

Chapter 5

One-way functions and pseudo-random generators

5.1 Chapter overview and basic definitions

Modern cryptography relies in an essential way on complexity theory. In cryptography, the typical objective is to design protocols whose functionality (e.g., the secrecy of a message, the authentication of the participating parties, etc.) cannot be altered by the malicious actions of an adversary. The most general and safest approach in cryptography is to consider as a parameter a bound for the computational power of the adversary and to ensure the functionality of the protocol against any malicious action that can be performed within this bound. This amounts to proving that any successful adversarial attack requires more computation than the assumed bound, a mission which falls in the territory of complexity theory. The cryptographical protocols need to utilize as basic primitives tasks that are computationally hard for the attacker. Moreover, having in mind the cryptographical applications, it is essential to quantify carefully the hardness of these primitives, which implies the need for a thorough quantitative analysis of the computational complexity of these tasks. In this chapter we undertake such an investigation for two such primitives, one-way functions and pseudo-random generators. We also discuss two related concepts, hard functions (which are a relaxation of one-way functions), and extractors (which are both a relaxation and a strengthening, in different aspects, of pseudo-random generators).

A one-way function is a function that is easy to compute and hard to invert. A pseudo-random generator is a function that takes a short random input, called the seed, and produces a long output that “looks” random to an adversary. These two types of functions are important per se and have numerous applications in computational complexity theory. In cryptography they play a major role and

almost all cryptosystems and cryptographic protocols rely on them in a quite direct way. To illustrate, we give just two simple and familiar examples.

Example 1. Consider the way passwords are handled in a multi-user computer system. Instead of storing them explicitly, the system keeps, for each password w , the value $f(w)$, where f ideally is a one-way function. At login, the user types w and the system computes $f(w)$ and compares it with the stored value. On the other hand, since f is one-way, no one (e.g., the system administrator) having $f(w)$ is able to retrieve w .

Example 2. The one-time pad encryption scheme works by bitwise XOR-ing the message m with a random string R that acts as the private key, i.e., the ciphertext c is obtained as $c = m \oplus R$ (\oplus denotes bitwise XOR). This is a perfect encryption scheme and, in fact, it is known that in any private key encryption scheme that does not leak any information to an adversary the length of the private key has to be at least as large as the message being encrypted. The drawback is that the two legal parties (usually called Alice and Bob) must share an extremely long private key. With a pseudo-random generator g , Alice and Bob need to share only a short seed r and encrypt the message m by $c = m \oplus g(r)$.

Let us now define formally the two notions of primary interest in this chapter. We start with one-way functions, and we first introduce some notation.

NOTATION. We will use more operations on binary strings and we introduce here notation that distinguishes them clearly. We recall that Σ denotes the binary alphabet $\{0, 1\}$, $|x|$ is the length of a string x , and $\|A\|$ denotes the cardinality of a set A . The concatenation of two binary strings x and y is denoted $x \odot y$.¹ This notation is extended to the concatenation of more strings and we write $x_1 \odot x_2 \odot x_3 \odot \dots \odot x_m$ instead of $(\dots((x_1 \odot x_2) \odot x_3) \dots \odot x_m)$. For strings x and y having the same length (i.e., $x, y \in \Sigma^n$ for some $n \in \mathbb{N}$), we define the inner product of x and y , denoted $x \cdot y$, as follows. We view x and y as vectors over the field $GF(2)$, $x = (x_1, \dots, x_n)$ and $y = (y_1, \dots, y_n)$, where the x_i 's and the y_j 's, the bits that form x and respectively y , are identified with elements of $GF(2)$ in the natural way: If the i -th bit of x is bit 0(1), then x_i is identified with the element 0(1) in $GF(2)$. Then, the inner product is $x \cdot y = x_1 \cdot y_1 + \dots + x_n \cdot y_n$, the arithmetical operations being done modulo 2, i.e., in $GF(2)$. Finally, we also use cartesian products of sets of strings and we use the standard tuple notation (i.e., (y_1, y_2, \dots, y_k)) to denote the elements of such a cartesian products.

We consider functions $f: \Sigma^* \rightarrow \Sigma^*$ with the property that, for all x_1 and x_2 , $|x_1| = |x_2|$ implies $|f(x_1)| = |f(x_2)|$. Such a function is called *length-regular*. This restriction is mainly a technical convenience and, moreover, appears natural for most cryptographic applications. A function $f: \Sigma^* \rightarrow \Sigma^*$ having the property that, for all $x \in \Sigma^*$, $|f(x)| = |x|$ is called *length-preserving*.

For any function $f: \Sigma^* \rightarrow \Sigma^*$ and $\ell \in \mathbb{N}$, f_ℓ denotes the restriction of f to Σ^ℓ . The set of functions $\{f_\ell: \Sigma^\ell \rightarrow \Sigma^* \mid \ell \in \mathbb{N}\}$, usually denoted $(f_\ell)_{\ell \in \mathbb{N}}$, is

¹This notation for concatenation is valid in this chapter only. We felt the need for a more striking notation here so as not to confuse concatenation with the other string operations.

called the *ensemble of functions* induced by f . Conversely, given a set of functions $\{f_\ell: \Sigma^\ell \rightarrow \Sigma^* \mid \ell \in \mathbb{N}\}$, the function $f: \Sigma^* \rightarrow \Sigma^*$ defined by $f(x) = f_{|x|}(x)$, for all $x \in \Sigma^*$, is called the function induced by the ensemble $(f_\ell)_{\ell \in \mathbb{N}}$.

Note that if f is length-regular, then, for each ℓ , there is a natural number ℓ' such that $f_\ell: \Sigma^\ell \rightarrow \Sigma^{\ell'}$. We want f to be computable in polynomial time which implies that the length of $f(x)$ is polynomially bounded in the length of x . In other words, ℓ' is bounded by $p(\ell)$, for some polynomial p . We also want f to be hard to invert. One trivial way to achieve this is to make ℓ' much smaller than ℓ (for example, assume that f_ℓ shrinks its input by an exponential factor). In that case no polynomial-time algorithm, on input $f(x)$, has the time to print x . This kind of “hard-to-invert” function is neither useful in cryptography nor interesting in computational complexity and, consequently, to avoid this situation, we will require that ℓ is polynomially bounded in ℓ' (i.e., $\ell \leq q(\ell')$, for some polynomial q) as well. If these two requirements are met for f , we say that the input and the output lengths are *polynomially related*.

We can now present the main definitions.

Definition 5.1.1 (One-way function) *Let $\epsilon: \mathbb{N} \rightarrow [0, 1]$ and $S: \mathbb{N} \rightarrow \mathbb{N}$ be two functions that are considered as parameters. A length regular function $f: \Sigma^* \rightarrow \Sigma^*$ with polynomially related input and output lengths is a one-way function with security (ϵ, S) if*

- (1) *There exists a deterministic polynomial-time machine M such that, for all $x \in \Sigma^*$, $M(x) = f(x)$;*
- (2) *For all sufficiently large ℓ and for any circuit C with $\text{size}(C) \leq S(\ell)$,*

$$\text{Prob}_{x \in \Sigma^\ell}(C(f_\ell(x)) \in f_\ell^{-1}(f_\ell(x))) < \epsilon(\ell).$$

A few remarks are necessary. Stating that f is easy to calculate does not raise any problem: We ask that there is a polynomial-time algorithm that calculates f .² Stating that f is hard to invert needs some elaboration. We want f to be resistant to inversion by an adversary endowed with some specified computational power. An adversary is represented by a circuit that attempts to invert f (at a given length) and $S(\ell)$ represents the computational power against which f is inversion resistant. The adversary is given $f_\ell(x)$ and is not looking strictly to retrieve x (since f is not necessarily 1-to-1 this would be impossible) but only some inverse of $f_\ell(x)$. We require that this happens with probability less than $\epsilon(\ell)$, where the probability is taken over x chosen uniformly at random in Σ^* .

Currently it is not known if one-way functions exist. Note that, in fact, an adversary can invert $f_\ell(x)$ *nondeterministically* in polynomial time by just guessing a value z such that $f_\ell(z) = f_\ell(x)$. Thus, the existence of one-way functions implies

²We could require that f is calculated by a *probabilistic* polynomial-time algorithm. All the results that we present here would remain valid with minor and obvious modifications.

$P \neq NP$. Even under the (plausible) assumption $P \neq NP$, it is not known whether good one-way functions exist. The main reason is that the function needs to be hard to invert on a large fraction of inputs at almost every length, while $P \neq NP$ only means that there are languages that are hard in the worst-case (perhaps on just one input per length, and not even for almost every length). However, the general opinion is that good one-way functions do exist and there are some candidates for one-way functions (e.g., based on integer factoring or on the discrete log problem) that are used with relatively high confidence in practice.

Depending on the parameters ϵ and S (in the Definition 5.1.1), we distinguish some important types of one-way functions. We mainly consider adversaries endowed with circuits whose size is larger than any polynomial function.

Definition 5.1.2 *A function $s: \mathbb{N} \rightarrow \mathbb{N}$ is superpolynomial if for every polynomial p , it holds that $s(\ell) \geq p(\ell)$, for all ℓ sufficiently large.*

Definition 5.1.3 *Let $C = (C_\ell)_{\ell \in \mathbb{N}}$ be a collection of circuits and let $S: \mathbb{N} \rightarrow \mathbb{N}$ be a function. We say that the circuits C have size at most S if, for every ℓ , $\text{size}(C_\ell) \leq S(\ell)$.*

Definition 5.1.4 (Strong one-way function) *A length-regular function $f: \Sigma^* \rightarrow \Sigma^*$ with polynomially related input and output lengths is a strong one-way function if for any polynomial p , f is one-way with security $(\frac{1}{p(\ell)}, p(\ell))$.*

We also consider the following particular type of a strong one-way function.

Definition 5.1.5 (Exponentially strong one-way function) *A length-regular function $f: \Sigma^* \rightarrow \Sigma^*$ with polynomially related input and output lengths is an exponentially strong one-way function if there is some positive constant c such that f is one-way with security $(\frac{1}{2^{c\ell}}, 2^{c\ell})$.*

In cryptography applications, one needs strong one-way functions. We will show that, in fact, it suffices if we have at hand a much weaker type of one-way function. Indeed, in the next section, we show that given a weak one-way function, as defined next, we can construct a strong one-way function.

Definition 5.1.6 (Weak one-way function) *A length-regular function $f: \Sigma^* \rightarrow \Sigma^*$ with polynomially related input and output lengths is a weak one-way function if there is a polynomial q such that for any polynomial p , f is one-way with security $(1 - \frac{1}{q(\ell)}, p(\ell))$.*

We move on to define formally pseudo-random generators. Intuitively, a pseudo-random generator is a function that takes random short strings and produces (much) longer strings that “look” random to an adversary. It is also desirable that the function is efficiently computable. Essential for our discussion are distributions on sets of binary strings of a given length, i.e., distributions on Σ^n , where n is some arbitrary natural number. Recall that a distribution X_n on Σ^n is a function $X_n: \Sigma^n \rightarrow [0, 1]$ with the property that $\sum_{\alpha \in \Sigma^n} X_n(\alpha) = 1$. A distribution will also be identified with a random variable having that distribution.

NOTATION. For each $n \in \mathbb{N}$, U_n denotes the uniform distribution on Σ^n , i.e., $U_n: \Sigma^n \rightarrow [0, 1]$ is the function defined by $U_n(\alpha) = \frac{1}{2^n}$, for all $\alpha \in \Sigma^n$.

Suppose that in some application (e.g., a cryptography protocol, or the execution of a probabilistic algorithm) we need random strings of length n . Ideally, we would like to utilize strings in Σ^n generated by some source of randomness according to the uniform distribution. Lacking this, we are also content if the source generates strings in Σ^n according to a distribution X_n on Σ^n that is *close* to U_n . There are more ways by which two distributions can be close. They depend on the type of distance between two distributions that we consider, and there are two distances that are relevant for us, the *statistical distance* and the *computational distance*. Let us first consider the statistical distance.

Definition 5.1.7 (Statistical distance between two distributions) *Let $n \in \mathbb{N}$. Let X_n, Y_n be two distributions on Σ^n . The statistical distance between X_n and Y_n is denoted $\Delta_{\text{stat}}(X_n, Y_n)$ and is defined by*

$$\Delta_{\text{stat}}(X_n, Y_n) = \sum_{\alpha \in \Sigma^n} |X_n(\alpha) - Y_n(\alpha)|.$$

For example, consider the following distributions, X_3, Y_3 and Z_3 defined on Σ^3 .

α	000	001	010	011	100	101	110	111
X_3	0	0	0	0	1/4	1/4	1/4	1/4
Y_3	1/16	3/16	1/16	3/16	1/16	3/16	1/16	3/16
Z_3	1/64	1/64	1/64	1/64	15/64	15/64	15/64	15/64

Note that $\Delta_{\text{stat}}(X_3, Y_3) = 1$ and $\Delta_{\text{stat}}(X_3, Z_3) = 1/8$ which agrees with the intuition that X_3 and Z_3 resemble each other more than X_3 and Y_3 .

One way to estimate the closeness of two distributions X_n and Y_n is to take some subset $A \subseteq \Sigma^n$ and to compare $\text{Prob}_{X_n}(A)$ and $\text{Prob}_{Y_n}(A)$. Such a subset is called (in this context) a *statistical test* or, simply, a test. To illustrate, let us take, in the above example, the test $A_1 = \{011, 100\}$. Then $\text{Prob}_{X_3}(A_1) = \text{Prob}_{Y_3}(A_1) = \frac{1}{4}$. Thus the test A_1 is not able to distinguish between the distributions X_3 and Y_3 . If we take the test $A_2 = \{000, 001, 010, 011\}$, then $\text{Prob}_{X_3}(A_2) = 0$ and $\text{Prob}_{Y_3}(A_2) = 1/2$. The test A_2 “sees” a quite significant difference between X_3 and Y_3 . It is not hard to observe that the test A_2 and its complement $A'_2 = \{100, 101, 110, 111\}$ “see” the largest difference between X_3 and Y_3 among all tests. In fact, the following lemma holds.

Lemma 5.1.8 *Let $n \in \mathbb{N}$, and let X_n, Y_n be two distributions on Σ^n . Then*

$$\Delta_{\text{stat}}(X_n, Y_n) = 2 \max_{A \subseteq \Sigma^n} |\text{Prob}_{X_n}(A) - \text{Prob}_{Y_n}(A)|.$$

Proof. Let $A = \{\alpha \in \Sigma^n \mid X_n(\alpha) \geq Y_n(\alpha)\}$. It is easy to see that A is the set for which $|\text{Prob}_{X_n}(A) - \text{Prob}_{Y_n}(A)|$ is maximum. Then

$$\begin{aligned} \Delta_{\text{stat}}(X_n, Y_n) &= \sum_{\alpha \in \Sigma^n} |X_n(\alpha) - Y_n(\alpha)| \\ &= \sum_{\alpha \in A} (X_n(\alpha) - Y_n(\alpha)) + \sum_{\alpha \in \Sigma^n - A} (Y_n(\alpha) - X_n(\alpha)) \\ &= \text{Prob}_{X_n}(A) - \text{Prob}_{Y_n}(A) + (1 - \text{Prob}_{Y_n}(A)) - (1 - \text{Prob}_{X_n}(A)) \\ &= 2|\text{Prob}_{X_n}(A) - \text{Prob}_{Y_n}(A)| = 2 \max_A |\text{Prob}_{X_n}(A) - \text{Prob}_{Y_n}(A)|, \end{aligned}$$

and the lemma is proved. \blacksquare

The statistical distance has the standard properties of a distance.

Lemma 5.1.9 *Let $n \in \mathbb{N}$, and let X_n, Y_n and Z_n be distributions over Σ^n . Then*

- (1) $\Delta_{\text{stat}}(X_n, Y_n) = 0 \Leftrightarrow X_n = Y_n$,
- (2) $\Delta_{\text{stat}}(X_n, Y_n) = \Delta_{\text{stat}}(Y_n, X_n)$,
- (3) $\Delta_{\text{stat}}(X_n, Z_n) \leq \Delta_{\text{stat}}(X_n, Y_n) + \Delta_{\text{stat}}(Y_n, Z_n)$ (*triangle inequality*).

Proof. All three properties follow immediately from the definition of statistical distance. \blacksquare

Let us consider now a function f that maps short strings into longer strings, as a pseudo-random generator is supposed to do. For concreteness, let us suppose that f maps binary strings of length n (i.e., Σ^n) into binary strings of length $2n$ (i.e., Σ^{2n}). The function f induces naturally a distribution X_{2n} on Σ^{2n} defined by $X_{2n}(\alpha) = \text{Prob}_{x \in \Sigma^n}(f(x) = \alpha)$. We would like X_{2n} to be statistically close to U_{2n} . Unfortunately, this is not possible because if we take the test A to be the image of f , then $\text{Prob}_{X_{2n}}(A) = 1$ and $\text{Prob}_{U_{2n}}(A) = \frac{2^n}{2^{2n}} = \frac{1}{2^n}$, and thus, according to Lemma 5.1.8, $\Delta_{\text{stat}}(X_{2n}, U_{2n}) \geq 2(1 - \frac{1}{2^n})$. Consequently, the distributions X_{2n} and U_{2n} are statistically far apart. However, it is possible that any test that is able to distinguish the two distributions, such as the above set A , is very complex and, in particular, beyond the capabilities of an adversary. In other words, it may happen that for every subset $A \subseteq \Sigma^{2n}$ that is computable by a circuit of size $S(n)$, $\text{Prob}_{X_{2n}}(A) \approx \text{Prob}_{U_{2n}}(A)$. In this case, the distributions X_{2n} and U_{2n} do look similar to an adversary that is endowed with computational power $S(n)$. This justifies the following definition.

Definition 5.1.10 (Computational distance between two distributions) *Let $n, S \in \mathbb{N}$. Let X_n, Y_n be two distributions on Σ^n . The computational distance between X_n and Y_n relative to size S is denoted $\Delta_{\text{comp}, S}(X_n, Y_n)$ and is defined by*

$$\Delta_{\text{comp}, S}(X_n, Y_n) = \max |\text{Prob}(C(X_n) = 1) - \text{Prob}(C(Y_n) = 1)|,$$

where the maximum is taken over all circuits C with inputs of size n and having size at most S .

Definition 5.1.11 Let $n, S \in \mathbb{N}$ and $\epsilon > 0$. Let X_n, Y_n be two distributions on Σ^n . We say that the distributions X_n and Y_n are computationally ϵ -close relative to size S if $\Delta_{\text{comp}, S}(X_n, Y_n) \leq \epsilon$.

Does the computational distance retain the three properties listed in Lemma 5.1.9? We will see in Proposition 5.1.17 that property (1) fails in a dramatic way: There are distributions that are far apart in the statistical sense, but very close in the computational sense. In fact, it is this failure that allows the very notion of a pseudo-random generator. On the other hand, properties (2) and (3) remain valid for the computational distance.

Lemma 5.1.12 Let $n, S \in \mathbb{N}$, and let X_n, Y_n and Z_n be distributions over Σ^n . Then

- (1) $\Delta_{\text{comp}, S}(X_n, Y_n) = \Delta_{\text{comp}, S}(Y_n, X_n)$,
- (2) $\Delta_{\text{comp}, S}(X_n, Z_n) \leq \Delta_{\text{comp}, S}(X_n, Y_n) + \Delta_{\text{comp}, S}(Y_n, Z_n)$ (triangle inequality).

Proof. The two properties follow immediately from the definition of computational distance. \blacksquare

Finally, we can define formally the notion of a pseudo-random generator.

Definition 5.1.13 (Pseudo-random generator) Let $\ell, L, S \in \mathbb{N}$ and $\epsilon > 0$ be parameters. A length-regular function $g: \Sigma^\ell \rightarrow \Sigma^L$ is a pseudo-random generator with security (ϵ, S) if

$$\Delta_{\text{comp}, S}(g(U_\ell), U_L) \leq \epsilon.$$

The value $(L - \ell)$ is called the extension of g .

In other words, by unwrapping all these definitions, $g: \Sigma^\ell \rightarrow \Sigma^L$ is a pseudo-random generator with security (ϵ, S) if for every circuit C on inputs of length L and with $\text{size}(C) \leq S$,

$$|\text{Prob}_{x \in \Sigma^\ell}(C(g(x)) = 1) - \text{Prob}_{y \in \Sigma^L}(C(y) = 1)| \leq \epsilon.$$

In general, we are interested in producing pseudo-random generators for many input lengths ℓ (ideally for any input length $\ell \in \mathbb{N}$).

Definition 5.1.14 (Ensemble of pseudo-random generators) Let $\epsilon: \mathbb{N} \rightarrow [0, 1]$ and $S: \mathbb{N} \rightarrow \mathbb{N}$ be two functions. An ensemble of pseudo-random generators with security (ϵ, S) is a family of functions $(g_\ell)_{\ell \in \mathbb{N}}$ such that

- (1) for some function $L: \mathbb{N} \rightarrow \mathbb{N}$, for all $\ell \in \mathbb{N}$, $g_\ell: \Sigma^\ell \rightarrow \Sigma^{L(\ell)}$, and
- (2) for each $\ell \in \mathbb{N}$, g_ℓ is a pseudo-random generator with security $(\epsilon(\ell), S(\ell))$.

Abusing terminology, when the context is clear, an ensemble of pseudo-random generators is called a pseudo-random generator itself.

Depending on the parameters ϵ and S and similarly to the taxonomy of one-way functions that we have introduced earlier, we distinguish two types of good pseudo-random generators.

Definition 5.1.15 (Strong pseudo-random generator) *An ensemble of pseudo-random generators $(g_\ell)_{\ell \in \mathbb{N}}$ with security (ϵ, S) is called strong if $1/\epsilon$ and S are both superpolynomial functions.*

Definition 5.1.16 (Exponentially strong pseudo-random generator) *An ensemble of pseudo-random generators $(g_\ell)_{\ell \in \mathbb{N}}$ with security (ϵ, S) is called exponentially strong if there is a positive constant c such that, for almost every ℓ , $1/\epsilon(\ell) > 2^{c\ell}$ and $S(\ell) > 2^{c\ell}$.*

A first observation is that good pseudo-random generators exist. We will show this for generators of the type $g_\ell: \Sigma^\ell \rightarrow \Sigma^{2\ell}$, but, from the demonstration, it will be clear that the assertion can be made more general.

Proposition 5.1.17 *There exists an ensemble of functions $(g_\ell)_{\ell \in \mathbb{N}}$, of type $g_\ell: \Sigma^\ell \rightarrow \Sigma^{2\ell}$, for all sufficiently large $\ell \in \mathbb{N}$, that is an exponentially strong pseudo-random generator.*

Proof. Let us fix $\ell \in \mathbb{N}$. We pick a function g_ℓ at random among the functions mapping strings of length ℓ into strings of length 2ℓ , and we show that, if ℓ is sufficiently large, the probability that there is a function having the property asserted in the statement is positive. It follows that such a function g_ℓ exists.

Thus, for each $x \in \Sigma^\ell$, $g_\ell(x)$ is defined to be a string in $\Sigma^{2\ell}$ picked uniformly at random. Let C be a fixed circuit on inputs in $\Sigma^{2\ell}$ and let $p = \text{Prob}_{x \in \Sigma^{2\ell}}(C(x) = 1)$. We enumerate Σ^ℓ as $\{\alpha_1, \dots, \alpha_{2^\ell}\}$, and, for each $i \in \{1, \dots, 2^\ell\}$, we define the random variable X_i to be 1 if $C(g(\alpha_i)) = 1$ and 0 if $C(g(\alpha_i)) = 0$. The random variables X_i , $i \in \{1, \dots, 2^\ell\}$, are independent and the expected value of each of them is p . Therefore, by the additive Chernoff bounds (see Appendix A),

$$\text{Prob}_{g_\ell} \left(\left| \frac{1}{2^\ell} \sum X_i - p \right| \geq 2^{-\ell/4} \right) \leq 2e^{-(1/3)(2^{-\ell/4})^2 \cdot 2^\ell} < \frac{1}{2^{(1/3)2^{\ell/2}}}.$$

Since $|\frac{1}{2^\ell} \sum X_i - p|$ is $|\text{Prob}_{\alpha \in \Sigma^\ell}(C(g(\alpha)) = 1) - \text{Prob}(C(x) = 1)|$, it follows that, for a fixed circuit C , the latter value is $\geq 2^{-\ell/4}$ with probability of g_ℓ less than $2^{-(1/3)2^{\ell/2}}$.

The number of circuits C with $\text{size}(C) \leq 2^{\ell/4}$ is bounded by $2^{O(\ell \cdot 2^{\ell/4})}$ (this is shown in Section 1.1.2). Therefore the probability of the event “There exists some circuit C of size at most $2^{\ell/4}$ such that $|\text{Prob}_{\alpha \in \Sigma^\ell}(C(g(\alpha)) = 1) - \text{Prob}(C(x) = 1)|$ is $\geq 2^{-\ell/4}$.” is less than $2^{O(\ell \cdot 2^{\ell/4})} \cdot 2^{-(1/3)2^{\ell/2}} < 1$. Thus, the probability of the complementary event is positive, from which, as we have discussed, the conclusion follows. ■

The foregoing proof is non-constructive and, therefore the result has only theoretical value. Even the theoretical merit is quite limited because what we need are pseudo-random generators that are efficiently computable. For example, it would be desirable that the pseudo-random generator $g_\ell: \Sigma^\ell \rightarrow \Sigma^{2\ell}$, whose existence is asserted above, is computable in polynomial time. Unfortunately, the above proof

does not say anything about the complexity of g_ℓ . In fact, proving the existence of an efficiently computable pseudo-random generator is beyond the current state of complexity theory. Indeed, an efficiently computable pseudo-random generator is also a one-way function (the existence of which, as we have argued earlier, implies $P \neq NP$). To keep the notation simple, we prove the foregoing assertion for a particular setting of some of the parameters.

Proposition 5.1.18 INFORMAL STATEMENT: *An efficiently computable pseudo-random generator is a one-way function.*

FORMAL STATEMENT: *Suppose there exists an ensemble of functions $(g_\ell)_{\ell \in \mathbb{N}}$ with the following properties:*

- (1) *For some functions $\epsilon: \mathbb{N} \rightarrow [0, 1]$ and $S: \mathbb{N} \rightarrow \mathbb{N}$, the ensemble $(g_\ell)_{\ell \in \mathbb{N}}$ is a pseudo-random generator with security (ϵ, S) ,*
- (2) *for all $\ell \in \mathbb{N}$, $g_\ell: \Sigma^\ell \rightarrow \Sigma^{2\ell}$,*
- (3) *there is a polynomial q such that, for all $\ell \in \mathbb{N}$, g_ℓ is computable in time $q(\ell)$.*

Then the function $g: \Sigma^ \rightarrow \Sigma^*$ induced by the ensemble $(g_\ell)_{\ell \in \mathbb{N}}$ is a one-way function with security $(\epsilon + \frac{1}{2^\ell}, S - p(\ell))$, for some polynomial p .*

Proof. Let us fix $\ell \in \mathbb{N}$ sufficiently large (so that the following arguments are valid). Suppose there is a circuit C_ℓ that inverts g_ℓ with probability at least $\epsilon(\ell) + \frac{1}{2^\ell}$, i.e.,

$$\text{Prob}_{x \in \Sigma^\ell} (C_\ell(g_\ell(x)) \in g_\ell^{-1}(g_\ell(x))) \geq \epsilon(\ell) + \frac{1}{2^\ell}. \quad (5.1)$$

We define $A = \{y \in \Sigma^{2\ell} \mid g_\ell(C_\ell(y)) = y\}$. Let $C_{\ell,A}$ be a circuit that calculates A (i.e., $C_{\ell,A}(y) = 1$ if and only if $y \in A$). There is a polynomial p , such that for all ℓ , $C_{\ell,A}$ can be taken with $\text{size}(C_{\ell,A}) \leq \text{size}(C_\ell) + p(\ell)$. Let us assume that $\text{size}(C_\ell) \leq S(\ell) - p(\ell)$. Thus, $\text{size}(C_{\ell,A}) \leq S(\ell)$. Since A is a subset of the image of g_ℓ , it follows that $\|A\| \leq \|\Sigma^\ell\| = 2^\ell$. Therefore,

$$\text{Prob}_{y \in \Sigma^{2\ell}} (C_{\ell,A}(y) = 1) \leq \frac{2^\ell}{2^{2\ell}} = \frac{1}{2^\ell}.$$

On the other hand,

$$\text{Prob}_{x \in \Sigma^\ell} (C_{\ell,A}(g_\ell(x)) = 1) \geq \epsilon(\ell) + \frac{1}{2^\ell}.$$

It follows that

$$\text{Prob}_{x \in \Sigma^\ell} (C_{\ell,A}(g_\ell(x)) = 1) - \text{Prob}_{y \in \Sigma^{2\ell}} (C_{\ell,A}(y) = 1) \geq \epsilon(\ell).$$

Since $\text{size}(C_{\ell,A}) \leq S(\ell)$, this contradicts the fact that g_ℓ is a pseudo-random generator with security $(\epsilon(\ell), S(\ell))$. Thus, the relation (5.1) is false. \blacksquare

On the other hand, a one-way function f is not necessarily a pseudo-random generator. Indeed, suppose a length-regular function $f: \Sigma^* \rightarrow \Sigma^*$, with

$f_\ell: \Sigma^\ell \rightarrow \Sigma^{2\ell}$, for all $\ell \in \mathbb{N}$ (the extension $\ell \rightarrow 2\ell$ has been taken arbitrarily), is one-way with security (ϵ, S) , for some functions $\epsilon: \mathbb{N} \rightarrow [0, 1]$ and $S: \mathbb{N} \rightarrow \mathbb{N}$. Consider the functions $g_\ell: \Sigma^\ell \rightarrow \Sigma^{2\ell+1}$ defined, for all $\ell \in \mathbb{N}$, by $g_\ell(x) = 0 \odot f_\ell(x)$. Then, it is easy to see that the ensemble of functions $(g_\ell)_{\ell \in \mathbb{N}}$ continues to be one-way. However, $g_\ell(x)$ does not look random at all since it always (i.e., for all ℓ and for all x) starts with 0. In particular, a small circuit that accepts an input string if and only if it starts with 0 will distinguish the distribution $g_\ell(U_\ell)$ from $U_{2\ell+1}$ and, therefore, g_ℓ is not a pseudo-random generator.

This example and Proposition 5.1.18 suggest that the requirements in the definition of a pseudo-random generator are much more exacting than what a one-way function provides. In spite of this, a remarkable result of Håstad, Impagliazzo, Levin, and Luby [HILL99b] (building on the work of many other researchers) shows that, given a strong one-way function f , one can construct a polynomial-time computable strong pseudo-random generator. The proof of this result is extremely complex and beyond the scope of this book. We prove in this chapter a weaker result by assuming that f in addition to being a strong one-way function is also a permutation at each length (i.e., for each ℓ , $f_\ell: \Sigma^\ell \rightarrow \Sigma^\ell$ is a bijection). We show that given such a function f one can construct a polynomial-time computable strong pseudo-random generator with polynomial extension (superpolynomial extension is discussed below).

This construction, as well as many other ones in this chapter, follows a pattern: Given one function f_1 with certain properties, there is an effective procedure that computes some other function f_2 . This concept is formalized in the following definition.

Definition 5.1.19 *Let $f_1: \Sigma^* \rightarrow \Sigma^*$ and $f_2: \Sigma^* \rightarrow \Sigma^*$. We say that f_2 is effectively computed from f_1 if there is an algorithm A that (a) has oracle access to the function f_1 , and (b) on input x , calculates $f_2(x)$. The function f_1 is called the building block of the construction. In case the algorithm A runs in time $t(|x|)$ on input x , for all $x \in \Sigma^*$, we say that f_2 is effectively computed from f_1 in time $t(\cdot)$.*

The transformation of the one-way function f into a pseudo-random generator is done in two steps. In Section 5.3, using the function f , we build a polynomial-time computable pseudo-random generator that has an extension of just one bit. The second step is done in Section 5.4, where the extension is enlarged to more significant values. How large an extension can one achieve depends on the quality (i.e., the security parameters ϵ and S) of the initial one-way permutation. In particular, if the one-way permutation is strong or exponentially strong, then the extension can be made superpolynomial or, respectively, exponential. Obviously, if the pseudo-random generator has superpolynomial extension then it cannot be computed in polynomial time. Therefore, it is desirable that each bit of the randomly looking output can be computed in polynomial time independently of the other bits of the input. Such an object can be viewed as a function which on input an index i returns the i -th bit of the string produced by the pseudo-random generator and indeed it is called a *pseudo-random function*. In Section 5.5, we

show how to build a pseudo-random function given, as a building block, a strong pseudo-random generator $(g_\ell)_{\ell \in \mathbb{N}}$ that doubles the length of the seed (i.e., for each ℓ , $g_\ell: \Sigma^\ell \rightarrow \Sigma^{2\ell}$).

We have considered efficiently computable pseudo-random generators and it seems natural to require that the output produced by such generators should look random to an adversary that is endowed with large computational resources, in particular, with computational resources that are superior to the ones needed to calculate the pseudo-random generator. Indeed our discussion so far has concentrated on pseudo-random generators with security (ϵ, S) that are computable in time significantly shorter than S . Let us call these type I pseudo-random generators. Håstad, Impagliazzo, Levin, and Luby have shown that such pseudo-random generators can be constructed given a one-way function (we only present the construction that uses a one-way permutation). Let us relax the efficiency requirement of a pseudo-random generator and allow it to be computable in time that is larger than the security parameter S . Let us call this a type II pseudo-random generator. Can we construct such pseudo-random generators under a relaxed assumption regarding the building block function? The answer is positive. Indeed, we show in Section 5.8 that if we use as a building block a hard function (in fact a hard predicate) then we can build type II pseudo-random generators. This is an interesting result for two reasons. First, hard functions do exist (even though it is not known how to actually obtain such hard functions), while the assumption needed for constructing type I pseudo-random generators, namely the existence of one-way functions, is only a conjecture (which, furthermore, implies $P \neq NP$). Secondly, type II pseudo-random generators, under some conditions, can be utilized to derandomize any polynomial-time probabilistic computation with bounded 2-sided error (this is usually called a BPP computation). More precisely, the alluded conditions require that (1) the pseudo-random generator has exponential extension, (2) the pseudo-random generator is computable in time polynomial in the output length, and (3) the pseudo-random generator is secure against adversaries that can spend time that is a fixed polynomial in the output length. In Section 5.9, we observe (using type II pseudo-random generators) that if these requirements are met, then $P = BPP$! We also show that, under a quite reasonable hypothesis, the above requirements are in fact satisfied. The hypothesis is that there exists a length-regular function $f: \Sigma^* \rightarrow \Sigma^*$ and two constants c_1 and c_2 such that (1) f is computable in time $2^{c_1 n}$, and (2) for almost every length n , no circuit of size $2^{c_2 n}$ can calculate f_n .

Since the construction of type II pseudo-random generators relies on hard functions, we need to clarify what we mean when we say that a function f is hard. The intent is to capture the idea that no adversary having some specified computational power can calculate f . The most basic definition of a hard function deals with functions defined for inputs of some fixed length.

Definition 5.1.20 (Hard function) *Let $\epsilon > 0$, $\ell \in \mathbb{N}$, $\ell' \in \mathbb{N}$ and $S \in \mathbb{N}$ be parameters. A function $f: \Sigma^\ell \rightarrow \Sigma^{\ell'}$ is (ϵ, S) -hard if for every circuit C of size S ,*

$$\|\{x \in \Sigma^\ell \mid C(x) \neq f(x)\}\| \geq \epsilon \cdot 2^\ell.$$

We move to functions that are defined at all lengths. In this case, the adversary is represented by a collection of deterministic circuits $(C_\ell)_{\ell \in \mathbb{N}}$, where each C_ℓ calculates a function whose domain is Σ^ℓ . Abusing notation, we use C_ℓ to also denote the function computed by the circuit C_ℓ .

As we have proceeded earlier, to prevent the trivial and uninteresting case of functions being hard simply because their output is too long, we will consider only length-regular hard functions $f: \Sigma^* \rightarrow \Sigma^*$ with $f_\ell: \Sigma^\ell \rightarrow \Sigma^{\ell'(\ell)}$ for which there is a polynomial p such that, for all ℓ , $\ell'(\ell) \leq p(\ell)$.

Also, as in the case of one-way functions and pseudo-random generators, we mainly consider adversaries endowed with circuits whose size is bounded by some superpolynomial function. Intuitively, f is hard for an adversary represented by a collection of circuits $(C_\ell)_{\ell \in \mathbb{N}}$ if, for all sufficiently large ℓ , C_ℓ fails to calculate f_ℓ . The failure can be more or less severe and correspondingly we have different degrees of hardness for a function.

Definition 5.1.21 (Worst-case hard function) *A length-regular function $f: \Sigma^* \rightarrow \Sigma^*$ is worst-case hard if there is a superpolynomial function S so that the following holds:*

For any family of circuits $(C_\ell)_{\ell \in \mathbb{N}}$ of size at most S ,

$$\|\{x \in \Sigma^\ell \mid C_\ell(x) \neq f_\ell(x)\}\| \geq 1,$$

for all sufficiently large ℓ .

Definition 5.1.22 (Constant-rate hard function) *Let k be a positive integer. A length-regular function $f: \Sigma^* \rightarrow \Sigma^*$ is k -constant-rate hard if there is a superpolynomial function S so that the following holds:*

For any family of circuits $(C_\ell)_{\ell \in \mathbb{N}}$ of size at most S ,

$$\|\{x \in \Sigma^\ell \mid C_\ell(x) \neq f_\ell(x)\}\| \geq \frac{1}{k} \cdot 2^\ell,$$

for all sufficiently large ℓ .

Definition 5.1.23 (Crypto hard function) *A length-regular function $f: \Sigma^* \rightarrow \Sigma^*$ is cryptographically hard (or, in short, crypto-hard) if there is a superpolynomial function S so that the following holds:*

For any polynomial p and for any family of circuits $(C_\ell)_{\ell \in \mathbb{N}}$ of size at most S ,

$$\|\{x \in \Sigma^\ell \mid C_\ell(x) \neq f_\ell(x)\}\| \geq \left(1 - \frac{1}{p(\ell)}\right) \cdot 2^\ell,$$

for all sufficiently large ℓ .

Definition 5.1.24 (Exponentially hard function) *A length-regular function $f: \Sigma^* \rightarrow \Sigma^*$ is exponentially hard if there is a constant $c > 0$ so that the following holds:*

For any family of circuits $(C_\ell)_{\ell \in \mathbb{N}}$ of size at most $2^{c\ell}$,

$$\|\{x \in \Sigma^\ell \mid C_\ell(x) \neq f_\ell(x)\}\| \geq (1 - 2^{-cl}) \cdot 2^\ell,$$

for all sufficiently large ℓ .

Of course, we can use probability to express the relations in Definition 5.1.21, Definition 5.1.22, and Definition 5.1.23. For example, in Definition 5.1.22, we can say

$$\text{Prob}(C_\ell(x) \neq f_\ell(x)) \geq \frac{1}{k}, \quad (5.2)$$

where the probability is taken over x chosen uniformly at random in Σ^ℓ . This formulation has the advantage that it can be extended easily to probabilistic algorithms. For instance, we can require that, for all probabilistic circuits of size at most S , the probability in (5.2) holds over x chosen uniformly at random in Σ^ℓ and over the random choices r made by the circuit. Note that here the fact that the adversary can utilize probabilistic algorithms does not give him much additional power. Indeed, if there is a probabilistic algorithm C_ℓ so that,

$$\text{Prob}_{x,r}(C_\ell(x,r) \neq f_\ell(x)) \geq \frac{1}{k},$$

(where x denotes the input and r denotes the random bits), then there has to be one fixed r_0 so that

$$\text{Prob}_x(C_\ell(x,r_0) \neq f_\ell(x)) \geq \frac{1}{k}.$$

By embedding r_0 into the circuit C_ℓ , this becomes deterministic and its size increases by only $|r_0|$ additional gates (needed to store r_0). Keeping in mind this observation, we restrict our attention to the case of adversary deterministic circuits.

Clearly, the property “ f is crypto hard” is much stronger than “ f is worst-case hard,” with “ f is constant-rate hard” falling in the middle.

Interestingly, hardness can be amplified, and, furthermore, the amplification can be accomplished effectively. Indeed, we show in Section 5.6 that using a worst-case function as a building block, one can construct a crypto hard function. The construction has two phases. First, we build a constant hard function from a worst-case hard function, and, in the second phase, a constant hard function is used to produce a crypto hard function.

In Section 5.7, we consider hard predicates, i.e., functions f of the form $f: \Sigma^* \rightarrow \{0, 1\}$. Since the range of a predicate has only two elements, there always is a small circuit that calculates a predicate on at least half of the inputs in Σ^n . Therefore, a predicate is considered hard if no circuit of some respectable size can calculate it on a fraction of inputs in Σ^n that is significantly larger (by, say,

$1/\text{poly}(n)$) than $1/2$ (see Definition 5.7.1). We present an effective construction of a hard predicate using a crypto hard function as a building block.

Section 5.10 is dedicated to extractors. An extractor is a function that resembles a pseudo-random generator in that its output has to pass some randomness tests. It is used to remedy sources of randomness that are in a sense weak. More precisely, suppose that there exists a device that generates random binary strings of length n . Ideally, to obtain perfect randomness, each string should be generated with probability 2^{-n} . Suppose instead that each string is generated with probability at most 2^{-k} , for some $k < n$ (k is a parameter, called min-entropy, that together with n characterizes the source, which in this case is called a (n, k) -weak source). Intuitively, the strings generated have k bits of randomness (out of the total length of n). An extractor is a function that depends on five parameters n, k, d, m , and ϵ : It takes as input a string produced by a (n, k) -weak source and a perfectly random but short additional string of length d , and produces a string of length m such that the distribution of the output is at a statistical distance at most ϵ from the uniform distribution on $\{0, 1\}^m$. We show that the construction, given in Section 5.8, of a type II pseudo-random generator using as a starting primitive a hard predicate f , can also be used to build an extractor function computable in polynomial time. The key idea is to view the truth-table of f as simply being a string produced by a weak source with min-entropy k and carry on the construction from Section 5.8. In the construction of a pseudo-random generator, we argue that if the output does not have a distribution within a short computational distance from the uniform distribution, then the starting predicate f is not hard. A similar argument works in the case of extractors: If the output distribution is not within a short statistical distance from the uniform distribution, then the starting truth-table belongs to a small family of strings which contains the set of strings generated by the weak source. This contradicts the fact that the min-entropy of the source is at least k (which implies that the number of generated strings is at least 2^k). The method leads to the construction of a polynomial-time computable extractor that can remedy $(n, \gamma n)$ -weak sources, for arbitrarily small constant γ , using an additional number of random bits $d = O(\log n)$ and with output length $m = k^{1-\alpha}$, for arbitrarily small positive α (for simplicity, the given proof obtains a shorter output length).

From our brief overview, it should be clear that most of the major results that are presented in this chapter are of the following form: Using a function f_1 of type T_1 as a building block, one can effectively construct a function f_2 of type T_2 (where type T_2 functions satisfy more demanding requirements than type T_1 functions). Table 5.1 gives a “road map” for these results.

Table 5.1: Summary of effective constructions. Results are of the type: Using f_1 as a building block, there is an effective construction of f_2 . (Note: An extender is a pseudo-random generator with extension 1.)

building block f_1	result f_2	where
weak one-way function	strong one-way function	Theorem 5.2.1 Section 5.2
one-way permutation	extender	Theorem 5.3.11 Section 5.3
extender	type I pseudo-random generator	Theorem 5.4.1 Section 5.4
type I pseudo-random generator	pseudo-random function	Theorem 5.5.1 Section 5.5
worst-case hard function	const.- rate hard function	Theorem 5.6.3(a) Section 5.6
$(\frac{1}{2^n}, 2^{cn})$ -hard function	$(\text{const.}, 2^{cn})$ -hard function	Theorem 5.6.3(b) Section 5.6
const.- rate hard function	crypto hard function	Theorem 5.6.12 (a) Section 5.6
$(\text{const.}, 2^{cn})$ -hard function	exp. hard function	Theorem 5.6.12 (b) Section 5.6 (no proof)
crypto hard function	crypto hard predicate	Theorem 5.7.4 Section 5.7
exp. hard function	exp. hard predicate	Theorem 5.7.4 Section 5.7
exp. hard predicate	type II pseudo-random generator	Corollary 5.8.7 Section 5.8

5.2 From weak to strong one-way functions

IN BRIEF: The “hard-to-invert” property of a one-way function can be amplified from a fraction of $1/\text{poly}(n)$ of the inputs to the fraction $(1 - (1/\text{superpoly}(n)))$ of the inputs.

Weak one-way functions and strong one-way functions have been defined in Section 5.1. Intuitively, in the case of a weak one-way function, any polynomial-time algorithm fails to invert at least a $(1/\text{poly}(n))$ fraction of the multiset $\{f(x) \mid x \in \Sigma^n\}$. It may very well be the case that some polynomial-time algorithm inverts a large majority of inputs. In the case of a strong one-way

function, any polynomial-time algorithm fails much more drastically, i.e., it fails on a $(1 - 1/\text{superpoly}(n))$ fraction of the multiset $\{f(x) \mid x \in \Sigma^n\}$. Obviously, if we need a one-way function, it better be a strong one. Note that the common candidates for one-way functions that have been considered are not strong. For example, let us take the case of the integer factoring problem. This problem offers a good basis for building a one-way function because, given two integers p and q , it is easy to calculate $n = p \cdot q$ and, on the other hand, there is no known polynomial-time algorithm for inverting the process, i.e., to find p and q given n . However, half of the integers n are even and for them it is very easy to find a prime factor (namely 2). Therefore factoring certainly does not provide a strong one-way function.

Thus, while the existence of weak one-way functions looks plausible, there seems to be little direct evidence in support of strong one-way function. The “hard-to-invert” requirement in the definition of a strong one-way function appears to be much more demanding than in the case of a weak one-way function. Perhaps surprisingly, in fact, strong one-way functions exist if and only if weak one-way functions exist. Since a strong one-way function obviously is also a weak one-way function, we only need to prove the following theorem.

Theorem 5.2.1 INFORMAL STATEMENT: *If weak one-way functions exist, then strong one-way functions exist.*

FORMAL STATEMENT: *Assume $f: \Sigma^* \rightarrow \Sigma^*$ is a weak one-way function. Then there is $g: \Sigma^* \rightarrow \Sigma^*$ a strong one-way function. Moreover, g is effectively computable from f in polynomial time.*

Proof. The assumption formally means that there is a polynomial q such that, for almost every n , for any polynomial p , and for any circuit A on inputs of length n with $\text{size}(A) \leq p(n)$,

$$\text{Prob}_{x \in \Sigma^n}(A(f(x)) \notin f^{-1}(f(x))) \geq \frac{1}{q(n)}. \quad (5.3)$$

Let us fix a sufficiently large n for which the assumption holds and for which the following arguments are valid. We focus on the restriction of f on Σ^n . Let $m = n \cdot q(n)$ and consider the function $g_{m,n}: \Sigma^{m,n} \rightarrow \Sigma^*$ defined by

$$g_{m,n}(x_1 \odot \dots \odot x_m) = f(x_1) \odot \dots \odot f(x_m),$$

where each substring x_i , $i = 1, \dots, m$, of the input has length n . In other words, $g_{m,n}$ is the concatenation of m independent copies of f . The function $g_{m,n}$ is still computable in polynomial time and, intuitively, it is harder to invert than f because one has to invert each “chunk” $f(x_i)$, $i = 1, \dots, m$. For example, if we indeed follow the strategy of finding the inverse of each $f(x_i)$ one at a time, the probability of success in inverting $g_{m,n}(x_1 \odot \dots \odot x_m)$ is at most $(1 - (1/q(n)))^m = (1 - (1/q(n)))^{nq(n)} < e^{-n}$. We will show that no other strategy inverts $g_{m,n}$ significantly better than this simple one.

The proof is by contradiction. Thus we assume that there is a polynomial p and a circuit B (running the presumed better inverting strategy) on inputs of size $m \cdot n$, with $\text{size}(B) \leq p(m \cdot n)$ (which is polynomial in n) such that

$$\begin{aligned} \text{Prob}(B(g_{m \cdot n}(x_1 \odot \dots \odot x_m)) \in g_{m \cdot n}^{-1}(g_{m \cdot n}(x_1 \odot \dots \odot x_m))) &\geq \frac{1}{p(m \cdot n)} \quad (5.4) \\ &> e^{-n} + \frac{1}{2p(m \cdot n)}. \end{aligned}$$

Using B , we construct a polynomial-time probabilistic algorithm D that successfully inverts $f(x)$ on more than a fraction of $(1 - (1/q(n)))$ of the strings x in Σ^n . This essentially contradicts the hypothesis (5.3) (the only small problem being that D is probabilistic).

The algorithm D proceeds as follows.

Input: $y = f(x)$, for some $x \in \Sigma^n$.

Repeat the following $2n \cdot m \cdot p(n \cdot m) = 2n^2 \cdot q(n) \cdot p(n^2 \cdot q(n))$ times.

Pick random $i \in \{1, \dots, m\}$.

Pick $m - 1$ random strings in Σ^n denoted $x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_m$.

Calculate $Y = f(x_1) \odot \dots \odot f(x_{i-1}) \odot f(x) \odot f(x_{i+1}) \odot \dots \odot f(x_m)$.

Call the circuit B to invert Y . The result is $x'_1, \dots, x'_i, \dots, x'_m$.

If $f(x'_i) = y$, then the algorithm outputs x'_i , reports SUCCESS, and stops.

End Repeat

If there was no SUCCESS, output arbitrarily 0^n .

We next estimate the success probability of the above algorithm. Let $\text{INV} \subseteq \Sigma^{n \cdot m}$ be the elements of $\Sigma^{n \cdot m}$ that B inverts. By our assumption,

$$\|\text{INV}\| \geq \left(e^{-n} + \frac{1}{2p(n \cdot m)} \right) \cdot \|\Sigma^{n \cdot m}\|.$$

For each $x \in \Sigma^n$, let $N(x)$ be the multiset consisting of m -tuples (x_1, \dots, x_m) , with each $x_i \in \Sigma^n$, such that for some $i \in \{1, \dots, m\}$, $x = x_i$. The multiplicity of such an m -tuple is the number of entries that are equal to x . Note that, for each $x \in \Sigma^n$, $\|N(x)\| = m \cdot 2^{n(m-1)}$. For any set $T \subseteq \Sigma^n$, $N(T)$ denotes $\bigcup_{x \in T} N(x)$.

It is useful to view the algorithm D from a different angle. On input $f(x)$, the algorithm calculates $Y = f(x_1) \odot \dots \odot f(x) \odot \dots \odot f(x_m)$, where $X = (x_1, \dots, x_m)$ is uniformly at random chosen in $N(x)$. The circuit B is next invoked to invert Y . If $X \in \text{INV}$ then the algorithm D is successful. Therefore, we would like to show that for a large fraction (more precisely for a fraction of $(1 - \frac{1}{q(n)})$) of x in Σ^n , $N(x) \cap \text{INV}$ is a large set, so that the success probability of D is large in inverting the input $f(x)$. This will lead to the desired contradiction of relation (5.3).

Claim 5.2.2 Let $V = \left\{ x \in \Sigma^n \mid \frac{\|N(x) \cap \text{INV}\|}{\|N(x)\|} \geq \frac{1}{2 \cdot p(n \cdot m) \cdot m} \right\}$. Then $\|V\| > \left(1 - \frac{1}{q(n)}\right) \cdot \|\Sigma^n\|$.

Proof. The complementary set of V is

$$\bar{V} = \left\{ x \in \Sigma^n \mid \frac{\|N(x) \cap \text{INV}\|}{\|N(x)\|} < \frac{1}{2 \cdot p(n \cdot m) \cdot m} \right\}.$$

Thus, for each $x \in \bar{V}$, $\|N(x) \cap \text{INV}\| < \frac{1}{2 \cdot p(n \cdot m) \cdot m} \cdot (m \cdot 2^{n(m-1)})$, and consequently,

$$\begin{aligned} \|N(\bar{V}) \cap \text{INV}\| &\leq \sum_{x \in \bar{V}} \|N(x) \cap \text{INV}\| \\ &< 2^n \cdot \frac{1}{2 \cdot p(n \cdot m) \cdot m} \cdot (m \cdot 2^{n(m-1)}) \\ &= \frac{1}{2 \cdot p(n \cdot m)} \cdot \|(\Sigma^n)^m\|. \end{aligned}$$

We show that this is possible only if $\|\bar{V}\| < \frac{1}{q(n)} \cdot \|\Sigma^n\|$. To this aim, let $S \subseteq \Sigma^n$ be a set with $\|S\| \geq \frac{1}{q(n)} \cdot \|\Sigma^n\|$. The essential observation is that $N(S)$ covers an overwhelming fraction of $(\Sigma^n)^m$. Indeed, note that the probability that a tuple (x_1, \dots, x_m) is not in $N(S)$ is equal to the probability of the event “ $x_1 \notin S \wedge \dots \wedge x_m \notin S$,” which is bounded by

$$\left(1 - \frac{1}{q(n)}\right)^m = \left(1 - \frac{1}{q(n)}\right)^{n \cdot q(n)} < e^{-n}.$$

Therefore, the complementary set of $N(S)$, denoted $\overline{N(S)}$, satisfies

$$\|\overline{N(S)}\| < e^{-n} \cdot \|(\Sigma^n)^m\|.$$

Then,

$$\begin{aligned} \|N(S) \cap \text{INV}\| &= \|\text{INV}\| - \|\text{INV} \cap \overline{N(S)}\| \\ &\geq \|\text{INV}\| - \|\overline{N(S)}\| \\ &> \left(e^{-n} + \frac{1}{2p(n \cdot m)}\right) \|(\Sigma^n)^m\| - e^{-n} \cdot \|(\Sigma^n)^m\| \\ &= \frac{1}{2p(n \cdot m)} \cdot \|(\Sigma^n)^m\|. \end{aligned}$$

Thus, necessarily, $\|\bar{V}\| < \frac{1}{q(n)} \cdot \|\Sigma^n\|$ and, therefore, $\|V\| > \left(1 - \frac{1}{q(n)}\right) \cdot \|\Sigma^n\|$. ■

Claim 5.2.2 implies that if $x \in V$, then the success probability of D in inverting $f(x)$ in one iteration is at least $(1/(2m \cdot p(n \cdot m)))$. Since the algorithm D makes $n \cdot (2m \cdot p(n \cdot m))$ iterations, the overall failure probability is at most $\left(1 - \frac{1}{2m \cdot p(n \cdot m)}\right)^{n \cdot (2m \cdot p(n \cdot m))} \leq e^{-n}$. An inspection of the algorithm D reveals that

it can be implemented by a probabilistic circuit C'_D with size polynomial in n . Using a standard technique, the circuit C'_D can be converted into a *deterministic* circuit C_D with size still polynomial in n that inverts $f(x)$ for all x in V . Namely, observe that since, for any x in V , $\text{Prob}(C'_D \text{ fails to invert } x) \leq e^{-n}$, we can conclude that $\text{Prob}(C'_D \text{ fails to invert some } x \text{ in } V) \leq 2^n \cdot e^{-n} < 1$. The probabilities are taken over the random strings used by C'_D . Then there is such a string r , so that C'_D acting according to r , will successfully invert $f(x)$ for all x in V . We can embed r into the wiring of C'_D , obtaining the deterministic circuit C_D which simulates the circuit C'_D replacing the utilization of the random string with queries from r . The size of C_D increases by only $|r|$ bits and thus it is still polynomial in n .

To conclude, we have obtained a circuit (i.e., C_D) of polynomial size that inverts f on more than a fraction of $1 - \frac{1}{q(n)}$ of strings in the multiset $\{f(x) \mid x \in \Sigma^n\}$. This contradicts the relation (5.3) and, therefore, our assumption (5.4) is false. Thus, for any polynomial p and for any circuit B with $\text{size}(B) \leq p(n \cdot m) = p(n^2 \cdot q(n))$, B inverts $g_{n^2 \cdot q(n)}$ on less than a fraction $\frac{1}{p(n^2 \cdot q(n))}$ of the multiset $\{g_{n^2 \cdot q(n)}(y) \mid y \in \Sigma^{n^2 \cdot q(n)}\}$. This assertion holds for almost all n .

This is almost the desired conclusion. The only problem is that we have only obtained the ensemble of functions $\{g_{n^2 \cdot q(n)}\}_{n \in \mathbb{N}}$, which does not cover all possible input lengths (as required in the definition of a one-way function). Fortunately, it is easy to extend the ensemble with functions $g_{n'}: \Sigma^{n'} \rightarrow \Sigma^{n'}$, for n' not of the form $n^2 \cdot q(n)$. Namely, for such an n' , we find the largest n such that $n^2 \cdot q(n) \leq n'$ and define $g_{n'}$ on input x to be $g_{n^2 \cdot q(n)}$ on the prefix of x of length $n^2 \cdot q(n)$. It is easy to check that the full ensemble $(g_n)_{n \in \mathbb{N}}$ is a strong one-way function. ■

5.3 From one-way permutations to extenders

IN BRIEF: For each one-way function f , there exists a small variation f' that admits a hard-core predicate b , i.e., a predicate $b(x)$ that is hard to compute given $f'(x)$.

Given a one-way permutation, one can effectively construct a pseudo-random generator with an extension of one bit.

An *extender* is a pseudo-random generator whose output is one bit longer than its input.

Definition 5.3.1 (Extender) (a) Let $\epsilon > 0$ and $S \in \mathbb{N}$. An extender with security (ϵ, S) is a function f such that (1) for some $n \in \mathbb{N}$, $f: \Sigma^n \rightarrow \Sigma^{n+1}$, and (2) f is a pseudo-random generator with security (ϵ, S) .

(b) Let $\epsilon: \mathbb{N} \rightarrow [0, 1]$ and $S: \mathbb{N} \rightarrow \mathbb{N}$ be two functions. An ensemble of extenders with security (ϵ, S) is a family of functions $(f_n)_{n \in \mathbb{N}}$ such that, for each n , $f_n: \Sigma^n \rightarrow \Sigma^{n+1}$ and f_n is an extender with security $(\epsilon(n), S(n))$.

As before, abusing the terminology, when the context is clear, an ensemble of extenders will usually be called an extender as well.

Our goal is to build an extender starting from a one-way permutation. This is a worthy operation because we will see in the next section that, using as a building block an extender, we can obtain pseudo-random generators with more impressive extensions. The decisive step in the construction is the production of a *hard-core predicate* for the one-way function. Intuitively, a hard-core predicate $h: \Sigma^n \rightarrow \{0, 1\}$ for a function $f: \Sigma^n \rightarrow \Sigma^{n'}$ provides a bit $h(x)$ that cannot be *predicted* by an adversary that knows $f(x)$ with probability significantly larger than $1/2$ (which is what anyone can obtain by simply guessing the value of the bit without any knowledge of $f(x)$). For this reason, we also say that a hard-core predicate provides a *hidden bit* for f . We prefer however to give the formal definition of a hard-core predicate by saying that the adversary cannot *distinguish* between the distributions $f(U_n) \odot h(U_n)$ and $f(U_n) \odot U_1$. The connection between predictor and distinguisher adversaries will be established shortly in Theorem 5.3.3. The merit of the next definition is that it involves distributions that are computationally close and this is useful for our goal of building pseudo-random generators.

Definition 5.3.2 (Hard-core predicate) *The values $n, n', S \in \mathbb{N}$ and $\epsilon > 0$ are parameters. Let $f: \Sigma^n \rightarrow \Sigma^{n'}$ and $h: \Sigma^n \rightarrow \{0, 1\}$. The function h is a hard-core predicate for f with security (ϵ, S) if the distributions $(f(U_n) \odot h(U_n))$ and $(f(U_n) \odot U_1)$ are computationally ϵ -close for circuits of size S .*

The next theorem states the promised relation between predicting a hard-core predicate $h(x)$ given $f(x)$ and distinguishing between the distributions $f(U_n) \odot h(U_n)$ and $f(U_n) \odot U_1$.

Theorem 5.3.3 (Predictors vs. Distinguishers)

INFORMAL STATEMENT: *A predicate $h(x)$ can be predicted from a function $f(x)$ with probability at least $1/2 + \epsilon$ by an adversary circuit of some given size if and only if an adversary circuit of essentially the same size can distinguish between the distributions $f(U_n) \odot h(U_n)$ and $f(U_n) \odot U_1$ with bias at least ϵ .*

FORMAL STATEMENT: *Let n and n' be two integer parameters and $\epsilon > 0$. Let $f: \Sigma^n \rightarrow \Sigma^{n'}$ and $h: \Sigma^n \rightarrow \{0, 1\}$. There exists a constant c with the following properties:*

- (1) *If there is a circuit D such that $\text{Prob}_x(D(f(x)) = h(x)) \geq \frac{1}{2} + \epsilon$, then there is a circuit C of size at most $\text{size}(D) + c$ so that*

$$\text{Prob}_{x \in \Sigma^n}(C(f(x) \odot h(x)) = 1) - \text{Prob}_{x \in \Sigma^n, u \in \Sigma}(C(f(x) \odot u) = 1) \geq \epsilon.$$

- (2) *If there is a circuit C such that*

$$|\text{Prob}_{x \in \Sigma^n}(C(f(x) \odot h(x)) = 1) - \text{Prob}_{x \in \Sigma^n, u \in \Sigma}(C(f(x) \odot u) = 1)| \geq \epsilon,$$

then there is a circuit D of size at most $\text{size}(C) + c$ so that

$$\text{Prob}_x(D(f(x)) = h(x)) \geq \frac{1}{2} + \epsilon.$$

Proof. (1) The circuit C on input $f(x) \odot y$, where $f(x) \in \Sigma^n$ and $y \in \Sigma$, calculates $D(f(x))$ and outputs 1 if $D(f(x)) = y$, and 0, otherwise. Now it is immediate to check that the asserted inequality holds because $\text{Prob}_{x,u}(C(f(x) \odot u) = 1) = 1/2$ and $\text{Prob}_x(C(f(x) \odot h(x)) = 1) \geq 1/2 + \epsilon$.

(2) We can eliminate the absolute value and assume that in fact

$$\text{Prob}_x(C(f(x) \odot h(x)) = 1) - \text{Prob}_{x,u}(C(f(x) \odot u) = 1) \geq \epsilon,$$

because otherwise we can just consider the circuit that flips (i.e., negates) the output of C . Thus, the circuit C is more likely to accept the string $f(x)$ followed by $h(x)$ than the string $f(x)$ followed by a random bit u . Based on this fact, the circuit D on input $f(x)$ does the following: It chooses a random bit b and calculates $C(f(x) \odot b)$. If the result is 1, it outputs b , otherwise it outputs $1 - b$. Let E be the event that $D(f(x)) = h(x)$ when x and b are chosen uniformly at random in Σ^n and, respectively, Σ . From the description of D , we observe that

$$\begin{aligned} \text{Prob}(E) &= \text{Prob}(h(x) = b) \cdot \text{Prob}(C(f(x) \odot h(x)) = 1) \\ &\quad + \text{Prob}(h(x) = 1 - b) \cdot \text{Prob}(C(f(x) \odot (1 - h(x))) = 0). \end{aligned}$$

Let

$$\begin{aligned} p &\stackrel{\text{def}}{=} \text{Prob}_x(C(f(x) \odot h(x)) = 1), & \text{and} \\ q &\stackrel{\text{def}}{=} \text{Prob}_{x,u}(C(f(x) \odot u) = 1). \end{aligned}$$

We have

$$\begin{aligned} q &= \text{Prob}(h(x) = u) \cdot \text{Prob}(C(f(x) \odot u) = 1 \mid h(x) = u) \\ &\quad + \text{Prob}(h(x) \neq u) \cdot \text{Prob}(C(f(x) \odot u) = 1 \mid h(x) \neq u) \\ &= \frac{1}{2} \text{Prob}(C(f(x) \odot h(x)) = 1) + \frac{1}{2} \text{Prob}(C(f(x) \odot (1 - h(x))) = 1) \\ &= \frac{1}{2} p + \frac{1}{2} \text{Prob}(C(f(x) \odot (1 - h(x))) = 1). \end{aligned}$$

It follows that

$$\text{Prob}(C(f(x) \odot (1 - h(x))) = 1) = 2q - p,$$

and, thus,

$$\text{Prob}(C(f(x) \odot (1 - h(x))) = 0) = 1 - 2q + p.$$

Therefore

$$\text{Prob}(E) = \frac{1}{2} \cdot p + \frac{1}{2}(1 - 2q + p) = \frac{1}{2} + (p - q) \geq \frac{1}{2} + \epsilon.$$

This ends the proof of (2). ■

The relation between hard-core predicates and extenders is stated in the following lemma.

Lemma 5.3.4 INFORMAL STATEMENT: *If g is a permutation and h is a hard-core predicate for g , then the function $f(x) = g(x) \odot h(x)$ is an extender.*

FORMAL STATEMENT: *Let $n, S \in \mathbb{N}$ and $\epsilon > 0$ be some parameters. Let $g: \Sigma^n \rightarrow \Sigma^n$ and $h: \Sigma^n \rightarrow \{0, 1\}$ be two functions such that g is a permutation and h is a hard-core predicate for g with security (ϵ, S) . Then*

$$\Delta_{\text{comp}, S}(g(U_n) \odot h(U_n), U_{n+1}) < \epsilon.$$

Proof. By the triangle inequality,

$$\begin{aligned} \Delta_{\text{comp}, S}(g(U_n) \odot h(U_n), U_{n+1}) \\ \leq \Delta_{\text{comp}, S}(g(U_n) \odot h(U_n), g_n(U_n) \odot U_1) + \Delta_{\text{comp}, S}(g_n(U_n) \odot U_1, U_{n+1}). \end{aligned}$$

Since g is a permutation, the distribution $g(U_n) \odot U_1$ is in fact the uniform distribution U_{n+1} . Thus the second term in the right hand side is zero. By assumption, the first term in the right hand side is less than ϵ . ■

Roughly speaking, we will show that any one-way function has a hard-core predicate. Note first that, if f is a one-way function with security (ϵ, S) (and, for concreteness, let us also assume that f is a permutation) then, given $f(x)$, no adversary circuit of size S can find x but with probability ϵ . This does not mean that any given bit of x (such as, say, the first bit) is hidden given $f(x)$. For example, if $g: \Sigma^n \rightarrow \Sigma^n$ is one-way with security $(\epsilon/2, S)$, then the function $f: \Sigma^{n+1} \rightarrow \Sigma^{n+1}$ defined by $f(0 \odot x) = (0 \odot g(x))$ and $f(1 \odot x) = (1 \odot g(x))$ is one-way with security (ϵ, S) ; however, clearly, an adversary, even if modestly endowed with computational power, that is seeing $f(y)$ can immediately retrieve the first bit of y . Nevertheless, the fact that x is globally hard to retrieve given $f(x)$ provides the basis for obtaining a hidden bit.

Actually, in our construction, the hard-core predicate is not built directly from a one-way function $f: \Sigma^n \rightarrow \Sigma^{n'}$, but from a small modification of it, $g: \Sigma^{2n} \rightarrow \Sigma^{n+n'}$, given by $g(x \odot r) = f(x) \odot r$ (where $|x| = |r| = n$). The hard-core predicate is obtained through the utilization of error-correcting codes.

We recall that an error-correcting code is a length-regular function c over strings written in a given alphabet with a strong injectivity property: Any two strings in the image (such strings are called *codewords*) are guaranteed to be far apart. More precisely, the Hamming distance between two strings α and β of equal length is the number of positions in which α and β differ, and it is denoted by $\text{dist}(\alpha, \beta)$. An error correcting code $c: \Sigma^n \rightarrow \Sigma^{n'}$, for some $n, n' \in \mathbb{N}$, has the property that, for some parameter d called the *distance* of the code, and for any $x_1 \neq x_2 \in \Sigma^n$, $\text{dist}(c(x_1), c(x_2)) \geq d \cdot n'$. The *Hamming distance* has the triangle property and it follows that for any $x_1 \neq x_2$ in Σ^n , the balls $B_{(d/2)n'}(c(x_1))$ and $B_{(d/2)n'}(c(x_2))$ are disjoint, where by definition $B_r(u) = \{v \mid \text{dist}(u, v) < r\}$. Thus, a string y that agrees with some codeword $c(x)$ in $> (1 - \frac{d}{2})n'$ bits, defines x uniquely, and, in particular, x can be retrieved effectively from y (provided c is a computable function). This is the decoding property of error-correcting codes, which is a very

strong invertibility feature of c . For many good error-correcting code, the decoding can be done efficiently.

In our construction, we take an error-correcting code $\text{Code} : \Sigma^n \rightarrow \Sigma^{\bar{n}}$ and define the hard-core predicate $P : \Sigma^n \times \Sigma^{\log(\bar{n})} \rightarrow \{0, 1\}$ by $P(x, r) = (\text{Code}(x))(r)$ (i.e., the r -th bit of $\text{Code}(x)$). Then, if we assume that there is a circuit C so that $C(f(x) \odot r) = (\text{Code}(x))(r)$ for a big fraction α of r (and for many x), then the bits $(C(f(x) \odot r))_{r \in \Sigma^{\log(\bar{n})}}$ make a string that is at distance at most $(1 - \alpha) \cdot \bar{n}$ from $\text{Code}(x)$, and, this, by the property of the error-correcting code, should allow us to retrieve x . This will contradict the fact that f is one-way. One problem is that in our assumption, α can not be taken to be $1 - d/2$ (where d is the distance of the code), but much smaller. Therefore we cannot retrieve x directly, as we have claimed, and we will be content to recover only a relatively small list of elements which contains x and whose cardinality is polynomial in n . This operation is called the *list-decoding* of an error-correcting code. List-decoding is here good enough because we can try all the elements in the list and check which one maps via f into $f(x)$. Thus we are able to invert f , still obtaining the contradiction we need.

For concreteness, we will be using the *Hadamard error-correcting code*, $\text{Had} : \Sigma^n \rightarrow \Sigma^{2^n}$, defined as follows: For all $y \in \{0, \dots, 2^n - 1\}$, the y -th bit of $\text{Had}(x)$ is the inner product $x \cdot y$ (in the vector space $(\text{GF}(2))^n$.) In other words, we write y in base 2 as $y_1 y_2 \dots y_n$ and the y -th bit of $\text{Had}(x)$, where $x = x_1 x_2 \dots x_n$, is $x_1 y_1 + x_2 y_2 + \dots + x_n y_n \pmod{2}$. Since $|\text{Had}(x)| = 2^n$, it follows that $|r| = n$ (r is the string from the previous paragraph) and, thus, we can not afford to calculate $C(f(x), r)$ for all $r \in \Sigma^n$ to construct a list of elements in Σ^n one of which is x . Fortunately, such a list can be obtained by looking at only a few elements of $(C(f(x), r))_{r \in \Sigma^n}$. This property of Hadamard codes is stated in the next theorem. In the statement, we make reference to a circuit that takes as inputs a string $x \in \Sigma^n$ and a string $y \in \Sigma^{2^n}$. Since y is very long and we do not want the circuit to have that many input gates, access to y is provided via the so called *oracle access* mechanism. This means that we use a different type of a circuit, called an *oracle circuit*, and the string y plays the role of an oracle set. An oracle circuit, in addition to the usual AND, OR, and NOT gates, also has *oracle gates*. Each oracle gate has n inputs and an instance $\alpha_1 \dots \alpha_n$ of these inputs, the gate outputs the bit of y located at address $\alpha_1 \dots \alpha_n$. The number of queries is the number of oracle gates. It will be the case that the oracle circuit only accesses a few of the bits of y and the locations of these bits are determined by the input x .

Theorem 5.3.5 (List decoding for Hadamard codes) *Let $n \in \mathbb{N}$ and $\epsilon > 0$ be two parameters. Consider the Hadamard error correcting code $\text{Had} : \Sigma^n \rightarrow \Sigma^{2^n}$. There is a probabilistic circuit A that has oracle access to a string of length 2^n so that:*

- (a) *If $x \in \Sigma^n$ and $y \in \Sigma^{2^n}$ are two strings so that $\text{dist}(y, \text{Had}(x)) \leq (\frac{1}{2} - \epsilon)2^n$, then, with probability at least $\frac{3}{4}$, A on input 1^n and with oracle access to y outputs a list of $(n \cdot \frac{1}{\epsilon^2} + 1)$ strings which includes x ;*
- (b) *The circuit A makes $n^2 \cdot \frac{1}{\epsilon^2}$ queries to y and has size $O(n^3 \cdot \frac{1}{\epsilon^4})$.*

Proof. For $y \in \Sigma^{2^n}$ and $r \in \Sigma^n$, $y(r)$ denotes as usual the r -th bit of y . For i , $1 \leq i \leq n$, let e_i be the vector $(0, \dots, 0, 1, 0, \dots, 0)$, where the single 1 is in position i . Let \oplus denote the addition of vectors in $(\text{GF}(2))^n$. One first observation is that if for some i we have the r and the $(r \oplus e_i)$ bits of $\text{Had}(x)$, then we can retrieve $x(i)$, the i -th bit of x . This is so because

$$(\text{Had}(x))(r) + (\text{Had}(x))(r \oplus e_i) = \sum x_i \cdot r_i + \sum x_i(r_i \oplus e_i) = x(i),$$

where the addition $+$ is in $\text{GF}(2)$ (i.e., modulo 2). Thus, if we pick at random r and we read the bits in positions r and $r \oplus e_i$ of y and if we are lucky and $y(r) = (\text{Had}(x))(r)$ and $y(r \oplus e_i) = (\text{Had}(x))(r \oplus e_i)$, then we can retrieve $x(i)$. However, the probability that “we are lucky” is only guaranteed to be at least $1 - 2(\frac{1}{2} - \epsilon) = 2\epsilon$, which may be less than $\frac{1}{2}$. Note that $\frac{1}{2}$ is the probability that we obtain if we just guess $x(i)$ directly. Therefore, if $\epsilon < 1/4$, it does not help to look at both $y(r)$ and $y(r \oplus e_i)$. To understand the idea of the algorithm, suppose that we know $(\text{Had}(x))(r)$ for some random r . Then it is enough to look at $y(r \oplus e_i)$ hoping that it is equal to $(\text{Had}(x))(r \oplus e_i)$. This would allow us to calculate $x(i)$ correctly with the probability that our hope comes true, i.e., with probability at least $1/2 + \epsilon$. However, we do not know $(\text{Had}(x))(r)$ for any r . This obstacle can be overcome by taking a sample set S of r 's, assigning all possible values to $((\text{Had}(x))(r))_{r \in S}$, looking at the corresponding $y(r \oplus e_i)$, and taking our guess for $x(i)$ to be given by the majority vote over all r in S . Each assigned value will give one candidate for x , and, with good probability, the correct assignment gives us x . The sample set is constructed by taking the random choices of r in Σ^n to be only pairwise independent. This allows us to obtain only a small number of possible candidates (i.e., the LIST) for x .

The complete algorithm is as follows.

Let $m = \log(n \cdot \frac{1}{\epsilon^2} + 1)$.

Step 1. Select a random binary $n \times m$ matrix T . (Comment: T is used for the pairwise independent choices of various r .)

For each vector $\tau \in \Sigma^m$, we produce a string $z_\tau \in \Sigma^n$ as shown below in Steps 2, 3, and 4. In the end, $\text{LIST} = \{z_\tau \mid \tau \in \Sigma^m\}$. So, let $\tau \in \Sigma^m$.

Step 2. For each $j \in \Sigma^m - \{(0, \dots, 0)\}$, calculate

$$r^j = T \cdot j \pmod{2}.$$

Step 3. For every $i \in \{1, \dots, n\}$ and every $j \in \Sigma^m - \{(0, \dots, 0)\}$, calculate

$$z_i^j = (\tau \cdot j) + y(r^j \oplus e_i) \pmod{2}.$$

(Comment:

We hope that $y(r^j \oplus e_i)$ is $(\text{Had}(x))(r^j \oplus e_i)$ and that $\tau \cdot j$ is $(\text{Had}(x))(r^j)$.)

Set z_i to the value of the majority of z_i^j (the majority is taken over $j \in \Sigma^m - \{(0, \dots, 0)\}$).

Step 4. We set $z_\tau = z_1 z_2 \dots z_n$.

Claim 5.3.6 *Suppose $\tau \cdot j = (\text{Had}(x))(r^j)$, for all $j \in \Sigma^m - \{(0, \dots, 0)\}$. Let $i \in \{1, \dots, n\}$. Then*

$$\text{Prob}(z_\tau(i) = x(i)) > 1 - \frac{1}{4n}.$$

Proof. For every vector $j \in \Sigma^m - \{(0, \dots, 0)\}$, we define the random variable

$$X^j = \begin{cases} 1, & \text{if } (\text{Had}(x))(r^j) + y(r^j \oplus e_i) = x(i) \pmod{2}, \\ 0, & \text{otherwise.} \end{cases}$$

The random variable $r^j = T \cdot j$ is uniformly distributed in Σ^n (when T is selected uniformly at random as a binary $n \times m$ matrix). Recalling that $(\text{Had}(x))(r) + (\text{Had}(x))(r \oplus e_i) = x(i) \pmod{2}$, it follows that

$$\text{Prob}(X^j = 1) \geq \text{Prob}(y(r^j \oplus e_i) = (\text{Had}(x))(r \oplus e_i)) \geq \frac{1}{2} + \epsilon$$

(the last inequality follows from the Theorem's hypothesis about $\text{dist}(y, \text{Had}(x))$). Let $M = 2^m - 1$. The probability that the majority of z_i^j is not equal to $x(i)$ is equal to $\text{Prob}(\sum_j X^j \leq \frac{1}{2} \cdot M)$. The next observation is that the variables r^j are pairwise independent. To check this, let T_i denote the i -th row of the matrix T , $i = 1, \dots, n$, let b_1 and b_2 be two bits, and let j and k be two distinct vectors in $(\text{GF}(2))^n$. It is easy to see that, for all i , $\text{Prob}_T(T_i \cdot j = b_1 \text{ and } T_i \cdot k = b_2) = \frac{1}{4}$. Then, for any two vectors u, v in $(\text{GF}(2))^n$,

$$\begin{aligned} \text{Prob}_T(r^j = u \text{ and } r^k = v) &= \text{Prob}_T(T \cdot j = u \text{ and } T \cdot k = v) \\ &= \text{Prob}_T(\forall i, T_i \cdot j = u_i \text{ and } T_i \cdot k = v_i) = \\ &= \prod_{i=1}^n \text{Prob}_T(T_i \cdot j = u_i \text{ and } T_i \cdot k = v_i) = \frac{1}{4^n} \\ &= \text{Prob}_T(r^j = u) \cdot \text{Prob}_T(r^k = v). \end{aligned}$$

It follows that the random variables X^j are pairwise independent as well. Thus, by Chebyshev's inequality, denoting expectation as $E(\cdot)$ and variation as $\text{Var}(\cdot)$,

$$\begin{aligned} &\text{Prob}\left(\sum X^j \leq \frac{1}{2} \cdot M\right) \\ &= \text{Prob}\left(\sum (1 - X^j) \geq \frac{1}{2} \cdot M\right) \\ &= \text{Prob}\left(\sum (1 - X^j) - M \cdot E(1 - X^1) \geq M\left(\frac{1}{2} - E(1 - X^1)\right)\right) \end{aligned}$$

$$\begin{aligned}
&\leq \text{Prob}\left(\left|\sum(1 - X^j) - \text{E}(1 - X^1) \cdot M\right| \geq M \cdot \epsilon\right) \\
&\leq \frac{\text{Var}(\sum(1 - X^j))}{M^2 \cdot \epsilon^2} && \text{(by Chebyshev's inequality)} \\
&= \frac{M \cdot \text{Var}(1 - X^1)}{M^2 \cdot \epsilon^2} && \text{(since } X^{j'} \text{ are pairwise independent)} \\
&\leq \frac{\text{Var}(1 - X^1)}{\epsilon^2 \cdot n \cdot \frac{1}{\epsilon^2}} \\
&\leq \frac{\frac{1}{4}}{n} = \frac{1}{4n}
\end{aligned}$$

(we took into account that $M = 2^m - 1 = n(\epsilon^{-2})$ and that the variance of any boolean random variable is at most $1/4$). ■

(Proof of Theorem 5.3.5 continued.) Hence, if $\tau \cdot j = (\text{Had}(x))(r^j)$ for all $j \in \Sigma^m - \{(0, \dots, 0)\}$, then the string z_τ is equal to x with probability at least $1 - n \cdot \frac{1}{4n} = \frac{3}{4}$. Note that

$$\begin{aligned}
\tau \cdot j &= (\text{Had}(x))(r^j) \\
\iff \tau \cdot j &= (\text{Had}(x))(T \cdot j) && \text{(by definition of } r^j) \\
\iff \tau \cdot j &= x \cdot (T \cdot j) && \text{(by definition of } \text{Had}(x)) \\
\iff \tau \cdot j &= (x \cdot T) \cdot j.
\end{aligned}$$

Since we try all $\tau \in \Sigma^m$, for $\tau = x \cdot T$ it holds that $\tau \cdot j = (\text{Had}(x))(r^j)$ for all $j \in \Sigma^m - \{(0, \dots, 0)\}$. Therefore with probability at least $\frac{3}{4}$, the string x is z_τ , and, thus, with probability at least $\frac{3}{4}$ it appears in LIST.

By inspecting the algorithm, we see that the number of queries to y is $n(2^m - 1) = (\frac{n}{\epsilon})^2$ and that the number of elementary operations is $O(2^m \cdot n \cdot (2^m - 1)) = O(n^3 \cdot \epsilon^{-4})$. This concludes the proof of Theorem 5.3.5. ■

Lemma 5.3.7 *Let $f: \Sigma^n \rightarrow \Sigma^{n'}$ be a function computable by a circuit of size $p(n)$. Let G be a circuit that on input $f(x) \odot r$, with $|r| = n$, attempts to calculate the function $P(x, r) \stackrel{\text{def}}{=} (\text{Had}(x))(r)$ and let*

$$\epsilon = \text{Prob}_{x,r}(G(f(x) \odot r) = P(x, r)) - \frac{1}{2}.$$

Then there is a circuit A with size bounded by $O(n^3 \cdot \frac{1}{\epsilon^4} + n \cdot p(n) \cdot \frac{1}{\epsilon^2} + n^2 \cdot \frac{1}{\epsilon^2} \cdot \text{size}(G))$ so that

$$\text{Prob}_x(A(f(x)) \in f^{-1}(f(x))) \geq \frac{3}{4} \cdot \epsilon.$$

Proof. We can assume that $\epsilon > 0$ (the case $\epsilon \leq 0$ is trivial and not interesting). For each $x \in \Sigma^n$, let

$$s(x) = \text{Prob}_r(G(f(x) \odot r) = P(x, r)).$$

Let $\text{GOOD} = \{x \in \Sigma^n \mid s(x) \geq 1/2 + \epsilon/2\}$. We first observe the following fact.

Claim 5.3.8 $\|\text{GOOD}\| \geq \epsilon \cdot 2^n$.

Proof. From the hypothesis, we know that $2^{-n} \sum_x (s(x) - 1/2) = \epsilon$. Also, for all $x \in \Sigma^n$, $s(x) - 1/2 \leq 1/2$. Then, $2^n \cdot \epsilon \leq (2^n - \|\text{GOOD}\|) \cdot \frac{\epsilon}{2} + \|\text{GOOD}\| \cdot \frac{1}{2}$. After some simple calculations, we obtain $\|\text{GOOD}\|/2^n \geq \epsilon$. ■

For each $x \in \Sigma^n$, let $y(x)$ be the string obtained by concatenating in the lexicographical order of the strings $r \in \Sigma^n$, the bits $G(f(x) \odot r)$. Then, for each $x \in \text{GOOD}$, $\text{dist}(y(x), \text{Had}(x)) \leq \frac{1}{2} - \frac{\epsilon}{2}$. We apply Theorem 5.3.5 for $y(x)$ and x ; however, instead of querying the bits of $y(x)$ we calculate them using the circuit G on input $f(x)$ and various r . We obtain the set of strings LIST having $(n \cdot (\frac{\epsilon}{2})^2 + 1)$ strings and which, with probability at least $\frac{3}{4}$, contains the string x . By checking all the elements in LIST, we find one which maps via f into the input $f(x)$. This entire procedure can be implemented by a probabilistic circuit A' of size $O(n^3 \cdot \frac{16}{\epsilon^4} + \frac{4n^2}{\epsilon^2} \cdot \text{size}(G) + \frac{4n}{\epsilon^2} \cdot p(n))$. Note that

$$\begin{aligned} & \text{Prob}(A'(f(x)) \in f^{-1}(f(x))) \\ & \geq \text{Prob}(x \in \text{GOOD}) \cdot \text{Prob}(A(f(x)) \in f^{-1}(f(x)) \mid x \in \text{GOOD}) \geq \epsilon \cdot \frac{3}{4}, \end{aligned}$$

where the probability is taken over x chosen uniformly at random in Σ^n and over the random choices of A' . We can convert A' to a deterministic circuit A by observing that there has to be a random choice r_0 so that the above relation holds when x is chosen in Σ^n and r is fixed to r_0 (i.e., the circuit A does what A' is doing only that it is using the fixed r_0 to simulate the random choices of A'). The size of A is equal to $\text{size}(A') + |r_0| \leq 2\text{size}(A')$. The proof of the lemma is now complete. ■

Theorem 5.3.9 INFORMAL STATEMENT: *Given a one-way function f , we can build a hard-core predicate for a small variation of f .*

FORMAL STATEMENT: *Let $\epsilon: \mathbb{N} \rightarrow [0, 1]$ and $S: \mathbb{N} \rightarrow \mathbb{N}$ be two functions such that $S(n)$ is superpolynomial and $\epsilon(n) \geq n \cdot (S(n))^{-1/4}$ for all sufficiently large n . Let $(f_n)_{n \in \mathbb{N}}$ be an (ϵ, S) one-way function. Let $P_n: \Sigma^n \times \Sigma^n \rightarrow \{0, 1\}$ be defined by $P_n(x, r) = (\text{Had}(x))(r)$ (i.e., $P_n(x, r) = x \cdot r$.) Then there is a constant c so that, for any family of circuits $(G_n)_{n \in \mathbb{N}}$ with $\text{size}(G_n)$ at most $c \cdot \frac{\epsilon(n)^2}{n^2} \cdot S(n)$, and for all sufficiently large n ,*

$$\text{Prob}_{x,r}(G_n(f_n(x) \odot r) = P_n(x, r)) < \frac{1}{2} + \frac{4}{3} \cdot \epsilon(n). \quad (5.5)$$

Proof. This is an easy consequence of Lemma 5.3.7. For some constant c , if $\text{size}(G_n) = c \cdot \frac{\epsilon(n)^2}{n^2} \cdot S(n)$, then the size of the circuit A that results from G_n in Lemma 5.3.7 is at most $S(n)$. Assume that the opposite of inequality (5.5) holds for a family of circuits $(G_n)_{n \in \mathbb{N}}$ with size as above and for infinitely many n . It follows that a circuit of size $S(n)$ (namely, the circuit A) can invert the function f_n with a bias that is at least $\epsilon(n)$. Since this happens for infinitely many n , we have reached a contradiction to the fact that $(f_n)_{n \in \mathbb{N}}$ is an (ϵ, S) one-way function. ■

Note that $(\text{Had}(x))(r)$ is simply the inner product of x and r , which we denote by $x \cdot r$ (x and r are viewed here as vectors in $\text{GF}(2)^n$). Taking into account Theorem 5.3.3, the above result says that, essentially, the inner product of x and r is a hard-core predicate of $f_n(x) \odot r$, whenever $(f_n)_{n \in \mathbb{N}}$ is a one-way function. One small problem is that this hard-core predicate is defined only for inputs of even length, but this can be remedied easily. For $y \in \Sigma^*$, let y_{left} be the left half of y and y_{right} be the right half of y with the provision that if $|y|$ is odd then $y_{\text{left}} = y(1 : \lfloor |y|/2 \rfloor)$ and $y_{\text{right}} = y(\lfloor |y|/2 \rfloor + 1, |y|)$.

Lemma 5.3.10 *Let $\epsilon: \mathbb{N} \rightarrow [0, 1]$ and $S: \mathbb{N} \rightarrow \mathbb{N}$ be two functions such that $S(n)$ is superpolynomial and $\epsilon(n) \geq n \cdot (S(n))^{-1/4}$ for all sufficiently large n . Let $(f_n)_{n \in \mathbb{N}}$ be an (ϵ, S) one-way function.*

(a) *For each $n \in \mathbb{N}$, let g_{2n} and h_{2n} be the functions defined on strings of length $2n$ by $g_{2n}(y) \stackrel{\text{def}}{=} f_n(y_{\text{left}}) \odot y_{\text{right}}$ and, respectively, by $h_{2n}(y) \stackrel{\text{def}}{=} y_{\text{left}} \cdot y_{\text{right}}$. Then, for some constant d and for all sufficiently large n , h_{2n} is a hard-core predicate for g_{2n} with security $(\frac{4}{3}\epsilon(n), d \cdot \frac{\epsilon(n)^2}{n^2} S(n))$.*

(b) *For each $n \in \mathbb{N}$ let g_{2n+1} and h_{2n+1} be the functions defined on strings of length $2n + 1$ by $g_{2n+1}(y) \stackrel{\text{def}}{=} f_n(y_{\text{left}}) \odot y_{\text{right}}$ and, respectively, by $h_{2n+1}(y) \stackrel{\text{def}}{=} y_{\text{left}} \cdot y_{\text{right}}(1 : n)$. Then, for some constant d and for all sufficiently large n , h_{2n+1} is a hard-core predicate for g_{2n+1} with security $(\frac{4}{3}\epsilon(n), d \cdot \frac{\epsilon(n)^2}{n^2} S(n))$.*

Proof. (a) By Theorem 5.3.9, any circuit of size at most $c \cdot \frac{\epsilon(n)^2}{n^2} \cdot S(n)$ on input $f_n(y_{\text{left}}) \odot y_{\text{right}}$ computes $y_{\text{left}} \cdot y_{\text{right}}$ on less than a fraction $\frac{1}{2} + \frac{4}{3}\epsilon(n)$ of all $y \in \Sigma^{2n}$. Theorem 5.3.3 implies that the computational distance between the distributions $g_{2n}(u_{2n}) \odot h_{2n}(u_{2n})$ and $g_{2n}(u_{2n}) \odot u_1$ is less than $\frac{4}{3}\epsilon(n)$ relative to circuit size $d \cdot \frac{\epsilon(n)^2}{n^2} \cdot S(n)$, for some constant d . \blacksquare

(b) It is easy to check that Theorem 5.3.9 implies that any circuit of size at most $c \cdot \frac{\epsilon(n)^2}{n^2} \cdot S(n)$ on input $f_n(y_{\text{left}}) \odot y_{\text{right}}$ computes $y_{\text{left}} \cdot y_{\text{right}}(1 : n)$ on less than a fraction $\frac{1}{2} + \frac{4}{3}\epsilon(n)$ of all $y \in \Sigma^{2n+1}$ (the last bit of y_{right} cannot be too helpful; if it were it could be fixed to some advantageous value and build a circuit that computes $y_{\text{left}} \cdot y_{\text{right}}(1 : n)$ on input $f_n(y_{\text{left}}) \odot y_{\text{right}}(1 : n)$ on at least a fraction of $\frac{1}{2} + \frac{4}{3}\epsilon(n)$ of all strings $y_{\text{left}} \odot y_{\text{right}}(1 : n)$). The rest is as in (a). \blacksquare

All the preparations are ready for producing an extender, provided we are given a one-way permutation.

Theorem 5.3.11 **INFORMAL STATEMENT:** *Given a one-way permutation, we can construct an extender.*

FORMAL STATEMENT: *Let $\epsilon: \mathbb{N} \rightarrow [0, 1]$ and $S: \mathbb{N} \rightarrow \mathbb{N}$ be two functions such that $S(n)$ is superpolynomial and $\epsilon(n) \geq n \cdot (S(n))^{-1/4}$ for all sufficiently large n . Suppose that $(f_n)_{n \in \mathbb{N}}$ is a one-way function with security (ϵ, S) such that, for each n , $f_n: \Sigma^n \rightarrow \Sigma^n$ is a permutation. Then there exists $(\tilde{g}_n)_{n \in \mathbb{N}}$ an extender with security $(\frac{4}{3}\epsilon(n), O(\frac{\epsilon(n)^2}{n^2} S(n)))$. Moreover, there exists a polynomial q such that, for all n , \tilde{g}_n is computable by a circuit of size $q(n)$.*

Proof. We consider the family of functions $(g_n)_{n \in \mathbb{N}}$ and $(h_n)_{n \in \mathbb{N}}$ constructed in Lemma 5.3.10 from the family of functions $(f_n)_{n \in \mathbb{N}}$. It is easy to see that g_n and h_n can be calculated in time bounded by $p(n)$, for some polynomial p . It is also easy to check that, for each n , g_n is a permutation of Σ^n . Since, for some constant d and for each n , h_n is a hard-core predicate for g_n with security $(\frac{4}{3}\epsilon(n), d \cdot \frac{\epsilon(n)^2}{n^2} \cdot S(n))$, using Lemma 5.3.4, we derive that

$$\Delta_{\text{comp}, d \cdot \frac{\epsilon(n)^2}{n^2} \cdot S(n)}(g_n(U_n) \odot h_n(U_n), U_{n+1}) < \frac{4}{3}\epsilon(n).$$

The conclusion follows for $\tilde{g}_n(x) \stackrel{\text{def}}{=} g_n(x) \odot h_n(x)$. ■

5.4 From extenders to pseudo-random generators

IN BRIEF: It is shown how to enlarge the extension of a pseudo-random generator.

Given an extender, we can build a pseudo-random generator with much more significant extension. We present here the construction. The starting point is a function $g: \Sigma^n \rightarrow \Sigma^{n+1}$ that is an extender with security (ϵ, S) . We will build a pseudo-random generator $h: \Sigma^n \rightarrow \Sigma^L$, for some arbitrary L^3 larger than n , as follows. First we compute $g(x)$, we retain the first bit of $g(x)$, and then we use the remaining n bits of $g(x)$ as a seed for a new invocation of g . Since g is a pseudo-random generator, this seed is almost as good as a fresh independently chosen random string of length n . We iterate this process L times and we output all the bits that have been retained.

More formally, let us define

$$\text{head}(x) = g(x)(1),$$

and

$$\text{tail}(x) = g(x)(2 : n + 1).$$

For each $k \geq 1$, let $h^k: \Sigma^n \rightarrow \Sigma^k$ be defined by

$$h^k(x) = \underbrace{(\text{head}(x) \odot \text{head}(\text{tail}(x)) \odot \dots \odot \text{head}(\text{tail}(\dots \text{tail}(x) \dots)))}_k.$$

As explained above, we take

$$h(x) = h^L(x).$$

³In order to make the construction meaningful, the value of L will in fact be bounded by a function of ϵ and S .

Theorem 5.4.1 INFORMAL STATEMENT: *Given an (ϵ, S) extender with large S , we can construct a pseudo-random generator with large extension.*

FORMAL STATEMENT: *Let $\epsilon > 0$ and $S, q, L \in \mathbb{N}$. Let $g: \Sigma^n \rightarrow \Sigma^{n+1}$ be an extender with security (ϵ, S) that is computable by a circuit of size q . Then the function $h: \Sigma^n \rightarrow \Sigma^L$ defined as above is a pseudo-random generator with security $(L \cdot \epsilon, S - 2L \cdot q - O(1))$.*

Proof. For the sake of this proof it is helpful to introduce the following distributions d^0, d^1, \dots, d^L , all defined on the space Σ^L . For each $k \in \{0, \dots, L\}$, d^k is the random variable obtained by taking uniformly at random a k -bits long binary string followed by the $(L - k)$ -bits long binary string that results from applying h^{L-k} to a random binary string of length n . Formally,

$$d^k = U_k \odot h^{L-k}(U_n).$$

Observe that $d^0 = h(U_n)$ and $d^L = U_L$. The distributions d^0, \dots, d^L are sometimes called *hybrid distributions* because they are obtained via a mixed usage of pure random strings and of the extender g . Note the fine gradual passage from d^0 to d^L realized by the intermediate distributions d^1, \dots, d^{L-1} . What is important is that the computational distance between any two consecutive d^i and d^{i+1} is, as we will prove shortly, at most ϵ for circuits of size not much smaller than S , which implies that the computational distance between the two extremes d^0 and d^L cannot be too large. This proof technique is an instance of the so-called *hybrid method* which we are going to see later as well.

Suppose that the computational distance between the distributions $(h(U_n))$ and (U_L) is greater than $L \cdot \epsilon$ for circuits of size $S - 2L \cdot q(n)$. In other words, suppose that there is a circuit C of size $S - 2L \cdot q(n)$ such that

$$|\text{Prob}(C(d^0) = 1) - \text{Prob}(C(d^L) = 1)| \geq L \cdot \epsilon.$$

Clearly,

$$\begin{aligned} & |\text{Prob}(C(d^0) = 1) - \text{Prob}(C(d^L) = 1)| \\ & \leq |\text{Prob}(C(d^0) = 1) - \text{Prob}(C(d^1) = 1)| \\ & \quad + |\text{Prob}(C(d^1) = 1) - \text{Prob}(C(d^2) = 1)| \\ & \quad \vdots \\ & \quad + |\text{Prob}(C(d^{L-1}) = 1) - \text{Prob}(C(d^L) = 1)|. \end{aligned}$$

Therefore, there is $k \in \{0, \dots, L - 1\}$ such that

$$|\text{Prob}(C(d^k) = 1) - \text{Prob}(C(d^{k+1}) = 1)| \geq \epsilon. \quad (5.6)$$

An inspection of d^k and d^{k+1} reveals that the difference between them stems from the fact that d^{k+1} is obtained by applying a deterministic easy-to-compute function to a random $y \in \Sigma^{n+1}$, while d^k is obtained by applying the same deterministic function to $g(x)$, with x randomly chosen in Σ^n . Thus, the relation (5.6)

implies that a minor modification of the circuit C is able to distinguish with bias greater than ϵ between the distributions $g(U_n)$ and U_{n+1} , and this contradicts the hypothesis.

Let us formalize this argument. For $z \in \Sigma^{n+1}$, let

$$\text{first}(z) = z(1), \text{ and}$$

$$\text{last}(z) = z(2 : n + 1).$$

Thus, $\text{head}(x) = \text{first}(g(x))$ and $\text{tail}(x) = \text{last}(g(x))$. Let $f: \Sigma^{n+1} \rightarrow \Sigma^{L-k}$ be defined by

$$f(z) = \text{first}(z) \odot h^{L-k-1}(\text{last}(z)).$$

It can be checked that f can be computed by a circuit of size $L \cdot q(n) + O(1)$. Now, d_k is

$$U_k \odot \underbrace{\text{head}(U_n) \odot \text{head}(\text{tail}(U_n)) \odot \dots \odot \text{head}(\text{tail}(\dots(\text{tail}(U_n)\dots)))}_{L-k}$$

and d_{k+1} is

$$U_k \odot U_1 \odot \underbrace{\text{head}(U_n) \odot \text{head}(\text{tail}(U_n)) \odot \dots \odot \text{head}(\text{tail}(\dots(\text{tail}(U_n)\dots)))}_{L-k-1}.$$

Thus,

$$d_k = U_k \odot f(g(U_n)).$$

Observe that d_{k+1} can be viewed as the concatenation of U_k with

$$\underbrace{\text{first}(U_{n+1}) \odot \text{head}(\text{last}(U_{n+1})) \odot \dots \odot \text{head}(\text{tail}(\dots(\text{tail}(\text{last}(U_{n+1}))\dots)))}_{L-k},$$

and thus,

$$d_{k+1} = U_k \odot f(U_{n+1}).$$

Therefore, the relation (5.6) can be rewritten as

$$\begin{aligned} & |\text{Prob}_{U_k, U_n}(C((U_k \odot f(g(U_n)))) = 1) \\ & \quad - \text{Prob}_{U_k, U_{n+1}}(C((U_k \odot f(U_{n+1}))) = 1)| \geq \epsilon. \end{aligned}$$

Clearly, there is some fixed string u_k^0 of length k such that

$$\begin{aligned} & |\text{Prob}_{U_n}(C((u_k^0 \odot f(g(U_n)))) = 1) \\ & \quad - \text{Prob}_{U_{n+1}}(C((u_k^0 \odot f(U_{n+1}))) = 1)| \geq \epsilon. \end{aligned} \quad (5.7)$$

Now we can define a circuit D that is able to distinguish between the distributions $(g(U_n))$ and U_{n+1} . The circuit D has u_k^0 hardwired in its circuitry, and on input

$z \in \Sigma^{n+1}$ it simulates the circuit C on input $u_k^0 \odot f(z)$. It is easy to see that the size of D is bounded by $\text{size}(C) + L \cdot q(n) + O(1) + L \leq \text{size}(C) + 2L \cdot q(n) \leq S$ (because we need to “store” u_k^0 and to calculate $f(z)$). By the definition of D ,

$$\text{Prob}_{U_n}(D(g(U_n)) = 1) = \text{Prob}_{U_n}(C((u_k^0 \odot f(g(U_n)))) = 1)$$

and

$$\text{Prob}_{U_{n+1}}(D(U_{n+1}) = 1) = \text{Prob}_{U_{n+1}}(C((u_k^0 \odot f(U_{n+1}))) = 1).$$

Therefore the relation (5.7) implies

$$|\text{Prob}_{U_n}(D(g(U_n)) = 1) - \text{Prob}_{U_{n+1}}(D(U_{n+1}) = 1)| \geq \epsilon,$$

and this contradicts the fact that g is an (ϵ, S) -secure extender. ■

We have finally built the pseudo-random generator. It is the moment to contemplate the entire construction. We have started with an ensemble $(f_n)_{n \in \mathbb{N}}$ of one-way permutations, $f_n: \Sigma^n \rightarrow \Sigma^n$, having security (ϵ, S) . We have next obtained an ensemble of extenders $(g_n)_{n \in \mathbb{N}}$ with security (ϵ', S') , where $\epsilon'(n) = \frac{4}{3}\epsilon(n)$ and $S'(n) = O(1) \cdot \frac{\epsilon^2}{n^2}S(n)$ (provided S is a superpolynomial function and $\epsilon(n) \geq n \cdot (S(n))^{-1/4}$). There is a polynomial $q(n)$ such that each g_n can be calculated in time $q(n)$. In the last construction stage, we have produced the ensemble $(h_n)_{n \in \mathbb{N}}$, $h_n: \Sigma^n \rightarrow \Sigma^{L(n)}$, of pseudo-random generators with security (ϵ'', S'') , where $\epsilon''(n) = L(n) \cdot \epsilon'(n)$ and $S''(n) = S'(n) - 2L(n)q(n) - O(1)$. Thus, $\epsilon''(n) = L(n) \cdot \frac{4}{3}\epsilon(n)$ and $S''(n) = O(1) \cdot \frac{\epsilon^2}{n^2}S(n) - 2L(n)q(n) - O(1)$.

Naturally, the quality of the pseudo-random generator depends on the quality of the one-way permutation. In particular, the following theorem holds.

Theorem 5.4.2 (a) *Suppose there exists a strong one-way function $(f_n)_{n \in \mathbb{N}}$ such that for all n , $f_n: \Sigma^n \rightarrow \Sigma^n$ is a permutation. Then there exists a strong pseudo-random generator $(g_n)_{n \in \mathbb{N}}$, where, for each n , $g_n: \Sigma^n \rightarrow \Sigma^{L(n)}$ and $L(n)$ is a superpolynomial function.*

(b) *Suppose there exists an exponentially strong one-way function $(f_n)_{n \in \mathbb{N}}$ such that for all n , $f_n: \Sigma^n \rightarrow \Sigma^n$ is a permutation. Then there exists an exponentially strong pseudo-random generator $(g_n)_{n \in \mathbb{N}}$, where, for each n , $g_n: \Sigma^n \rightarrow \Sigma^{L(n)}$ and $L(n)$ is an exponential function (i.e., $L(n) = 2^{cn}$ for some constant c).*

Proof. (a) We are using the notation from the previous paragraph. We can assume that $\epsilon(n) \geq n \cdot (S(n))^{-1}$. (If the opposite inequality holds, we can substitute $\epsilon(n)$ with the larger $\epsilon_1(n) = n \cdot (S(n))^{-1/4}$ and of course $(f_n)_{n \in \mathbb{N}}$ is one-way with security $(\epsilon_1(n), S(n)^{1/4})$ and $1/\epsilon_1$ is still superpolynomial.) We take $L(n) = \min((\epsilon(n))^{-1/2}, (S(n))^{1/3})$. One can check that $1/\epsilon''(n)$ and $S''(n)$ are superpolynomial.

(b) Similar to (a). ■