product of Z as the ensemble $\{(Z_n^{(1)}, ..., Z_n^{(m(n))})\}_{n\in\mathbb{N}}$, 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 X 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

$$\Delta(n) \stackrel{\text{def}}{=} |\mathsf{Pr}\left(D(X_n^{(1)}, ..., X_n^{(m)})=1\right) - \mathsf{Pr}\left(D(Y_n^{(1)}, ..., Y_n^{(m)})=1\right)| > \frac{1}{p(n)}$$

where $m \stackrel{\text{def}}{=} m(n)$, 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).

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

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,

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

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

By our hypothesis, algorithm D can distinguish the extreme hybrids (i.e., H_n^0 and H_n^m). 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 α (supposedly in the range of either X_n or Y_n), algorithm D' proceeds as follows. Algorithm D', first selects k uniformly in the set $\{0, 1, ..., m-1\}$. Using the efficient sampling algorithm for the ensemble X, algorithm D' generates k independent samples of X_n. These samples are denoted $x^1, ..., x^k$. Likewise, using the efficient sampling algorithm for the ensemble Y, algorithm D' generates $m-k-1$ independent samples of Y_n, denoted $y^{k+2}, ..., y^m$. Finally, algorithm D' invokes algorithm D and halts with output $D(x^1, ..., x^k, \alpha, y^{k+2}, ..., y^m)$.

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

Claim 3.2.6.1:

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

and

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

Proof: By construction of algorithm D', we have

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

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

Claim 3.2.6.2:

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

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

$$|\mathsf{Pr}(D'(X_n)=1) - \mathsf{Pr}(D'(Y_n)=1)|$$

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

$$= \frac{1}{m} \cdot |\sum_{k=0}^{m-1} \mathsf{Pr}(D(H_n^{k+1})=1) - \mathsf{Pr}(D(H_n^k)=1)|$$
$$= \frac{1}{m} \cdot |\mathsf{Pr}(D(H_n^m)=1) - \mathsf{Pr}(D(H_n^0)=1)|$$
$$= \frac{\Delta(n)}{m}$$

The last equality follows by observing that $H_n^m = X_n^{(1)}, ..., X_n^{(m)}$ and $H_n^0 = Y_n^{(1)}, ..., Y_n^{(m)}$, 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. ■

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 3.2.6, 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.

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

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 *standard uniform ensemble*. Yet, it will be convenient to call *uniform* also ensembles of the form $\{U_{l(n)}\}_{n\in\mathbb{N}}$, where l is a function on natural numbers.

Definition 3.2.7 (pseudorandom ensembles): *Let $U \stackrel{\text{def}}{=} \{U_{l(n)}\}_{n\in\mathbb{N}}$ be a uniform ensemble, and $X \stackrel{\text{def}}{=} \{X_n\}_{n\in\mathbb{N}}$ 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 $|U_m| = m$). In fact, with high probability $|X_n|$ equals $l(n)$).

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:*

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

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 $\{U_{l(n)}\}_{n\in\mathbb{N}}$ 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 α outputs 1 if and only if $|\alpha| > n$ will distinguish $G(U_n)$ from $U_{l(n)}$ (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 $U_{l(n)}$, as follows

$$\begin{aligned}
\sum_x |\mathsf{Pr}(U_{l(n)}=x) - \mathsf{Pr}(G(U_n)=x)| &\geq \sum_{x\notin\{G(s):s\in\{0,1\}^n\}} |\mathsf{Pr}(U_{l(n)}=x) - \mathsf{Pr}(G(U_n)=x)| \\
&= (2^{l(n)} - 2^n)\cdot 2^{-l(n)} \\
&\geq \frac{1}{2}
\end{aligned}$$

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 $\{U_{l(n)}\}_{n\in\mathbb{N}}$. For simplicity, we consider in the sequel, only pseudorandom generators G satisfying $|G(x)| = l(|x|)$ 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* $l : \mathbb{N} \mapsto \mathbb{N}$ *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).

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

The above definition says little about the expansion factor $l:\mathbb{N}\mapsto\mathbb{N}$. We merely know that for every n it holds that $l(n) \geq n+1$, that $l(n) \leq \mathrm{poly}(n)$, 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, $l_1:\mathbb{N}\mapsto\mathbb{N}$ and $l_2:\mathbb{N}\mapsto\mathbb{N}$, that can be computed in $\mathrm{poly}(n)$-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 $l_1(n) \stackrel{\mathrm{def}}{=} n+1$ to construct, for every polynomial $p(\cdot)$, a pseudorandom generator with expansion factor $p(n)$. Note that a pseudorandom generator with expansion factor $l_1(n) \stackrel{\mathrm{def}}{=} n+1$ 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 $s_0 \stackrel{\mathrm{def}}{=} s$, the bit σ_i 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 σs, where σ 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.

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

Proof: The proof is by a "reducibility argument" . Suppose, to the contradiction, that G is not a pseudorandom generator. It follows that the ensembles $\{G(U_n)\}_{n\in\mathbb{N}}$ and $\{U_{p(n)}\}_{n\in\mathbb{N}}$ are not polynomial-time indistinguishable. We will show that it follows that the ensembles $\{G_1(U_n)\}_{n\in\mathbb{N}}$ and $\{U_{n+1}\}_{n\in\mathbb{N}}$ are not polynomial-time indistinguishable, in contradiction to the hypothesis that G_1 is a pseudorandom generator with expansion factor $l_1(n) = n+1$. The implication is proven, using the hybrid technique.

For every k, $0 \le k \le p(n)$, we define a hybrid $H^k_{p(n)}$ as follows. First we define, for every k, a function $g^k_n : \{0,1\}^n \mapsto \{0,1\}^k$ by letting $g^0_n(x) \stackrel{\rm def}{=} \lambda$ (the empty string) and $g^{k+1}_n(x) = \sigma g^k_n(y)$, where σ is the first bit of $G_1(x)$ and y is the n-bit long suffix of $G_1(x)$ (i.e., $\sigma y = G_1(x)$). Namely, for every $k \le p(|x|)$, the string $g^k_n(x)$ equals the k-bit long prefix of $G(x)$. Define the random variable $H^k_{p(n)}$ resulting by concatenating a uniformly chosen k-bit long string and the random variable $g^{p(n)-k}(U_n)$. Namely

$$H^k_{p(n)} \stackrel{\rm def}{=} U^{(1)}_k g^{p(n)-k}(U^{(2)}_n)$$

where $U^{(1)}_k$ and $U^{(2)}_n$ are independent random variables (the first uniformly distributed over $\{0,1\}^k$ and the second uniformly distributed over $\{0,1\}^n$). Intuitively, the hybrid $H^k_{p(n)}$ consists of the k-bit long prefix of $U_{p(n)}$ and the $(p(n)-k)$-bit long suffix of $G(X_n)$, where X_n is obtained from U_n by applying G_1 for k times each time to the n-bit long suffix of the previous result. However, the later way of looking at the hybrids is less convenient for our purposes.

At this point it is clear that $H^0_{p(n)}$ equals $G(U_n)$, whereas $H^{p(n)}_{p(n)}$ equals $U_{p(n)}$. It follows that if an algorithm D can distinguish the extreme hybrids then D can also distinguish two neighbouring hybrids, since the total number of hybrids is polynomial in n and a non-negligible gap between the extreme hybrids translates into a non-negligible gap between some neighbouring hybrids. The punch-line is that, using the structure of neighbouring hybrids, algorithm D can be easily modified to distinguish the ensembles $\{G_1(U_n)\}_{n\in\mathbb{N}}$ and $\{U_{n+1}\}_{n\in\mathbb{N}}$. Details follow.

The core of the argument is the way in which the distinguishability of neighbouring hybrids relates to the distinguishability of $G(U_n)$ from U_{n+1}. As stated, this relation stems from the structure of neighbouring hybrids. Let us, thus, take a closer look at the hybrids $H^k_{p(n)}$ and $H^{k+1}_{p(n)}$, for some $0 \le k \le p(n)-1$. To this end, define a function $f^m : \{0,1\}^{n+1} \mapsto \{0,1\}^m$ by letting $f^0(z) \stackrel{\rm def}{=} \lambda$ and $f^{m+1}(z) \stackrel{\rm def}{=} \sigma g^m(y)$, where $z = \sigma y$ with $\sigma \in \{0,1\}$.

Claim 3.3.4.1:

1. $H^k_{p(n)} = U^{(1)}_k f^{p(n)-k}(X_{n+1})$, where $X_{n+1} = G_1(U^{(2)}_n)$.

2. $H^{k+1}_{p(n)} = U^{(1)}_k f^{p(n)-k}(Y_{n+1})$, where $Y_{n+1} = U^{(3)}_{n+1}$.

Proof:

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

1. By definition of the functions g^m and f^m, we have $g^m(x) = f^m(G_1(x))$. Using the definition of the hybrid $H^k_{p(n)}$, it follows that

$$H^k_{p(n)} = U^{(1)}_k g^{p(n)-k}(U^{(2)}_n) = U^{(1)}_k f^{p(n)-k}(G_1(U^{(2)}_n))$$

2. On the other hand, by definition $f^{m+1}(\sigma y) = \sigma g^m(y)$, and using the definition of the hybrid $H^{k+1}_{p(n)}$, we get

$$H^{k+1}_{p(n)} = U^{(1)}_{k+1} g^{p(n)-k-1}(U^{(2)}_n) = U^{(1)}_k f^{p(n)-k}(U^{(3)}_{n+1})$$

□

Hence distinguishing $G_1(U_n)$ from U_{n+1} is reduced to distinguishing the neighbouring hybrids (i.e. $H^k_{p(n)}$ and $H^{k+1}_{p(n)}$), by applying $f^{p(n)-k}$ to the input, padding the outcome (in front of) by a uniformly chosen string of length k, and applying the hybrid-distinguisher to the resulting string. Further details follow.

We assume, to the contrary of the theorem, that G is not a pseudorandom generators. Suppose that D is a probabilistic polynomial-time algorithm so that for some polynomial $q(\cdot)$ and for infinitely many n's it holds that

$$\Delta(n) \stackrel{\text{def}}{=} |\mathsf{Pr}(D(G(U_n)\!=\!1) - \mathsf{Pr}(D(U_{p(n)})\!=\!1)| > \frac{1}{q(n)}$$

We derive a contradiction by constructing a probabilistic polynomial-time algorithm, D', that distinguishes $G_1(U_n)$ from U_{n+1}.

Algorithm D' uses algorithm D as a subroutine. On input $\alpha \in \{0,1\}^{n+1}$, algorithm D' operates as follows. First, D' selects an integer k uniformly in the set $\{0, 1, ..., p(n) - 1\}$, next D' selects β uniformly in $\{0,1\}^k$, and finally D' halts with output $D(\beta f^{p(n)-k}(\alpha))$, where $f^{p(n)-k}$ is as defined above.

Clearly, D' can be implemented in probabilistic polynomial-time (in particular $f^{p(n)-k}$ is computed by applying G_1 polynomially many times). It is left to analyze the performance of D' on each of the distributions $G_1(U_n)$ and U_{n+1}.

Claim 3.3.4.2:

$$\mathsf{Pr}(D'(G(U_n))\!=\!1) = \frac{1}{p(n)} \sum_{k=0}^{p(n)-1} \mathsf{Pr}(D(H^k_{p(n)})\!=\!1)$$

and

$$\mathsf{Pr}(D'(U_{n+1})\!=\!1) = \frac{1}{p(n)} \sum_{k=0}^{p(n)-1} \mathsf{Pr}(D(H^{k+1}_{p(n)})\!=\!1)$$

Proof: By construction of D' we get, for every $\alpha \in \{0,1\}^{n+1}$,

$$\mathsf{Pr}(D'(\alpha)\!=\!1) = \frac{1}{p(n)} \sum_{k=0}^{p(n)-1} \mathsf{Pr}(D(U_k f^{p(n)-k}(\alpha))\!=\!1)$$

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

Using Claim 3.3.4.1, our claim follows. □

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

$$\begin{aligned} |\Pr(D'(G_1(U_n))=1) - \Pr(D'(U_{n+1})=1)| &= \frac{1}{p(n)} \cdot |\sum_{k=0}^{p(n)-1} d^k(n) - d^{k+1}(n)| \\ &= \frac{|d^0(n) - d^{p(n)}(n)|}{p(n)} \\ &= \frac{\Delta(n)}{p(n)} \end{aligned}$$

Recall that by our (contradiction) hypothesis $\Delta(n) > \frac{1}{q(n)}$, 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/expanders 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

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

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*

$$\mathsf{Pr}(A(1^n, X_n) = \mathrm{next}_A(1^n, X_n)) < \frac{1}{2} + \frac{1}{p(n)}$$

where $\mathrm{next}_A(x)$ *returns the* $i+1^{\mathrm{st}}$ *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* $f:\{0,1\}^* \mapsto \{0,1\}^*$ *defined by letting* $f(x,y) \stackrel{\mathrm{def}}{=} G(x)$*, 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 $f(U_{2n})$ with success probability at least $\frac{1}{\mathrm{poly}(n)}$. We will construct a probabilistic polynomial-time algorithm, D, that distinguishes U_{2n} 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 α 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,

$$\mathsf{Pr}(f(A(f(U_{2n}))) = f(U_{2n})) > \frac{1}{p(n)}$$

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

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

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

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 flavour, 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 $G(s) \stackrel{\text{def}}{=} f(s)b(s)$, 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 $f(U_n)\overline{b}(U_n)$, where $\overline{b}(x)$ is the complement bit of $b(x)$ (i.e., $\overline{b}(x) \stackrel{\text{def}}{=} \{0,1\}-b(x)$). 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.

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

We assume, on the contrary, that there exists a probabilistic polynomial-time algorithm D and a polynomial $p(\cdot)$ so that for infinitely many n's

$$|\mathsf{Pr}(D(G(U_n))=1) - \mathsf{Pr}(D(U_{n+1})=1)| > \frac{1}{p(n)}$$

Assume, without loss of generality, that for infinitely many n's it holds that

$$\Delta(n) \stackrel{\text{def}}{=} (\mathsf{Pr}(D(G(U_n))=1) - \mathsf{Pr}(D(U_{n+1})=1)) > \frac{1}{p(n)}$$

We construct a probabilistic polynomial-time algorithm, A, for predicting $b(x)$ from $f(x)$. Algorithm A uses the algorithm D as a subroutine. On input y (equals $f(x)$ for some x), algorithm A proceeds as follows. First, A selects uniformly $\sigma \in \{0,1\}$. Next, A applies D to $y\sigma$. Algorithm A halts outputting σ if $D(y\sigma) = 1$ and outputs the complement of σ, denoted $\overline{\sigma}$, otherwise.

Clearly, A works in polynomial-time. It is left to evaluate the success probability of algorithm A. We evaluate the success probability of A by considering two complementary events. The event we consider is whether or not "on input x algorithm A selects σ so that $\sigma = b(x)$".

Claim 3.4.1.1:

$$\begin{aligned}
\mathsf{Pr}(A(f(U_n))=b(U_n)\,|\,\sigma=b(U_n)) &= \mathsf{Pr}(D(f(U_n)b(U_n))=1) \\
\mathsf{Pr}(A(f(U_n))=b(U_n)\,|\,\sigma\neq b(U_n)) &= 1 - \mathsf{Pr}(D(f(U_n)\overline{b}(U_n))=1)
\end{aligned}$$

where $\overline{b}(x)=\{0,1\} - b(x)$.

Proof: By construction of A,

$$\begin{aligned}
\mathsf{Pr}(A(f(U_n))=b(U_n)\,|\,\sigma=b(U_n)) &= \mathsf{Pr}(D(f(U_n)\sigma)=1\,|\,\sigma=b(U_n)) \\
&= \mathsf{Pr}(D(f(U_n)b(U_n))=1\,|\,\sigma=b(U_n)) \\
&= \mathsf{Pr}(D(f(U_n)b(U_n))=1)
\end{aligned}$$

where the last equality follows since D's behavior is independent of the value of σ. Likewise,

$$\begin{aligned}
\mathsf{Pr}(A(f(U_n))=b(U_n)\,|\,\sigma\neq b(U_n)) &= \mathsf{Pr}(D(f(U_n)\sigma)=0\,|\,\sigma=\overline{b}(U_n)) \\
&= \mathsf{Pr}(D(f(U_n)\overline{b}(U_n))=0\,|\,\sigma=\overline{b}(U_n)) \\
&= 1 - \mathsf{Pr}(D(f(U_n)\overline{b}(U_n))=1)
\end{aligned}$$

The claim follows. $\square$

Claim 3.4.1.2:

$$\begin{aligned}
\mathsf{Pr}(D(f(U_n)b(U_n))=1) &= \mathsf{Pr}(D(G(U_n))=1) \\
\mathsf{Pr}(D(f(U_n)\overline{b}(U_n))=1) &= 2\cdot\mathsf{Pr}(D(U_{n+1})=1) - \mathsf{Pr}(D(f(U_n)b(U_n))=1)
\end{aligned}$$

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

Proof: By definition of G, we have $G(U_n) = f(U_n)b(U_n)$, and the first claim follows. To justify the second claim, we use the fact that f is a permutation over $\{0,1\}^n$, and hence $f(U_n)$ is uniformly distributed over $\{0,1\}^n$. It follows that U_{n+1} can be written as $f(U_n)U_1$. We get

$$\mathsf{Pr}(D(U_{n+1})=1) = \frac{\mathsf{Pr}(D(f(U_n)b(U_n))=1) + \mathsf{Pr}(D(f(U_n)\overline{b}(U_n))=1)}{2}$$

and the claim follows. □

Combining Claims 3.4.1.1 and 3.4.1.2, we get

$$\begin{aligned}
\mathsf{Pr}(A(f(U_n))=b(U_n)) &= \mathsf{Pr}(\sigma=b(U_n)) \cdot \mathsf{Pr}(A(f(U_n))=b(U_n) \mid \sigma=b(U_n)) \\
&\quad + \mathsf{Pr}(\sigma\neq b(U_n)) \cdot \mathsf{Pr}(A(f(U_n))=b(U_n) \mid \sigma\neq b(U_n)) \\
&= \frac{1}{2} \cdot \left(\mathsf{Pr}(D(f(U_n)b(U_n))=1) + 1 - \mathsf{Pr}(D(f(U_n)\overline{b}(U_n))=1)\right) \\
&= \frac{1}{2} + (\mathsf{Pr}(D(G(U_n))=1) - \mathsf{Pr}(D(U_{n+1})=1)) \\
&= \frac{1}{2} + \Delta(n)
\end{aligned}$$

Since $\Delta(n) > \frac{1}{p(n)}$ for infinitely many n's, we derive a contradiction and the theorem follows. ■

3.4.2 Construction based on Collections of Permutations

We now combine the underlying ideas of Construction 3.3.3 (of Subsection 3.3.3) and Theorem 3.4.1 (above) to present a construction of pseudorandom generators based on collections of one-way permutations. Let (I, D, F) be a triplet of algorithms defining a collection of one-way permutations (see Section 2.4.2). Recall that $I(1^n, r)$ denotes the output of algorithm I on input 1^n and coin tosses r. Likewise, $D(i, s)$ denotes the output of algorithm D on input i and coin tosses s. The reader may assume, for simplicity, that $|r|=|s|=n$. Actually, this assumption can be justified in general - see Exercise 13. However, in many applications it is more natural to assume that $|r|=|s|=q(n)$ for some fixed polynomial $q(\cdot)$. We remind the reader that Theorem 2.5.2 applies also to collections of one-way permutations.

Construction 3.4.2 *Let (I, D, F) be a triplet of algorithms defining a strong collection of one-way permutations, and let B be a hard-core predicate for this collection. Let $p(\cdot)$ be an arbitrary polynomial. Define $G(r, s) = \sigma_1 \cdots \sigma_{p(n)}$, where $i \stackrel{\rm def}{=} I(1^n, r)$, $s_0 \stackrel{\rm def}{=} D(i, s)$, and for every $1 \le j \le p(|s|)$ it holds that $\sigma_j = B(s_{j-1})$ and $s_j = f_i(s_{j-1})$.*

On seed (r, s), algorithm G first uses r to determine a permutation f_i over D_i (i.e., $i \leftarrow I(1^n, r)$). Secondly, algorithm G uses s to determine a "starting point", s_0, in D_i. For simplicity, let us shorthand f_i by f. The essential part of algorithm G is the repeated application of the function f to the starting point s_0 and the extraction of a hard-core

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

predicate for each resulting element. Namely, algorithm G computes a sequence of elements $s_1, ..., s_{p(n)}$, where $s_j = f(s_{j-1})$ for every j (i.e., $s_j = f^{(j)}(s_0)$, where $f^{(j)}$ denotes j successive applications of the function f). Finally, algorithm G outputs the string $\sigma_1 \cdots \sigma_{p(n)}$, where $\sigma_j = B(s_{j-1})$. Note that σ_j is easily computed from s_{j-1} but is a "hard to approximate" from $s_j = f(s_{j-1})$. The pseudorandomness property of algorithm G depends on the fact that G does not output the intermediate s_j's. (In the sequel, we will see that outputting the last element, namely $s_{p(n)}$, does not hurt the pseudorandomness property.) The expansion property of algorithm G depends on the choice of the polynomial $p(\cdot)$. Namely, the polynomial $p(\cdot)$ should be larger than the polynomial $2q(\cdot)$ (where $2q(n)$ equals the total length of r and s corresponding to $I(1^n)$).

Theorem 3.4.3 *Let (I, D, F), B, $p(\cdot)$, and G be as in Construction 3.4.2 (above), so that $p(n) > 2q(n)$ for all n's. Suppose that for every i in the range of algorithm I, the random variable $D(i)$ is uniformly distbuted over the set D_i. Then G is a pseudorandom generator.*

Theorem 3.4.3 is an immediate corollary of the following proposition.

Proposition 3.4.4 *Let n and t be integers. For every i in the range of $I(1^n)$ and every x in D_i, define $G_{i,t}(x) = \sigma_1 \cdots \sigma_t$, where $s_0 \stackrel{\text{def}}{=} x$, $s_j = f_i^{(j)}(x)$ ($f^{(j)}$ denotes j successive applications of the function f) and $\sigma_j = B(s_{j-1})$, for every $1 \le j \le t$. Let (I, D, F) and B be as in Theorem 3.4.3 (above), I_n be a random variable representing $I(1^n)$, and $X_n = D(I_n)$ be a random variable depending on I_n. Then, for every polynomial $p(\cdot)$, the ensembles $\{(I_n, G_{I_n,p(n)}(X_n), f_{I_n}^{(p(n))}(X_n))\}_{n\in\mathbb{N}}$ and $\{(I_n, U_{p(n)}, f_{I_n}^{(p(n))}(X_n))\}_{n\in\mathbb{N}}$ are polynomial-time indistinguishable.*

Hence, the distinguishing algorithm gets, in addition to the $p(n)$-bit long sequence to be examined, also the index i chosen by G (in the first step of G's computation) and the last s_j (i.e., $s_{p(n)}$) computed by G. Even with this extra information it is infeasible to distinguish $G_{I_n,p(n)}(X_n) = G(1^n U_{2q(n)})$ from $U_{p(n)}$.

Proof Outline: The proof follows the proofs of Theorems 3.3.4 and 3.4.1 (of Subsection 3.3.3 and the current subsection, resp.). First, the statement is proven for $p(n) = 1$ (for all n's). This part is very similar to the proof of Theorem 3.4.1. Secondly, observe that the random variable X_n has distribution identical to the random variable $f_{I_n}(X_n)$, even conditioned on $I_n = i$ (of every i). Finally, assuming the validity of the case $p(\cdot) = 1$, the statement is proven for every polynomial $p(\cdot)$. This part is analogous to the proof of Theorem 3.3.4: one has to construct hybrids so that the k^{th} hybrid starts with an element i in the support of I_n, followed by k random bits, and ends with $G_{i,p(n)-k}(X_n)$ and $f_i^{p(n)-k}(X_n)$, where $X_n = D(i)$. The reader should be able to complete the argument. ■

Proposition 3.4.4 and Theorem 3.4.3 remain valid even if one relaxes the condition concerning the distribution of $D(i)$, and only requires that $D(i)$ is statistically close (as a function in $|i|$) to the uniform distribution over D_i.

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

3.4.3 Practical Constructions

As an immediate application of Construction 3.4.2, we derive pseudorandom generators based on either of the following assumptions

- *The Intractability of the Discrete Logarithm Problem*: The genertor is based on the fact that it is hard to predict, given a prime P, a primitive element G in the multiplicative group mod P, and an element Y of the group, whether there exists $0 \leq x \leq P/2$ so that $Y \equiv G^x \bmod P$. In other words, this bit constitues a hard-core for the DLP collection (of Subsection 2.4.3).

- *The difficulty of inverting RSA*: The genertor is based on the fact that the least significant bit constitues a hard-core for the RSA collection.

- *The Intractability of Factoring Blum Integers*: The genertor is based on the fact that the least significant bit constitues a hard-core for the Rabin collection, when viewed as a collection of permutations over the quadratic residues of Blum integers (see Subsection 2.4.3).

We ellaborate on the last example since it offers the most efficient implementation and yet is secure under a widely believed intractability assumption. The generator uses its seed in order to generate a composite number, N, which is the product of two relatively large primes.

*** PROVIDE DETAILS ABOVE. MORE EFFICIEN HEURISTIC BELOW...

3.5 * Construction based on One-Way Functions

It is known that one-way functions exist if and only if pseudorandom generators exist. However, the known construction which transforms arbitrary one-way functions into pseudorandom generators is impractical. Furthermore, the proof that this construction indeed yields pseudorandom generators is very complex and unsuitable for a book of the current nature. Instead, we refrain to present some of the ideas underlying this construction.

3.5.1 Using 1-1 One-Way Functions

Recall that if f is a 1-1 length-preserving one-way function and b is a corresponding hard-core predicate then $G(s) \stackrel{\text{def}}{=} f(s)b(s)$ constitutes a pseudorandom generator. Let us relax the condition imposed on f and assume that f is a 1-1 one-way function (but is not necessarily length preserving). Without loss of generality, we may assume that there exists a polynomial $p(\cdot)$ so that $|f(x)| = p(|x|)$ for all x's. In case f is not length preserving, it follows that $p(n) > n$. At first glance, one may think that we only benefit in such a case since f by itself has an expanding property. The impression is misleading since the expanded strings may not "look random". In particular, it may be the case that the first bit of $f(x)$ is

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

zero for all x's. More generally, $f(U_n)$ may be easy to distinguish from $U_{p(n)}$ (otherwise f itself constitutes a pseudorandom generator). Hence, in the general case, we need to get rid of the expansion property of f since it is not accompanied by a "pseudorandom" property. In general, we need to shrink $f(U_n)$ back to length $\leq n$ so that the shrunk result induces uniform distribution. The question is how to *efficiently* carry on this process (i.e., of shrinking $f(x)$ back to length $|x|$, so that the shrunk $f(U_n)$ induces a uniform distribution on $\{0,1\}^n$).

Suppose that there exists an efficiently computable function h so that $f_h(x) \stackrel{\text{def}}{=} h(f(x))$ is length preserving and 1-1. In such a case we can let $G(s) \stackrel{\text{def}}{=} h(f(s))b(s)$, where b is a hard-core predicate for f, and get a pseudorandom generator. The pseudorandomness of G follows from the observation that if b is a hard-core for f it is also a hard-core for f_h (since an algorithm guessing $b(x)$ from $h(f(x))$ can be easily modified so that it guesses $b(x)$ from $f(x)$, by applying h first). The problem is that we "know nothing about the structure" of f and hence are not guaranteed that h as above does exist. An important observation is that a uniformly selected *hashing* function will have approximately the desired properties. Hence, hashing functions play a central role in the construction, and consequently we need to discuss these functions first.

Hashing Functions

The following terminology relating to hashing functions is merely an ad-hoc terminology (which is not a standard one). Let S_n^m be a set of strings representing functions mapping n-bit strings to m-bit strings. In the sequel we freely associate the strings in S_n^m with the functions that they represent. Let H_n^m be a random variable uniformly distributed over the set S_n^m. We call S_n^m a **hashing family** if it satisfies the following three conditions:

1. S_n^m is *a pairwise independent family of mappings*: for every $x \neq y \in \{0,1\}^n$, the random variables $H_n^m(x)$ and $H_n^m(y)$ are independent and uniformly distributed in $\{0,1\}^m$.

2. S_n^m has *succinct representation*: $S_n^m = \{0,1\}^{\text{poly}(n \cdot m)}$.

3. S_n^m can be *efficiently evaluated*: there exists a polynomial-time algorithm that, on input a representation of a function, h (in S_n^m), and a string $x \in \{0,1\}^n$, returns $h(x)$.

A widely used hashing family is the set of affine transformations mapping n-dimensional binary vectors to m-dimensional ones (i.e., transformations affected by multiplying the n-dimensional vector by an n-by-m binary matrix and adding an m-dimensional vector to the result). A hashing family with more succinct representation is obtained by considering only the transformations affected by Toeplitz matrices (i.e., matrices which are invariant along the diagonals). For further details see Exercise 18. Following is a lemma, concerning hashing functions, that is central to our application (as well as to many applications of hashing functions in complexity theory). Loosely speaking, the lemma asserts that most

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

h's in a hashing family have $h(X_n)$ distributed almost uniformly, provided X_n does not assign too much probability mass to any single string.

Lemma 3.5.1 *Let $m < n$ be integers, S_n^m be a hashing family, and b and δ be two reals so that $b \leq n$ and $\delta \geq 2^{-\frac{b-m}{2}}$. Suppose that X_n is a random variable distributed over $\{0,1\}^n$ so that for every x it holds that $\mathsf{Pr}(X_n = x) \leq 2^{-b}$. Then, for every $\alpha \in \{0,1\}^m$, and for all but a $2^{-(b-m)}\delta^{-2}$ fraction of the h's in S_n^m, it holds that*

$$\mathsf{Pr}(h(X_n) = \alpha) \in (1 \pm \delta) \cdot 2^{-m}$$

A function h not satisfying $\mathsf{Pr}(h(X_n) = \alpha) \in (1 \pm \delta) \cdot 2^{-m}$ is called *bad* (for α and the random variable X_n). Averaging on all h's we have $\mathsf{Pr}(h(X_n) = \alpha)$ equal 2^{-m}. Hence the lemma bounds the fraction of h's which deviate from the average value. Typically we shall use $\delta \stackrel{\rm def}{=} 2^{-\frac{b-m}{3}} \ll 1$ (making the deviation from average equal the fraction of bad h's). Another useful choice is $\delta > 1$ (which yields an even smaller fraction of bad h's, yet badness has only a "lower bound interpretation", i.e. $\mathsf{Pr}(h(X_n) = \alpha) \leq (1 + \delta) \cdot 2^{-m}$).

Proof: Fix an arbitrary random variable X_n, satisfying the conditions of the lemma, and an arbitrary $\alpha \in \{0,1\}^m$. Denote $w_x \stackrel{\rm def}{=} \mathsf{Pr}(X_n = x)$. For every h we have

$$\mathsf{Pr}(h(X_n) = \alpha) = \sum_x w_x \zeta_x(h)$$

where $\zeta_x(h)$ equal 1 if $h(x) = \alpha$ and 0 otherwise. Hence, we are interested in the probability, taken over all possible choices of h, that $|2^{-m} - \sum_x w_x \zeta_x(h)| > \delta 2^{-m}$. Looking at the ζ_x's as random variables defined over the random variable H_n^m, it is left to show that

$$\mathsf{Pr}\left(|2^{-m} - \sum_x w_x \zeta_x| > \delta \cdot 2^{-m}\right) > \frac{2^{-(b-m)}}{\delta^2}$$

This is proven by applying Chebyshev's Inequality, using the fact that the ζ_x's are pairwise independent, and that ζ_x equals 1 with probability 2^{-m} (and 0 otherwise). (We also take advantage on the fact that $w_x \leq 2^{-b}$.) Namely,

$$\begin{aligned} \mathsf{Pr}\left(|2^{-m} - \sum_x w_x \zeta_x| > \delta \cdot 2^{-m}\right) &\leq \frac{\mathsf{V}\left(\sum_x w_x \zeta_x\right)}{(\delta \cdot 2^{-m})^2} \\ &< \frac{\sum_x 2^{-m} w_x^2}{\delta^2 \cdot 2^{-2m}} \\ &\leq \frac{2^{-m} 2^{-b}}{\delta^2 \cdot 2^{-2m}} \end{aligned}$$

The lemma follows. ∎

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

Constructing "Almost" Pseudorandom Generators

Using any 1-1 one-way function and any hashing family, we can take a major step towards constructing a pseudorandom generator.

Construction 3.5.2 *Let $f:\{0,1\}^* \mapsto \{0,1\}^*$ be a function satisfying $|f(x)| = p(|x|)$ for some polynomial $p(\cdot)$ and all x's. For any integer function $l:\mathbb{N}\mapsto\mathbb{N}$, let $g:\{0,1\}^*\mapsto\{0,1\}^*$ be a function satisfying $|g(x)| = l(|x|)+1$, and $S_{p(n)}^{n-l(n)}$ be a hashing family. For every $x \in \{0,1\}^n$ and $h \in S_{p(n)}^{n-l(n)}$, define*

$$G(x,h) \stackrel{\text{def}}{=} (h(f(x)), h, g(x))$$

Clearly, $|G(x,h)| = (|x|-l(|x|)) + |h| + (l(|x|)+1) = |x| + |h| + 1$.

Proposition 3.5.3 *Let f, l, g and G be as above. Suppose that f is 1-to-1 and g is a hard-core function of f. Then, for every probabilistic polynomial-time algorithm A, every polynomial $p(\cdot)$, and all sufficiently large n's*

$$|\mathsf{Pr}(A(G(U_n,U_k))=1) - \mathsf{Pr}(A(U_{n+k+1})=1)| < 2^{-\frac{l(n)}{3}} + \frac{1}{p(n)}$$

where k is the length of the representation of the hashing functions in $S_{p(n)}^{n-l(n)}$.

The proposition can be extended to the case in which the function f is polynomial-to-1 (instead of 1-to-1). Specifically, let f satisfy $|f^{-1}f(x)| < q(|x|)$, for some polynomial $q(\cdot)$ and all sufficiently long x's. The modified proposition asserts that for every probabilistic polynomial-time algorithm A, every polynomial $p(\cdot)$, and all sufficiently large n's

$$|\mathsf{Pr}(A(G(U_n,U_k))=1) - \mathsf{Pr}(A(U_{n+k+1})=1)| < 2^{-\frac{l(n)-\log_2 q(n)}{3}} + \frac{1}{p(n)}$$

where k is as above.

In particular, the above proposition holds for functions $l(\cdot)$ of the form $l(n) \stackrel{\text{def}}{=} c\log_2 n$, where $c > 0$ is a constant. For such functions l, every one-way function (can be easily modified into a function which) has a hard-core g as required in the proposition's hypothesis (see Subsection 2.5.3). Hence, we get very close to constructing a pseudorandom generator.

Proof Sketch: We first note that

$$\begin{aligned} G(U_nU_k) &= (H_{p(n)}^{n-l(n)}(f(U_n)), H_{p(n)}^{n-l(n)}, g(U_n)) \\ U_{n+k+1} &= (U_{n-l(n)}, H_{p(n)}^{n-l(n)}, U_{l(n)+1}) \end{aligned}$$

We consider the hybrid $(H_{p(n)}^{n-l(n)}(f(U_n)), H_{p(n)}^{n-l(n)}, U_{l(n)+1})$. The proposition is a direct consequence of the following two claims.

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

Claim 3.5.3.1: The ensembles

$$\{(H_{p(n)}^{n-l(n)}(f(U_n)), H_{p(n)}^{n-l(n)}, g(U_n))\}_{n\in\mathbb{N}}$$

and

$$\{(H_{p(n)}^{n-l(n)}(f(U_n)), H_{p(n)}^{n-l(n)}, U_{l(n)+1})\}_{n\in\mathbb{N}}$$

are polynomial-time indistinguishable.
Proof Idea: Use a "reducibility argument". If the claim does not hold then contradiction to the hypothesis that g is a hard-core of f is derived. $\Box$
Claim 3.5.3.2: The statistical difference between the random variables

$$(H_{p(n)}^{n-l(n)}(f(U_n)), H_{p(n)}^{n-l(n)}, U_{l(n)+1})$$

and

$$(U_{n-l(n)}, H_{p(n)}^{n-l(n)}, U_{l(n)+1})$$

is bounded by $2^{-l(n)/3}$.
Proof Idea: Use the hypothesis that $S_{p(n)}^{n-l(n)}$ is a hashing family, and apply Lemma 3.5.1. $\Box$
Since the statistical difference is a bound on the ability of algorithms to distinguish, the proposition follows. $\blacksquare$

Applying Proposition 3.5.3

Once the proposition is proven we consider the possibilities of applying it in order to construct pseudorandom generators. We stress that applying Proposition 3.5.3, with length function $l(\cdot)$, requires having a hard-core function g for f with $|g(x)| = l(|x|) + 1$. By Theorem 2.5.4 (of Subsection 2.5.3) such hard-core functions exist practically for all one-way functions, provided that $l(\cdot)$ is logarithmic (actually, Theorem 2.5.4 asserts that such hard-cores exist for a modification of any one-way function which preserves its 1-1 property). Hence, combining Theorem 2.5.4 and Proposition 3.5.3, and using a logarithmic length function, we get very close to constructing a pseudorandom generator. In particular, for every polynomial $p(\cdot)$, using $l(n) \stackrel{\text{def}}{=} 3\log_2 p(n)$, we can construct a deterministic polynomial-time algorithm expanding n-bit long seeds into $(n+1)$-bit long strings so that no polynomial-time algorithm can distinguish the output strings from uniformly chosen ones, with probability greater than $\frac{1}{p(n)}$ (except for finitely many n's). Yet, this does not imply that the output is pseudorandom (i.e., that the distinguishing gap is smaller than any polynomial fraction). A final trick is needed (since we cannot use $l(\cdot)$ bigger than any logarithmic function). In the sequel we present two alternative ways for obtaining a pseudorandom generator from the above construction.

The first alternative is to use Construction 3.3.3 (of Subsection 3.3.3) in order to increase the expansion factor of the above algorithms. In particular, for every integer k, we construct a deterministic polynomial-time algorithm expanding n-bit long seeds into n^3-bit long strings so that no polynomial-time algorithm can distinguish the output strings

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

from uniformly chosen ones, with probability greater than $\frac{1}{n^k}$ (except for finitely many n's). Denote these algorithms by $G_1, G_2, ...$, and construct a pseudorandom generator G by letting

$$G(s) \stackrel{\rm def}{=} G_1(s_1) \oplus G_2(s_2) \cdots \oplus G_{k(|s|)}(s_{k(|s|)})$$

where $\oplus$ denotes bit-by-bit exclusive-or of strings, $s_1 s_2 \cdots s_{k(|s|)} = s$, $|s_i| = \frac{|s|}{k(|s|)} \pm 1$, and $k(n) \stackrel{\rm def}{=} \sqrt[3]{n}$. Clearly, $|G(s)| \approx k(|s|) \cdot (\frac{|s|}{k(|s|)})^3 = |s|^2$. The pseudorandomness of G follows by a "reducibility argument". (The choice of the function k is rather arbitrary, and any unbounded function $k(\cdot)$ satisfying $k(n) < n^{2/3}$ will do.)

The second alternative is to apply Construction 3.5.2 to the function $\overline{f}$ defined by

$$\overline{f}(x_1, ..., x_n) \stackrel{\rm def}{=} f(x_1) \cdots f(x_n)$$

where $|x_1| = \cdots = |x_n| = n$. The benefit in applying Construction 3.5.2 to the function $\overline{f}$ is that we can use $l(n^2) \stackrel{\rm def}{=} n-1$, and hence Proposition 3.5.3 yields that G is a pseudorandom generator. All that is left is to show that $\overline{f}$ has a hard core function which maps n^2-bit strings into n-bit strings. Assuming that b is a hard-core predicate of the function f, we can construct such a hard-core function for $\overline{f}$. Specifically,

Construction 3.5.4 *Let* $f: \{0,1\}^* \mapsto \{0,1\}^*$ *and* $b: \{0,1\}^* \mapsto \{0,1\}$. *Define*

$$\begin{aligned} \overline{f}(x_1, ..., x_n) &\stackrel{\rm def}{=} f(x_1) \cdots f(x_n) \\ \overline{g}(x_1, ..., x_n) &\stackrel{\rm def}{=} b(x_1) \cdots b(x_n) \end{aligned}$$

where $|x_1| = \cdots = |x_n| = n$.

Proposition 3.5.5 *Let f and b be as above. If b is a hard-core predicate of f then $\overline{g}$ is a hard-core function of $\overline{f}$.*

Proof Idea: Use the hybrid technique. The $i^{\rm th}$ hybrid is

$$\left(\overline{f}(U_n^{(1)}, ..., U_n^{(n)}), b(U_n^{(1)}), ..., b(U_n^{(i)}), U_1^{(i+1)}, ..., U_1^{(n)}\right)$$

Use a reducibility argument (as in Theorem 3.4.1 of Subsection 3.4.1) to convert a distinguishing algorithm into one predicting b from f. ■

Using either of the above alternatives, we get

Theorem 3.5.6 *If there exist 1-1 one-way functions then pseudorandom generators exist as well.*

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

The entire argument can be extended to the case in which the function f is polynomial-to-1 (instead of 1-to-1). Specifically, let f satisfy $|f^{-1}f(x)| < q(|x|)$, for some polynomial $q(\cdot)$ and all sufficiently long x's. Then if f is one-way then (either of the above alternatives yields that) pseudorandom generators exists. Proving the statement using the first alternative is quite straightforward given the discussion proceeding Proposition 3.5.3. In proving the statement using the second alternative apply Construction 3.5.2 to the function $\overline{f}$ with $l(n^2) \stackrel{\rm def}{=} n \cdot (1 + \log_2 q(n)) - 1$. This requires showing that $\overline{f}$ has a hard core function which maps n^2-bit strings into $n(1+\log_2 q(n))$-bit strings. Assuming that g is a hard-core function of the function f, with $|g(x)| = 1 + \log_2 q(|x|)$, we can construct such a hard-core function for $\overline{f}$. Specifically,

$$\overline{g}(x_1, ..., x_n) \stackrel{\rm def}{=} g(x_1) \cdots g(x_n)$$

where $|x_1| = \cdots = |x_n| = n$.

3.5.2 Using Regular One-Way Functions

The validity of Proposition 3.5.3 relies heavily on the fact that if f is 1-1 then $f(U_n)$ maintains the "entropy" of U_n in a strong sense (i.e., $\mathsf{Pr}(f(U_n) = \alpha) \leq 2^{-n}$ for every α). In this case, it was possible to shrink $f(U_n)$ and get almost uniform distribution over $\{0,1\}^{n-l(n)}$. As stressed above, the condition may be relaxed to requiring that f is polynomial-to-1 (instead of 1-to-1). In such a case only logarithmic loss of "entropy" occurs, and such a loss can be compensated by an appropriate increase in the range of the hard-core function. We stress that hard-core functions of logarithmic length (i.e., satisfying $|g(x)| = O(\log |x|)$) can be constructed for any one-way function. However, in general, the function f may not be polynomial-to-1 and in particular it can map exponentially many images to the same range element. If this is the case then applying f to U_n yields a great loss in "entropy", which cannot be compensated using the above methods. For example, if $f(x, y) \stackrel{\rm def}{=} f'(x)0^{|y|}$, for $|x| = |y|$, then $\mathsf{Pr}(f(U_n) = \alpha) \geq 2^{-\frac{|\alpha|}{2}}$ for some α's. In this case, achieving uniform distribution from $f(U_n)$ requires shrinking it to length $\approx n/2$. In general, we cannot compensate for these lost bits since f may not have a hard-core with such huge range (i.e., a hard-core g satisfying $|g(\alpha)| = \frac{|\alpha|}{2}$). Hence, in this case, the above methods fail for constructing an algorithm that expands its input into a longer output. A new idea is needed, and indeed presented below.

The idea is that, in case f maps different preimages into the same image y, we can augment y by the index of the preimage, in the set $f^{-1}(y)$, *without damaging the hardness-to-invert of* f. Namely, we define $F(x) \stackrel{\rm def}{=} f(x) \cdot {\rm idx}_f(x)$, where ${\rm idx}_f(x)$ denotes the index (say by lexicographic order) of x in the set $\{x' : f(x') = f(x)\}$. We claim that inverting F is not substantially easier than inverting f. This claim can be proven by a "reducibility argument". Given an algorithm for inverting F we can invert f as follows. On input y (supposedly in the range of $f(U_n)$), we first select m uniformly in $\{1, ..., n\}$, next select i uniformly in $\{1, ..., 2^m\}$, and finally try to invert F on (y, i). When analyzing this algorithm, consider the case $i = \lceil \log_2 |f^{-1}(y)| \rceil$.

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

The function F suggested above does preserve the hardness-to-invert of f. The problem is that it does not preserve the easy-to-compute property of f. In particular, for general f it is not clear how to compute $\mathrm{idx}_f(x)$ (i.e., the best we can say is that this task can be performed in polynomial *space*). Again, hashing functions come to the rescue. Suppose, for example that f is 2^m-to-1 on strings of length n. Then, we can set $\mathrm{idx}_f(x) = (H_n^m, H_n^m(x))$, obtaining "probabilistic indexing" of the set of preimages. We stress that applying the above trick requires having a good estimate for the size of the set of preimages (of a given image). That is, given x it should be easy to compute $|f^{-1}f(x)|$. A simple case where such an estimate can be handy is the case of regular functions.

Definition 3.5.7 (Regular functions): *A function $f:\{0,1\}^* \mapsto \{0,1\}^*$ is called* regular *if there exists an integer function $m:\mathbb{N}\mapsto\mathbb{N}$ so that for all sufficiently long $x \in \{0,1\}^*$ it holds*

$$|\{y : f(x)=f(y) \wedge |x|=|y|\}| = 2^{m(|x|)}$$

For simplicity, the reader may further assume that there exists an algorithm that on input n computes $m(n)$ in $\mathrm{poly}(n)$-time. As we shall see, in the end of this subsection, one can do without this assumption. For sake of simplicity (of notation), we assume in the sequel that if $f(x)=f(y)$ then $|x|=|y|$.

Construction 3.5.8 *Let $f:\{0,1\}^* \mapsto \{0,1\}^*$ be a regular function with $m(|x|) = \log_2 |f^{-1}f(x)|$ for some integer function $m(\cdot)$. Let $l:\mathbb{N}\mapsto\mathbb{N}$ be an integer function, and $S_n^{m(n)-l(n)}$ be a hashing family. For every $x \in \{0,1\}^n$ and $h \in S_n^{m(n)-l(n)}$, define*

$$F(x,h) \stackrel{\mathrm{def}}{=} (f(x), h(x), h)$$

If f can be computed in polynomial-time and $m(n)$ can be computed from n in $\mathrm{poly}(n)$-time, then F can be computed in polynomial-time. We now show that if f is a regular one-way function, then F is "hard to invert". Furthermore, if $l(\cdot)$ is logarithmic then F is "almost 1-1".

Proposition 3.5.9 *Let f, m, l and F be as above. Suppose that there exists an algorithm that on input n computes $m(n)$ in* $\mathrm{poly}(n)$*-time. Then,*

1. F is "almost" 1-1:

$$\Pr\left(|F^{-1}F(U_n, H_n^{m(n)-l(n)})| > 2^{l(n)+1}\right) < 2^{-\frac{l(n)}{2}}$$

(Recall that H_n^k denotes a random variable uniformly distributed over S_n^k.)

2. F "preserves" the one-wayness of f:
If f is strongly (resp. weakly) one-way then so is F.

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

Proof Sketch: Part (1) is proven by applying Lemma 3.5.1, using the hypothesis that $S_n^{m(n)-l(n)}$ is a hashing family. Part (2) is proven using a "reducibility argument" . Assuming, to the contradiction, that there exists an efficient algorithm A that inverts F with unallowable success probability, we construct an efficient algorithm A' that inverts f with unallowable success probability (reaching contradiction). For sake of concreteness, we consider the case in which f is strongly one-way, and assume to the contradiction that algorithm A inverts F on $F(U_n, H_n^{m(n)-l(n)})$ with success probability $\epsilon(n)$, so that $\epsilon(n) > \frac{1}{\mathrm{poly}(n)}$ for infinitely many n's. Following is a description of A'.

On input y (supposedly in the range of $f(U_n)$), algorithm A' repeats the following experiment for $\mathrm{poly}(\frac{n}{\epsilon(n)})$ many times. Algorithm A' selects uniformly $h \in S_n^{m(n)-l(n)}$ and $\alpha \in \{0,1\}^{m(n)-l(n)}$, and initiates A on input (y, α, h). Algorithm A' sets x to be the n-bit long prefix of $A(y, \alpha, h)$, and outputs x if $y = f(x)$. Otherwise, algorithm A' continues to the next experiment.

Clearly, algorithm A' runs in polynomial-time, provided that $\epsilon(n) > \frac{1}{\mathrm{poly}(n)}$. We now evaluate the success probability of A'. For every possible input, y, to algorithm A', we consider a random variable X_n uniformly distributed in $f^{-1}(y)$. Let $\delta(y)$ denote the success probability of algorithm A on input $(y, H_n^k(X_n), H_n^k)$, where $n \stackrel{\mathrm{def}}{=} |y|$ and $k \stackrel{\mathrm{def}}{=} m(n) - l(n)$. Clearly, $\mathsf{E}(\delta(f(U_n))) = \epsilon(n)$, and $\mathsf{Pr}(\delta(f(U_n)) > \frac{\epsilon(n)}{2}) > \frac{\epsilon(n)}{2}$ follows. We fix an arbitrary $y \in \{0,1\}^n$ so that $\delta(y) > \frac{\epsilon(n)}{2}$. We prove the following technical claim.

Claim 3.5.9.1: Let n, k and X_n be as above. Suppose that B is a set of pairs, and

$$\delta \stackrel{\mathrm{def}}{=} \mathsf{Pr}((H_n^k(X_n), H_n^k) \in B)$$

Then,

$$\mathsf{Pr}((U_k, H_n^k) \in B) > \frac{\delta^4}{2^8 \cdot k}$$

Using this claim, it follows that the probability that A' inverts f on y in a single iteration is at least $(\frac{\delta(y)}{4})^4 \cdot \frac{1}{k}$. We reach a contradiction (to the one-wayness of f), and the proposition follows. All that is left is to prove Claim 3.5.9.1. The proof, given below, is rather technical.
■

We stress that the fact that $m(n)$ can be computed from n does not play an essential role in the reducibility argument (as it is possible to try all possible values of $m(n)$).

Claim 3.5.9.1 is of interest for its own sake. However, its proof provides no significant insights and may be skipped without significant damage (especially by readers that are more interested in cryptography than in "probabilistic analysis").

Proof of Claim 3.5.9.1: We first use Lemma 3.5.1 to show that only a "tiny" fraction of the hashing functions in S_n^k can map "large" probability mass into "small" subsets. Once this is done, the claim is proven by dismissing those few bad functions and relating the two probabilities, appearing in the statement of the claim, conditioned on the function not being bad. Details follow.

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

We begin by bounding the fraction of the hashing functions that map "large" probability mass into "small" subsets. We say that a function $h \in S_n^k$ is (T,Δ)-*expanding* if there exists a set $R \subset \{0,1\}^k$ of cardinality $\Delta \cdot 2^k$ so that $\Pr(h(X_n) \in R)) \geq (T+1)\cdot\Delta$. In other words, h maps to some set of density Δ a probability mass $T+1$ times the density of the set. Our first goal is to prove that at most $\frac{\delta}{4}$ of the h's are $(\frac{32k}{\delta^2}, \frac{\delta^3}{64k})$-expanding. In other words, only $\frac{\delta}{4}$ of the function map to some set of density $\frac{\delta^3}{64k}$ a probability mass of more than $\frac{\delta}{2}$.

We start with a related question. We say that $\alpha \in \{0,1\}^k$ is t-*overweighted* by the function h if $\Pr(h(X_n)=\alpha)) \geq (t+1)\cdot 2^{-k}$. A function $h \in S_n^k$ is called (t,ρ)-*overweighting* if there exists a set $R \subset \{0,1\}^k$ of cardinality $\rho 2^k$ so that each $\alpha \in R$ is t-overweighted by h. (Clearly, if h is (t,ρ)-overweighting then it is also (t,ρ)-expanding, but the converse is not necessarily true.) We first show that at most a $\frac{1}{t^2\rho}$ fraction of the h's are (t,ρ)-overweighting. The proof is given in the rest of this paragraph. Recall that $\Pr(X_n = x) \leq 2^{-k}$, for every x. Using Lemma 3.5.1, it follows that each $\alpha \in \{0,1\}^k$ is t-overweighted by at most a t^{-2} fraction of the h's. Assuming, to the contradiction, that more than a $\frac{1}{t^2\rho}$ fraction of the h's are (t,ρ)-overweighting, we construct a bipartite graph by connecting each of these h's with the α's that it t-overweights. Contradiction follows by observing that there exists an α which is connected to more than $\frac{\frac{|S_n^k|}{t^2\rho}\cdot\rho 2^k}{2^k} = \frac{1}{t^2}\cdot|S_n^k|$ of the h's.

We now relate the expansion and overweighting properties. Specifically, if h is (T,Δ)-expanding then there exists an integer $i \in \{1,...,k\}$ so that h is $(T\cdot 2^{i-1}, \frac{\Delta}{k\cdot 2^i})$-overweighting. Hence, at most a

$$\sum_{i=1}^{k} \frac{1}{(T\cdot 2^{i-1})^2 \cdot \frac{\Delta}{k\cdot 2^i}} < \frac{4k}{T^2\cdot\Delta}$$

fraction of the h's can be (T,Δ)-expanding. It follows that at most $\frac{\delta}{4}$ of the h's are $(\frac{32k}{\delta^2}, \frac{\delta^3}{64k})$-expanding.

We call h *honest* if it is **not** $(\frac{32k}{\delta^2}, \frac{\delta^3}{64k})$-expanding. Hence, if h is honest and $\Pr(h(X_n) \in R) \geq \frac{\delta}{2}$ then R has density at least $\frac{\delta^3}{64k}$. Concentrating on the honest h's, we now evaluate the probability that (α,h) hits B, when α is uniformly chosen. We call h *good* if $\Pr((h(X_n),h) \in B) \geq \frac{\delta}{2}$. Clearly, the probability that H_n^k is good is at least $\frac{\delta}{2}$, and the probability H_n^k is both good and honest is at least $\frac{\delta}{4}$. Denote by G the set of these h's (i.e., h's which are both good and honest). Clearly, for every $h \in G$ we have $\Pr((h(X_n),h) \in B) \geq \frac{\delta}{2}$ (since h is good) and $\Pr((U_k,h) \in B) \geq \frac{\delta^3}{64k}$ (since h is honest). Using $\Pr(H_n^k \in G) \geq \frac{\delta}{4}$, the claim follows. $\square$

Applying Proposition 3.5.9

It is possible to apply Construction 3.5.2 to the function resulting from Construction 3.5.8, and the statement of Proposition 3.5.3 still holds with minor modifications. Specifically, Construction 3.5.2 is applied with $l(\cdot)$ twice the function (i.e., the $l(\cdot)$) used in Construction 3.5.8, and the bound in Proposition 3.5.3 is $3\cdot 2^{-\frac{l(n)}{6}}$ (instead of $2^{-\frac{l(n)}{3}}$). The argument leading to Theorem 3.5.6, remains valid as well. Furthermore, we may even waive the

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

requirement that $m(n)$ can be computed (since we can construct functions F_m for every possible value of $m(n)$). Finally, we note that the entire argument holds even if the definition of regular functions is relaxed as follows.

Definition 3.5.10 (Regular functions - revised definition): *A function $f:\{0,1\}^* \mapsto \{0,1\}^*$ is called* regular *if there exists an integer function $m':\mathbb{N} \mapsto \mathbb{N}$ and a polynomial $q(\cdot)$ so that for all sufficiently long $x \in \{0,1\}^*$ it holds*

$$2^{m'(|x|)} \leq |\{y : f(x)=f(y)\}| \leq q(|x|) \cdot 2^{m'(|x|)}$$

When using these (relaxed) regular functions in Construction 3.5.8, set $m(n) \stackrel{\text{def}}{=} m'(n)$. The resulting function F will have a slightly weaker "almost" 1-1 property. Namely,

$$\mathsf{Pr}\left(|F^{-1}F(U_n, H_n^{m(n)-l(n)})| > q(n) \cdot 2^{l(n)+1}\right) < 2^{-\frac{l(n)}{2}}$$

The application of Construction 3.5.2 will be modified accordingly. We get

Theorem 3.5.11 *If there exist regular one-way functions then pseudorandom generators exist as well.*

3.5.3 Going beyond Regular One-Way Functions

The proof of Proposition 3.5.9 relies heavily on the fact that the one-way function f is regular (at least in the weak sense). Alternatively, Construction 3.5.8 needs to be modified so that different hashing families are associated to different $x \in \{0,1\}^n$. Furthermore, the argument leading to Theorem 3.5.6 cannot be repeated unless it is easy to compute the cardinality of set $f^{-1}(f(x))$ given x. Note that this time we cannot construct functions F_m for every possible value of $\lceil \log_2 |f^{-1}(y)| \rceil$ since none of the functions may satisfy the statement of Proposition 3.5.9. Again, a new idea is needed.

A key observation is that although the value of $\log_2 |f^{-1}(f(x))|$ may vary for different $x \in \{0,1\}^n$, the value $m(n) \stackrel{\text{def}}{=} \mathsf{E}(\log_2 |f^{-1}(f(U_n))|)$ is unique. Furthermore, the function $\overline{f}$ defined by

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

where $|x_1| = |x_{n^2}| = n$, has the property that all but a negligible fraction of the domain reside in preimage sets with logarithm of cardinality not deviating too much from the expected value. Specifically, let $\overline{m}(n^3) \stackrel{\text{def}}{=} \mathsf{E}(\log_2 |\overline{f}^{-1}(\overline{f}(U_{n^3}))|)$. Clearly, $\overline{m}(n^3) = n^2 \cdot m(n)$. Using Chernoff Bound, we get

$$\mathsf{Pr}\left(\text{abs}\left(\overline{m}(n^3) - \log_2 |\overline{f}^{-1}(\overline{f}(U_{n^3}))|\right) > n^2\right) < 2^{-n}$$

Suppose we apply Construction 3.5.8 to $\overline{f}$ setting $l(n^3) \stackrel{\text{def}}{=} n^2$. Denote the resulting function by $\overline{F}$. Suppose we apply Construction 3.5.2 to $\overline{F}$ setting this time $l(n^3) \stackrel{\text{def}}{=} 2n^2 - 1$.

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

Using the ideas presented in the proofs of Propositions 3.5.3 and 3.5.9, one can show that if the n^3-bit to $l(n^3)+1$-bit function used in Construction 3.5.2 is a hard-core of $\overline{F}$ then the resulting algorithm constitutes a pseudorandom generator. Yet, we are left with the problem of constructing a huge hard-core function, $\overline{G}$, for the function $\overline{F}$. Specifically, $|\overline{G}(x)|$ has to equal $2|x|^{\frac{2}{3}}$, for all x's. A natural idea is to define $\overline{G}$ analogously to the way $\overline{g}$ is defined in Construction 3.5.4. Unfortunately, we do not know how to prove the validity of this construction (when applied to $\overline{F}$), and a much more complicated construction is required. This construction does use all the above ideas in conjunction with additional ideas not presented here. The proof of validity is even more complex, and is not suitable for a book of the current nature. We thus conclude this section by merely stating the result obtained.

Theorem 3.5.12 *If there exist one-way functions then pseudorandom generators exist as well.*

3.6 Pseudorandom Functions

Pseudorandom generators enable to generate, exchange and share a large number of pseudorandom values at the cost of a much smaller number of random bits. Specifically, poly(n) pseudorandom bits can be generated, exchanged and shared at the cost of n (uniformly chosen bits). Since any efficient application uses only a polynomial number of random values, providing access to polynomially many pseudorandom entries seems sufficient. However, the above conclusion is too hasty, since it assumes implicitly that these entries (i.e., the addresses to be accessed) are fixed beforehand. In some natural applications, one may need to access addresses which are determined "dynamically" by the application. For example, one may want to assign random values to (poly(n) many) n-bit long strings, produced throughout the application, so that these values can be retrieved at latter time. Using pseudorandom generators the above task can be achieved at the cost of generating n random bits and storing poly(n) many values. The challenge, met in the sequel, is to achieve the above task at the cost of generating and storying only n random bits. The key to the solution is the notion of pseudorandom functions. In this section we define pseudorandom functions and show how to efficiently implement them. The implementation uses as a building block any pseudorandom generator.

3.6.1 Definitions

Loosely speaking, pseudorandom functions are functions which cannot be distinguished from truly random functions by any efficient procedure which can get the value of the function at arguments of its choice. Hence, the distinguishing procedure may query the function being examined at various points, depending possibly on previous answers obtained, and yet can not tell whether the answers were supplied by a function taken from the pseudorandom ensemble (of functions) or from the uniform ensemble (of function). Hence, to formalize the

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

notion of pseudorandom functions we need to consider ensembles of functions. For sake of concreteness we consider in the sequel ensembles of length preserving functions. Extensions are discussed in Exercise 23.

Definition 3.6.1 (function ensembles): *A* function ensemble *is a sequence $F = \{F_n\}_{n\in\mathbb{N}}$ of random variables, so that the random variable F_n assumes values in the set of functions mapping n-bit long strings to n-bit long strings. The* uniform function ensemble, *denoted $H = \{H_n\}_{n\in\mathbb{N}}$, has H_n uniformly distributed over the set of functions mapping n-bit long strings to n-bit long strings.*

To formalize the notion of pseudorandom functions we use (probabilistic polynomial-time) *oracle machines*. We stress that our use of the term oracle machine is almost identical to the standard one. One deviation is that the oracle machines we consider have a length preserving function as oracle rather than a Boolean function (as is standard in most cases in the literature). Furthermore, we assume that on input 1^n the oracle machine only makes queries of length n. These conventions are not really essential (they merely simplify the exposition a little).

Definition 3.6.2 (pseudorandom function ensembles): *A function ensemble, $F = \{F_n\}_{n\in\mathbb{N}}$, is called* pseudorandom *if for every probabilistic polynomial-time oracle machine M, every polynomial $p(\cdot)$ and all sufficiently large n's*

$$|\mathsf{Pr}(M^{F_n}(1^n)=1) - \mathsf{Pr}(M^{H_n}(1^n)=1)| < \frac{1}{p(n)}$$

where $H = \{H_n\}_{n\in\mathbb{N}}$ is the uniform function ensemble.

Using techniques similar to those presented in the proof of Proposition 3.2.3 (of Subsection 3.2.2), one can demonstrate the existence of pseudorandom function ensembles which are not statistically close to the uniform one. However, to be of practical use, we need require that the pseudorandom functions can be efficiently computed.

Definition 3.6.3 (efficiently computable function ensembles): *A function ensemble, $F = \{F_n\}_{n\in\mathbb{N}}$, is called* efficiently computable *if the following two conditions hold*

1. (efficient indexing): *There exists a probabilistic polynomial time algorithm, I, and a mapping from strings to functions, ϕ, so that $\phi(I(1^n))$ and F_n are identically distributed.*
   We denote by f_i the $\{0,1\}^n \mapsto \{0,1\}^n$ function assigned to i (i.e., $f_i \stackrel{\text{def}}{=} \phi(i)$).

2. (efficient evaluation): *There exists a probabilistic polynomial time algorithm, V, so that $V(i,x) = f_i(x)$.*

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

In particular, functions in an efficiently computable function ensemble have relatively succinct representation (i.e., of polynomial rather than exponential length). It follows that efficiently computable function ensembles may have only exponentially many functions (out of the double-exponentially many possible functions).

Another point worthy of stressing is that pseudorandom functions may (if being efficiently computable) be efficiently evaluated at given points, *provided that the function description is give as well.* However, if the function (or its description) is *not* known (and it is only known that it is chosen from the pseudorandom ensemble) then the value of the function at a point cannot be approximated (even in a very liberal sense and) even if the values of the function at other points is also given.

In the rest of this book we consider only efficiently computable pseudorandom functions. Hence, in the sequel we sometimes shorthand such ensembles by calling them pseudorandom functions.

3.6.2 Construction

Using any pseudorandom generator, we construct a (efficiently computable) pseudorandom function (ensemble).

Construction 3.6.4 *Let G be a deterministic algorithm expanding inputs of length n into strings of length $2n$. We denote by $G_0(s)$ the $|s|$-bit long prefix of $G(s)$, and by $G_1(s)$ the $|s|$-bit long suffix of $G(s)$ (i.e., $G(s) = G_0(s)G_1(s)$). For every $s \in \{0,1\}^n$, we define a function $f_s : \{0,1\}^n \mapsto \{0,1\}^n$ so that for every $\sigma_1, ..., \sigma_n \in \{0,1\}$*

$$f_s(\sigma_1\sigma_2\cdots\sigma_n) \stackrel{\text{def}}{=} G_{\sigma_n}(\cdots(G_{\sigma_2}(G_{\sigma_1}(s))\cdots)$$

Let F_n be a random variable defined by uniformly selecting $s \in \{0,1\}^n$ and setting $F_n = f_s$. Finally, let $F = \{F_n\}_{n\in\mathbb{N}}$ be our function ensemble.

Pictorially, the function f_s is defined by n-step walks down a full binary tree of depth n having labels on the vertices. The root of the tree, hereafter referred to as the level 0 vertex of the tree, is labelled by the string s. If an internal node is labelled r then its left child is labelled $G_0(r)$ whereas its right child is labelled $G_1(r)$. The value of $f_s(x)$ is the string residing in the leaf reachable from the root by a path corresponding to string x, when the root is labelled by s. The random variable F_n is assigned labelled trees corresponding to all possible 2^n labellings of the root, with uniform probability distribution.

A function, operating on n-bit strings, in the ensemble constructed above can be specified by n bits. Hence, selecting, exchanging and storing such a function can be implemented at the cost of selecting, exchanging and storing a single n-bit string.

Theorem 3.6.5 *Let G and F be as in Construction 3.6.4, and suppose that G is a pseudorandom generator. Then F is an efficiently computable ensemble of pseudorandom functions.*

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

Proof: Clearly, the ensemble F is efficiently computable. To prove that F is pseudorandom we use the hybrid technique. The $k^{\rm th}$ hybrid will be assigned functions which result by uniformly selecting labels for the vertices of the $k^{\rm th}$ (highest) level of the tree and computing the labels of lower levels as in Construction 3.6.4. The 0-hybrid will correspond to the random variable F_n (since a uniformly chosen label is assigned to the root), whereas the n-hybrid will correspond to the uniform random variable H_n (since a uniformly chosen label is assigned to each leaf). It will be shown that an efficient oracle machine distinguishing neighbouring hybrids can be transformed into an algorithm that distinguishes polynomially many samples of $G(U_n)$ from polynomially many samples of U_{2n}. Using Theorem 3.2.6 (of Subsection 3.2.3), we derive a contradiction to the hypothesis (that G is a pseudorandom generator). Details follows.

For every k, $0\le k\le n$, we define a hybrid distribution H_n^k (assigned as values functions $f:\{0,1\}^n\mapsto\{0,1\}^n$) as follows. For every $s_1,s_2,...,s_{2^k}\in\{0,1\}^n$, we define a function $f_{s_1,...,s_{2^k}}:\{0,1\}^n\mapsto\{0,1\}^n$ so that

$$f_{s_1,...,s_{2^k}}(\sigma_1\sigma_2\cdots\sigma_n)\stackrel{\rm def}{=}G_{\sigma_n}(\cdots(G_{\sigma_{k+2}}(G_{\sigma_{k+1}}(s_{{\rm idx}(\sigma_k\cdots\sigma_1)}))\cdots)$$

where ${\rm idx}(\alpha)$ is index of α in the standard lexicographic order of strings of length $|\alpha|$. (In the sequel we take the liberty of associating the integer ${\rm idx}(\alpha)$ with the string α.) Namely, $f_{s_{0^k},...,s_{1^k}}(x)$ is computed by first using the k-bit long prefix of x to determine one of the s_j's, and next using the $(n-k)$-bit long suffix of x to determine which of the functions G_0 and G_1 to apply at each remaining stage. The random variable H_n^k is uniformly distributed over the above $(2^n)^{2^k}$ possible functions. Namely,

$$H_n^k\stackrel{\rm def}{=}f_{U_n^{(1)},...,U_n^{(2^k)}}$$

where $U_n^{(j)}$'s are independent random variables each uniformly distributed over $\{0,1\}^n$.

At this point it is clear that H_n^0 is identical to F_n, whereas H_n^n is identical to H_n. Again, as usual in the hybrid technique, ability to distinguish the extreme hybrids yields ability to distinguish a pair of neighbouring hybrids. This ability is further transformed (as sketched above) so that contradiction to the pseudorandomness of G is reached. Further details follow.

We assume, in contradiction to the theorem, that the function ensemble F is not pseudorandom. It follows that there exists a probabilistic polynomial-time oracle machine, M, and a polynomial $p(\cdot)$ so that for infinitely many n's

$$\Delta(n)\stackrel{\rm def}{=}|{\sf Pr}(M^{F_n}(1^n)=1)-{\sf Pr}(M^{H_n}(1^n)=1)|>\frac{1}{p(n)}$$

Let $t(\cdot)$ be a polynomial bounding the running time of $M(1^n)$ (such a polynomial exists since M is polynomial-time). It follows that, on input 1^n, the oracle machine M makes at most $t(n)$ queries (since the number of queries is clearly bounded by the running time).

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

Using the machine M, we construct an algorithm D that distinguishes the $t(\cdot)$-product of the ensemble $\{G(U_n)\}_{n\in\mathbb{N}}$ from the $t(\cdot)$-product of the ensemble $\{U_{2n}\}_{n\in\mathbb{N}}$ as follows.

On input $\alpha_1,...,\alpha_t \in \{0,1\}^{2n}$ (with $t = t(n)$), algorithm D proceeds as follows. First, D selects uniformly $k \in \{0,1,...,n-1\}$. This random choice, hereafter called the *checkpoint*, and is the only random choice made by D itself. Next, algorithm D invokes the oracle machine M (on input 1^n) and answers M's queries as follows. The first query of machine M, denoted q_1, is answered by

$$G_{\sigma_n}(\cdots(G_{\sigma_{k+2}}(P_{\sigma_{k+1}}(\alpha_1)))\cdots)$$

where $q_1 = \sigma_1\cdots\sigma_n$, and $P_0(\alpha)$ denotes the n-bit prefix of α (and $P_1(\alpha)$ denotes the n-bit suffix of α). In addition, algorithm D records this query (i.e., q_1). Subsequent queries are answered by first checking if their k-bit long prefix equals the k-bit long prefix of a previous query. In case the k-bit long prefix of the current query, denoted q_i, is different from the k-bit long prefixes of all previous queries, we associate this prefix a new input string (i.e., α_i). Namely, we answer query q_i by

$$G_{\sigma_n}(\cdots(G_{\sigma_{k+2}}(P_{\sigma_{k+1}}(\alpha_i)))\cdots)$$

where $q_i = \sigma_1\cdots\sigma_n$. In addition, algorithm D records the current query (i.e., q_i). The other possibility is that the k-bit long prefix of the i^{th} query equals the k-bit long prefix of some previous query. Let j be the smallest integer so that the k-bit long prefix of the i^{th} query equals the k-bit long prefix of the j^{th} query (by hypothesis $j < i$). Then, we record the current query (i.e., q_i) but answer it using the string associated with query q_j. Namely, we answer query q_i by

$$G_{\sigma_n}(\cdots(G_{\sigma_{k+2}}(P_{\sigma_{k+1}}(\alpha_j)))\cdots)$$

where $q_i = \sigma_1\cdots\sigma_n$. Finally, when machine M halts, algorithm D halts as well and outputs the same output as M.

Pictorially, algorithm D answers the first query by first placing the two halves of α_1 in the corresponding children of the tree-vertex reached by following the path from the root corresponding to $\sigma_1\cdots\sigma_k$. The labels of all vertices in the subtree corresponding to $\sigma_1\cdots\sigma_k$ are determined by the labels of these two children (as in the construction of F). Subsequent queries are answered by following the corresponding paths from the root. In case the path does not pass through a $(k+1)$-level vertex which has already a label, we assign this vertex and its sibling a new string (taken from the input). For sake of simplicity, in case the path of the i^{th} query requires a new string we use the i^{th} input string (rather than the first input string not used so far). In case the path of a new query passes through a $(k+1)$-level vertex which has been labelled already, we use this label to compute the labels of subsequent vertices along this path (and in particular the label of the leaf). We stress that the algorithm *does not* necessarily compute the labels of all vertices in a subtree corresponding to $\sigma_1\cdots\sigma_k$ (although these labels are determined by the label of the vertex corresponding to $\sigma_1\cdots\sigma_k$), but rather computes only the labels of vertices along the paths corresponding to the queries.

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

Clearly, algorithm D can be implemented in polynomial-time. It is left to evaluate its performance. The key observation is that when the inputs are taken from the $t(n)$-product of $G(U_n)$ and algorithm D chooses k as the checkpoint then M behaves exactly as on the k^{th} hybrid. Likewise, when the inputs are taken from the $t(n)$-product of U_{2n} and algorithm D chooses k as the checkpoint then M behaves exactly as on the $k+1^{\text{st}}$ hybrid. Namely,

Claim 3.6.5.1: Let n be an integer and $t \stackrel{\text{def}}{=} t(n)$. Let K be a random variable describing the random choice of checkpoint by algorithm D (on input a t-long sequence of $2n$-bit long strings). Then for every $k \in \{0, 1, ..., n-1\}$

$$\begin{aligned} \mathsf{Pr}\left(D(G(U_n^{(1)}), ..., G(U_n^{(t)})) = 1 \mid K = k\right) &= \mathsf{Pr}\left(M^{H_n^k}(1^n) = 1\right) \\ \mathsf{Pr}\left(D(U_{2n}^{(1)}, ..., U_{2n}^{(t)}) = 1 \mid K = k\right) &= \mathsf{Pr}\left(M^{H_n^{k+1}}(1^n) = 1\right) \end{aligned}$$

where the $U_n^{(i)}$'s and $U_{2n}^{(j)}$'s are independent random variables uniformly distributed over $\{0,1\}^n$ and $\{0,1\}^{2n}$, respectively.

The above claim is quite obvious, yet a rigorous proof is more complex than one realizes at first glance. The reason being that M's queries may depend on previous answers it gets, and hence the correspondence between the inputs of D and possible values assigned to the hybrids is less obvious than it seems. To illustrate the difficulty consider a n-bit string which is selected by a pair of interactive processes, which proceed in n iterations. At each iteration the first party chooses a new location, based on the entire history of the interaction, and the second process sets the value of this bit by flipping an unbiased coin. It is intuitively clear that the resulting string is uniformly distributed, and the same holds if the second party sets the value of the chosen locations using the outcome of a coin flipped beforehand. In our setting the situation is slightly more involved. The process of determining the string is terminated after $k < n$ iterations and statements are made of the partially determined string. Consequently, the situation is slightly confusing and we feel that a detailed argument is required.

Proof of Claim 3.6.5.1: We start by sketching a proof of the claim for the extremely simple case in which M's queries are the first t strings (of length n) in lexicographic order. Let us further assume, for simplicity, that on input $\alpha_1, ..., \alpha_t$, algorithm D happens to choose checkpoint k so that $t = 2^{k+1}$. In this case the oracle machine M is invoked on input 1^n and access to the function $f_{s_1,...,s_{2^{k+1}}}$, where $s_{2j-1+\sigma} = P_\sigma(\alpha_j)$ for every $j \leq 2^k$ and $\sigma \in \{0,1\}$. Thus, if the inputs to D are uniformly selected in $\{0,1\}^{2n}$ then M is invoked with access to the $k+1^{\text{st}}$ hybrid random variable (since in this case the s_j's are independent and uniformly distributed in $\{0,1\}^n$). On the other hand, if the inputs to D are distributed as $G(U_n)$ then M is invoked with access to the k^{th} hybrid random variable (since in this case $f_{s_1,...,s_{2^{k+1}}} = f_{r_1,...,r_{2^k}}$ where the r_j's are seeds corresponding to the α_j's).

For the general case we consider an alternative way of defining the random variable H_n^m, for every $0 \leq m \leq n$. This alternative way is somewhat similar to the way in which

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

D answers the queries of the oracle machine M. (We use the symbol m instead of k since m *does not* necessarily equal the checkpoint, denoted k, chosen by algorithm D.) This way of defining H_n^m consists of the interleaving of two random processes, which together first select at random a function $g : \{0,1\}^m \mapsto \{0,1\}^n$, that is later used to determine a function $f:\{0,1\}^n \mapsto \{0,1\}^n$. The first random process, denoted ρ, is an arbitrary process ("given to us from the outside"), which specifies points in the domain of g. (The process ρ corresponds to the queries of M, whereas the second process corresponds to the way A answers these queries.) The second process, denoted ψ, assigns uniformly selected n-bit long strings to every new point specified by ρ, thus defining the value of g on this point. We stress that in case ρ specifies an old point (i.e., a point for which g is already defined) then the second process does nothing (i.e., the value of g at this point is left unchanged). The process ρ may depend on the history of the two processes, and in particular on the values chosen for the previous points. When ρ terminates the second process (i.e., ψ) selects random values for the remaining undefined points (in case such exist). We stress that the second process (i.e., ψ) is fixed for all possible choices of a ("first") process ρ. The rest of this paragraph gives a detailed description of the interleaving of the two random processes (*and may be skipped*). We consider a randomized process ρ mapping sequences of n-bit strings (representing the history) to single m-bit strings. We stress that ρ is *not* necessarily memoryless (and hence may "remember" its previous random choices). Namely, for every fixed sequence $v_1,...,v_i \in \{0,1\}^n$, the random variable $\rho(v_1,...,v_i)$ is (arbitrarily) distributed over $\{0,1\}^m \cup \{\perp\}$ where $\perp$ is a special symbol denoting termination. A "random" function $g:\{0,1\}^m \mapsto \{0,1\}^n$ is defined by iterating the process ρ with the random process ψ defined below. Process ψ starts with g which is undefined on every point in its domain. At the i^{th} iteration ψ lets $p_i \stackrel{\text{def}}{=} \rho(v_1,...,v_{i-1})$ and, assuming $p_i \neq \perp$, sets $v_i \stackrel{\text{def}}{=} v_j$ if $p_i = p_j$ for some $j < i$ and lets v_i be uniformly distributed in $\{0,1\}^n$ otherwise. In the latter case (i.e., p_i is new and hence g is not yet defined on p_i), ψ sets $g(p_i) \stackrel{\text{def}}{=} v_i$ (in fact $g(p_i) = g(p_j) = v_j = v_i$ also in case $p_i = p_j$ for some $j < i$). When ρ terminates, i.e., $\rho(v_1,...,v_T) = \perp$ for some T, ψ completes the function g (if necessary) by choosing independently and uniformly in $\{0,1\}^n$ values for the points at which g is undefined yet. (Alternatively, we may augment the process ρ so that it terminates only after specifying all possible m-bit strings.)

Once a function g is totally defined, we define a function $f^g:\{0,1\}^n \mapsto \{0,1\}^n$ by

$$f^g(\sigma_1\sigma_2\cdots\sigma_n) \stackrel{\text{def}}{=} G_{\sigma_n}(\cdots(G_{\sigma_{k+2}}(G_{\sigma_{k+1}}(g(\sigma_k\cdots\sigma_1)))\cdots)$$

The reader can easily verify that f^g equals $f_{g(0^m),...,g(1^m)}$ (as defined in the hybrid construction above). Also, one can easily verify that the above random process (i.e., the interleaving of ψ with any ρ) yields a function g that is uniformly distributed over the set of all possible functions mapping m-bit strings to n-bit strings. It follows that the above described random process yields a result (i.e., a function) that is distributed identically to the random variable H_n^m.

Suppose now that the checkpoint chosen by D equals k and that D's inputs are independently and uniformly selected in $\{0,1\}^{2n}$. In this case the way in which D answers the

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

M's queries can be viewed as placing independently and uniformly selected n-bit strings as the labels of the $(k+1)$-level vertices. It follows that the way in which D answers M's queries corresponds to the above described process with $m = k+1$ (with M playing the role of ρ and A playing the role of ψ). Hence, in this case M is invoked with access to the $k+1^{\text{st}}$ hybrid random variable.

Suppose, on the other hand, that the checkpoint chosen by D equals k and that D's inputs are independently selected so that each is distributed identically to $G(U_n)$. In this case the way in which D answers the M's queries can be viewed as placing independently and uniformly selected n-bit strings as the labels of the k-level vertices. It follows that the way in which D answers the M's queries corresponds to the above described process with $m = k$. Hence, in this case M is invoked with access to the k^{th} hybrid random variable. $\Box$

Using Claim 3.6.5.1, it follows that

$$|\mathsf{Pr}\left(D(G(U_n^{(1)}),...,G(U_n^{(t)}))=1\right) - \mathsf{Pr}\left(D(U_{2n}^{(1)},...,U_{2n}^{(t)})=1\right)| = \frac{\Delta(n)}{n}$$

which, by the contradiction hypothesis is greater than $\frac{1}{n \cdot p(n)}$, for infinitely many n's. Using Theorem 3.2.6, we derive a contradiction to the hypothesis (of the current theorem) that G is a pseudorandom generator, and the current theorem follows. ■

3.6.3 A general methodology

Author's Note: *Ellaborate on the following.*

The following two-step methodology is useful in many cases:

1. Design your scheme assuming that all legitimate users share a random function, $f : \{0,1\}^n \mapsto \{0,1\}^n$. (The adversaries may be able to obtain, from the legitimate users, the values of f on arguments of their choice, but do not have direct access to f.) This step culminates in proving the security of the scheme assuming that f is indeed uniformly chosen among all possible such functions, while ignoring the question of how such an f can be selected and handled.

2. Construct a real scheme by replacing the random function by a pseudorandom function. Namely, the legitimate users will share a random/secret seed specifying such a pseudorandom function, whereas the adversaries do not know the seed. As before, at most the adversaries may obtain (from the legitimate users) the value of the function at arguments of their choice. Finally, conclude that the real scheme (as presented above) is secure (since otherwise one could distinguish a pseudorandom function from a truly random one).

We stress that the above methodology may be applied only if legitimate users can share random/secret information not known to the adversary (i.e., as is the case in private-key encryption schemes).

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

3.7 * Pseudorandom Permutations

In this section we present definitions and constructions for pseudorandom permutations. Clearly, pseudorandom permutations (over huge domains) can be used instead of pseudorandom functions in any efficient application, yet pseudorandom permutation offer the extra advantage of having unique preimages. This extra advantage may be useful sometimes, but not always (e.g., it is not used in the rest of this book). The construction of pseudorandom permutation uses pseudorandom functions as a building block, in a manner identical to the high level structure of the DES. Hence, the proof presented in this section can be viewed as a supporting the DES's methodology of converting "randomly looking" functions into "randomly looking" permutations. (The fact that in the DES this methodology is applied to functions which are not "randomly looking" is not of our concern here.)

3.7.1 Definitions

We start with the definition of pseudorandom permutations. Loosely speaking a pseudorandom ensemble of permutations is defined analogously to a pseudorandom ensemble of functions. Namely,

Definition 3.7.1 (permutation ensembles): *A* permutation ensemble *is a sequence $P = \{P_n\}_{n\in\mathbb{N}}$ of random variables, so that the random variable P_n assumes values in the set of permutations mapping n-bit long strings to n-bit long strings. The* uniform permutation ensemble, *denoted $K = \{K_n\}_{n\in\mathbb{N}}$, has K_n uniformly distributed over the set of permutations mapping n-bit long strings to n-bit long strings.*

Every permutation ensemble is a function ensemble. Hence, the definition of an *efficiently computable* permutation ensemble is obvious (i.e., it is derived from the definition of an efficiently computable function ensemble). Pseudorandom permutations are defined as computationally indistinguishable from the uniform *permutation* ensemble.

Definition 3.7.2 (pseudorandom permutation ensembles): *A permutation ensemble, $P = \{P_n\}_{n\in\mathbb{N}}$, is called* pseudorandom *if for every probabilistic polynomial-time oracle machine M, every polynomial $p(\cdot)$ and all sufficiently large n's*

$$|\mathsf{Pr}(M^{P_n}(1^n)=1) - \mathsf{Pr}(M^{K_n}(1^n)=1)| < \frac{1}{p(n)}$$

where $K = \{K_n\}_{n\in\mathbb{N}}$ is the uniform permutation ensemble.

The fact that P is a pseudorandom permutation ensemble rather then just being a pseudorandom function ensemble cannot be detected in poly(n)-time by an observer given oracle access to P_n. This fact steams from the observation that the uniform permutation ensemble is polynomial-time indistinguishable from the uniform function ensemble. Namely,

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

Proposition 3.7.3 *The uniform permutation ensemble (i.e., $K = \{K_n\}_{n\in\mathbb{N}}$) constitutes a pseudorandom function ensemble.*

Proof Sketch: The probability that an oracle machine detects a collision in the oracle-function, when given access to H_n, is bounded by $t^2 \cdot 2^{-n}$, where t denotes the number of queries made by the machine. Conditioned on not finding such a collision, the answers of H_n are indistinguishable from those of K_n. Finally, using the fact that a polynomial-time machine can ask at most polynomially many queries, the proposition follows. ■

Hence, using pseudorandom permutations instead of pseudorandom functions has reasons beyond the question of whether a computationally restricted observer can detect the difference. Typically, the reason is that one wants to be guaranteed of the *uniqueness* of preimages. A natural strengthening of this requirement is to require that, given the description of the permutation, *the* (unique) *preimage can be efficiently found.*

Definition 3.7.4 (efficiently computable and invertible permutation ensembles): *A permutation ensemble, $P = \{P_n\}_{n\in\mathbb{N}}$, is called* efficiently computable and invertible *if the following three conditions hold*

1. (efficient indexing): *There exists a probabilistic polynomial time algorithm, I, and a mapping from strings to permutation, ϕ, so that $\phi(I(1^n))$ and P_n are identically distributed.*

2. (efficient evaluation): *There exists a probabilistic polynomial time algorithm, V, so that $V(i,x) = f_i(x)$, where (as before) $f_i \stackrel{\rm def}{=} \phi(i)$.*

3. (efficient inversion): *There exists a probabilistic polynomial time algorithm, N, so that $N(i,x) = f_i^{-1}(x)$ (i.e., $f_i(N(i,x))=x$).*

Items (1) and (2) are guaranteed by the definition of an efficiently computable permutation ensemble. The additional requirement is stated in item (3). In some settings it makes sense to augment also the definition of a pseudorandom ensemble by requiring that the ensemble cannot be distinguished from the uniform one even when the observer gets access to two oracles: one for the permutation and the other for its inverse.

Definition 3.7.5 (strong pseudorandom permutations): *A permutation ensemble, $P = \{P_n\}_{n\in\mathbb{N}}$, is called* strongly pseudorandom *if for every probabilistic polynomial-time oracle machine M, every polynomial $p(\cdot)$ and all sufficiently large n's*

$$|\mathsf{Pr}(M^{P_n,P_n^{-1}}(1^n)=1) - \mathsf{Pr}(M^{K_n,K_n^{-1}}(1^n)=1)| < \frac{1}{p(n)}$$

where $M^{f,g}$ can ask queries to both of its oracles (e.g., query $(1,q)$ is answered by $f(q)$, whereas query $(2,q)$ is answered by $g(q)$).

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

3.7.2 Construction

The construction of pseudorandom permutation uses pseudorandom functions as a building block, in a manner identical to the high level structure of the DES. Namely,

Construction 3.7.6 *Let $f:\{0,1\}^n \mapsto \{0,1\}^n$. For every $x, y \in \{0,1\}^n$, we define*

$$\mathrm{DES}_f(x,y) \stackrel{\mathrm{def}}{=} (y, x \oplus f(y))$$

where $x \oplus y$ denotes the bit-by-bit exclusive-or of the binary strings x and y. Likewise, for $f_1, ..., f_t:\{0,1\}^n \mapsto \{0,1\}^n$, we define

$$\mathrm{DES}_{f_t,\ldots,f_1}(x,y) \stackrel{\mathrm{def}}{=} \mathrm{DES}_{f_t,\ldots,f_2}(\mathrm{DES}_{f_1}(x,y))$$

For every function ensemble $F = \{F_n\}_{n\in\mathbb{N}}$, and every function $t : \mathbb{N} \mapsto \mathbb{N}$, we define the function ensemble $\{\mathrm{DES}_{F_n}^{t(n)}\}_{n\in\mathbb{N}}$ by letting $\mathrm{DES}_{F_n}^{t(n)} \stackrel{\mathrm{def}}{=} \mathrm{DES}_{F_n^{(t)},\ldots,F_n^{(1)}}$, where $t = t(n)$ and the $F_n^{(i)}$'s are independent copies of the random variable F_n.

Theorem 3.7.7 *Let F_n, $t(\cdot)$, and $\mathrm{DES}_{F_n}^{t(n)}$ be as in Construction 3.7.6 (above). Then, for every polynomial-time computable function $t(\cdot)$, the ensemble $\{\mathrm{DES}_{F_n}^{t(n)}\}_{n\in\mathbb{N}}$ is an efficiently computable and invertible permutation ensemble. Furthermore, if $F = \{F_n\}_{n\in\mathbb{N}}$ is a pseudorandom function ensemble then the ensemble $\{\mathrm{DES}_{F_n}^{3}\}_{n\in\mathbb{N}}$ is pseudorandom, and the ensemble $\{\mathrm{DES}_{F_n}^{4}\}_{n\in\mathbb{N}}$ is strongly pseudorandom.*

Clearly, the ensemble $\{\mathrm{DES}_{F_n}^{t(n)}\}_{n\in\mathbb{N}}$ is efficiently computable. The fact that it is a permutation ensemble, and furthermore one with efficient inverting algorithm, follows from the observation that for every $x, y \in \{0,1\}^n$

$$\begin{aligned}
\mathrm{DES}_{f,\mathrm{zero}}(\mathrm{DES}_f(x,y)) &= \mathrm{DES}_{f,\mathrm{zero}}(y, x \oplus f(y)) \\
&= \mathrm{DES}_f(x \oplus f(y), x) \\
&= (y, (x \oplus f(y)) \oplus f(y)) \\
&= (x, y)
\end{aligned}$$

where $\mathrm{zero}(z) \stackrel{\mathrm{def}}{=} 0^{|z|}$ for all $z \in \{0,1\}^n$.

To prove the pseudorandomness of $\{\mathrm{DES}_{F_n}^{3}\}_{n\in\mathbb{N}}$ (resp., strong pseudorandomness of $\{\mathrm{DES}_{F_n}^{4}\}_{n\in\mathbb{N}}$) it suffices to prove the pseudorandomness of $\{\mathrm{DES}_{H_n}^{3}\}_{n\in\mathbb{N}}$ (resp., strong pseudorandomness of $\{\mathrm{DES}_{H_n}^{4}\}_{n\in\mathbb{N}}$). The reason being that if, say, $\{\mathrm{DES}_{H_n}^{3}\}_{n\in\mathbb{N}}$ is pseudorandom while $\{\mathrm{DES}_{F_n}^{3}\}_{n\in\mathbb{N}}$ is not, then one can derive a contradiction to the pseudorandomness of the function ensemble F (i.e., a hybrid argument is used to bridge between the three copies of H_n and the three copies of F_n). Hence, Theorem 3.7.7 follows from

Proposition 3.7.8 $\{\mathrm{DES}_{H_n}^{3}\}_{n\in\mathbb{N}}$ *is pseudorandom, whereas* $\{\mathrm{DES}_{H_n}^{4}\}_{n\in\mathbb{N}}$ *is strongly pseudorandom.*

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

Proof Sketch: We start by proving that $\{\mathrm{DES}^3_{H_n}\}_{n\in\mathbb{N}}$ is pseudorandom. Let $P_{2n} \stackrel{\rm def}{=} \{\mathrm{DES}^3_{H_n}\}_{n\in\mathbb{N}}$, and K_{2n} be the random variable uniformly distributed over all possible permutation acting on $\{0,1\}^{2n}$. We prove that for every oracle machine, M, that, on input 1^n, asks at most m queries, it holds that

$$|\mathsf{Pr}(M^{P_{2n}}(1^n)=1) - \mathsf{Pr}(M^{K_{2n}}(1^n)=1)| \leq \frac{2m^2}{2^n}$$

Let $q_i = (L^0_i, R^0_i)$, with $|L^0_i|=|R^0_i|=n$, denote the random variable representing the $i^{\rm th}$ query of M when given access to oracle P_{2n}. Recall that $P_{2n} = \mathrm{DES}_{H^{(3)}_n, H^{(2)}_n, H^{(1)}_n}$, where the $H^{(j)}_n$'s are three independent random variables each uniformly distributed over the functions acting on $\{0,1\}^n$. Let $R^{k+1}_i \stackrel{\rm def}{=} L^k_i \oplus H^{(k+1)}_n(R^k_i)$ and $L^{k+1}_i \stackrel{\rm def}{=} R^k_i$, for $k=0,1,2$. We assume, without loss of generality, that M never asks the same query twice. We define the following a random variable ζ_m representing the event "there exists $i<j\leq m$ and $k\in\{1,2\}$ so that $R^k_i = R^k_j$" (namely, "on input 1^n and access to oracle P_{2n} two of the m first queries of M satisfy the relation $R^k_i = R^k_j$). Using induction on m, the reader can prove concurrently the following two claims (see guidelines below).

Claim 3.7.8.1: Given $\neg\zeta_m$, we have the R^3_i's uniformly distributed over $\{0,1\}^n$ and the L^3_i's uniformly distributed over the n-bit strings not assigned to previous L^3_j's. Namely, for every $\alpha_1, ..., \alpha_m \in \{0,1\}^n$

$$\mathsf{Pr}\left(\wedge_{i=1}^m (R^3_i=\alpha_i) \,|\, \neg\zeta_m\right) = \left(\frac{1}{2^n}\right)^m$$

whereas, for every *distinct* $\beta_1, ..., \beta_m \in \{0,1\}^n$

$$\mathsf{Pr}\left(\wedge_{i=1}^m (L^3_i=\beta_i) \,|\, \neg\zeta_m\right) = \prod_{i=1}^m \frac{1}{2^n - i + 1}$$

Claim 3.7.8.2:

$$\mathsf{Pr}\left(\zeta_{m+1} \,|\, \neg\zeta_m\right) \leq \frac{2m}{2^n}$$

Proof Idea: The proof of Claim 3.7.8.1 follows by observing that the R^3_i's are determined by applying the random function $H^{(3)}_n$ to different arguments (i.e., the R^2_i's), whereas the $L^3_i = R^2_i$'s are determined by applying the random function $H^{(2)}_n$ to different arguments (i.e., the R^1_i's) and conditioning that the R^2_i's are different. The proof of Claim 3.7.8.2 follows by considering the probability that $R^k_{m+1} = R^k_i$, for some $i \leq m$ and $k \in \{1,2\}$. Say that $R^0_i = R^0_{m+1}$ then certainly (by recalling $q_i \neq q_{m+1}$) we have

$$R^1_i = L^0_i \oplus H^{(1)}_n(R^0_i) = L^0_i \oplus H^{(1)}_n(R^0_j) \neq L^0_{m+1} \oplus H^{(1)}_n(R^0_{m+1}) = R^1_{m+1}$$

On the other hand, say that $R^0_i \neq R^0_{m+1}$ then

$$\mathsf{Pr}\left(R^1_i = R^1_{m+1}\right) = \mathsf{Pr}\left(H^{(1)}_n(R^0_i) \oplus H^{(1)}_n(R^0_{m+1}) = L^0_i \oplus L^0_{m+1}\right) = 2^{-n}$$

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

Furthermore, if $R_i^1 \neq R_{m+1}^1$ then

$$\mathsf{Pr}\left(R_i^2 = R_{m+1}^2\right) = \mathsf{Pr}\left(H_n^{(2)}(R_i^1) \oplus H_n^{(2)}(R_{m+1}^1) = R_i^0 \oplus R_{m+1}^0\right) = 2^{-n}$$

Hence, both claims follow. $\square$

Combining the above claims, we conclude that $\mathsf{Pr}(\zeta_m) < \frac{m^2}{2^n}$, and furthermore, given that ζ_m is false, the answers of P_{2n} 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_{2n} 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_{2n} or by K_{2n}) is bounded by $\frac{2m^2}{2^n}$. The first part of the proposition follows.

The proof that $\{\mathrm{DES}^4_{H_n}\}_{n\in\mathbb{N}}$ is strongly pseudorandom is more complex, yet uses essentially the same ideas. In particular, the event corresponding to ζ_m is the disjunction of four types of events. Events of the first type are of the form $R_i^k = R_j^k$ for $k \in \{2,3\}$, where $q_i = (L_i^0, R_i^0)$ and $q_j = (L_j^0, R_j^0)$ are queries of the forward direction. Similarly, events of the second type are of the form $R_i^k = R_j^k$ for $k \in \{2,1\}$, where $q_i = (L_i^4, R_i^4)$ and $q_j = (L_j^4, R_j^4)$ are queries of the backwards direction. Events of the third type are of the form $R_i^k = R_j^k$ for $k \in \{2,3\}$, where $q_i = (L_i^0, R_i^0)$ is of the forward direction, $q_j = (L_j^4, R_j^4)$ is of the backward direction, and $j < i$. Similarly, events of the fourth type are of the form $R_i^k = R_j^k$ for $k \in \{2,1\}$, where $q_i = (L_i^4, R_i^4)$ is of the forward direction, $q_j = (L_j^0, R_j^0)$ is of the backward direction, and $j < i$. As before, one bounds the probability of event ζ_m, and bounds the statistical distance between answers by K_{2n} and answers by $\{\mathrm{DES}^4_{H_n}\}_{n\in\mathbb{N}}$ given that ζ_m is false. $\blacksquare$

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 [GK89a].

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. [GMT82] and Blum et. al. [BBS82]. Consequently, Yao proved that

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

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$_2$ 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 1989) contains several pointers to works which present applications of pseudorandom functions; e.g.,* [GGM84b], [G86] *and* [G87b,O89]. *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,K88]. In particular, [ACGS84] presents pseudoran-

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

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].

Author's Note: *The reference is – M. Li and P. Vitányi,* An Introduction to Kolmogorov Complexity and its Applications, *Springer-Verlag, 1993.*

3.8.3 Open Problems

Although Hastad 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 $< \sqrt[4]{n}$. 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 $f:\{0,1\}^* \mapsto \{0,1\}^*$ 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

$$\Delta_L(n) \stackrel{\text{def}}{=} |\mathsf{Pr}(X_n \in L) - \mathsf{Pr}(Y_n \in L)|$$

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

is negligible in n.

It is tempting to think the 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 probabilitic 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 $L \stackrel{\text{def}}{=} \{x : \mathsf{Pr}(A(x)=) \geq \frac{1}{2}\}$, then L is not necessarily in $\mathcal{BPP}$.

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 $S \subseteq \{0,1\}^*$,

$$\Delta_S(n) \stackrel{\text{def}}{=} |\mathsf{Pr}(X_n \in S) - \mathsf{Pr}(Y_n \in S)|$$

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 $f:\{0,1\}^* \mapsto \{0,1\}$ a set $S_f \stackrel{\text{def}}{=} \{x : f(x)=1\}$.)

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 $C \stackrel{\text{def}}{=} \{C_n\}_{n\in\mathbb{N}}$ be a family of probabilistic polynomial-size circuits. Prove that there exists a family of (deterministic) polynomial-size circuits, $D \stackrel{\text{def}}{=} \{D_n\}_{n\in\mathbb{N}}$, so that for every n

$$\Delta_D(n) \geq \Delta_C(n)$$

where

$$\begin{aligned}\Delta_D(n) &\stackrel{\text{def}}{=} |\mathsf{Pr}(D_n(X_n))=1) - \mathsf{Pr}(D_n(Y_n))=1)| \\ \Delta_C(n) &\stackrel{\text{def}}{=} |\mathsf{Pr}(C_n(X_n))=1) - \mathsf{Pr}(C_n(Y_n))=1)|\end{aligned}$$

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

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

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

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 $y^{k+2}, ..., y^m$ 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_i's, each X_n has support of size at most $t+1$ and $\Pr(f(X_n) = 1) \leq \Pr(f_i(U_n) = 1)$, for each $i = 1, ..., t$. (Extra hint – consider the t-dimentional 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 [GM97indist].)

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 $\{U_{l(n)}\}_{n\in\mathbb{N}}$ 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 α, 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 α is longer than the length recorded.

Exercise 11: Consider a modification of Construction 3.3.3, where $s_i\sigma_i = G_1(s_{i-1})$ is used instead of $\sigma_i s_i = G_1(s_{i-1})$. 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 $G'(s) \stackrel{\rm def}{=} G(s) \oplus s0^{|G(s)|-|s|}$ is also a pseudorandom generator.

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

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

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 $G'(s) \stackrel{\text{def}}{=} h(G(s))$ and $G''(s) \stackrel{\text{def}}{=} G(h(s))$ 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., $G(s) \stackrel{\text{def}}{=} G_1^{p(|s|)}(s)$ where $G_1^0(s) \stackrel{\text{def}}{=} s$ and $G_1^{i+1} \stackrel{\text{def}}{=} G_1(G_1^i(s))$). 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 $l : \mathbb{N} \mapsto \mathbb{N}$ so that X_n ranges over string of length $l(n)$, and $l(n)$ can be computed in time 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 $\{U_{l(n)}\}_{n \in \mathbb{N}}$, 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: an argument analogous to that of the proof of Theorem 3.4.1 has to be used as well.

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

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 $n + m - 1$, where as 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 δ be as in Lemma 3.5.1. Prove that, for every set $S \subseteq \{0,1\}^m$, and for all but a $2^{-(b-m+\log_2|S|)}\delta^{-2}$ fraction of the h's in S_n^m, it holds that

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

(Hint: Follow the proof of Lemma 3.5.1, defining $\zeta_x(h) = 1$ 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 $S \subseteq \{0,1\}^m$ be sets, and let $b \stackrel{\text{def}}{=} \log_2|B|$ and $s \stackrel{\text{def}}{=} \log_2|S|$. Prove that, for all but a $\frac{2^m}{|B|\cdot|S|} \cdot \delta^{-2}$ 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

$$f_s(x) \stackrel{\text{def}}{=} G_{\sigma_n}(\cdots(G_{\sigma_2}(G_{\sigma_1}(x))\cdots)$$

where $s = \sigma_1 \cdots \sigma_n$, and G_σ is as in Construction 3.6.4. Namely the roles of x and s in Construction 3.6.4 are switched (i.e., the root is labelled x 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 G is a pseudorandom generator).

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

Guideline: Show, first, that if pseudorandom generators exist then there exists a pseudorandom generator G satisfying $G(0^n) = 0^{2n}$.

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 $f : \{0,1\}^* \mapsto \{0,1\}$). 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 $f : \{0,1\}^* \mapsto \{0,1\}^*$).

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

$$\mathsf{Pr}(\mathrm{corr}^{F_n}(M^{F_n}(1^n))) < \frac{1}{2} + \frac{1}{p(n)}$$

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 $\mathrm{corr}^f(x, \sigma)$ is defined as the predicate "$f(x) = \sigma$". Intuitively, after getting the value of f on points of its choice, the machine M

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

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

$$\mathsf{Pr}(M^{F_n}(U_n) = F_n(U_n)) < \frac{1}{2} + \frac{1}{p(n)}$$

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

$$\mathsf{Pr}(M^{F_n}(X_n) = F_n(X_n)) < \frac{1}{2} + \frac{1}{p(n)}$$

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

Exercise 27: Let F_n and $\mathrm{DES}^t_{F_n}$ 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 $\mathrm{DES}^2_{F_n}$ is *not* pseudorandom. Similarly, prove that the ensemble $\mathrm{DES}^3_{F_n}$ is *not strongly* pseudorandom.

Guideline: Start by showing that the ensemble $\mathrm{DES}^1_{F_n}$ is *not* pseudorandom.

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