

A Pseudorandom Generator from any One-way Function

Johan Håstad* Russell Impagliazzo† Leonid A. Levin‡ Michael Luby§

Abstract

Pseudorandom generators are fundamental to many theoretical and applied aspects of computing. We show how to construct a pseudorandom generator from *any* one-way function. Since it is easy to construct a one-way function from a pseudorandom generator, this result shows that there is a pseudorandom generator iff there is a one-way function.

Warning: Essentially this paper has been published in SIAM Journal on Computing and is hence subject to copyright restrictions. It is for personal use only.

1 Introduction

One of the basic primitives in the study of the interaction between randomness and feasible computation is a pseudorandom generator. Intuitively, a *pseudorandom generator* is a

*Department of Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm, Sweden, email JohanH@nada.kth.se. Research supported by the Swedish National Board for Technical Development.

†Department of Computer Science, University of California at San Diego. Research partially done while at U.C.Berkeley, email Russell@cs.ucsd.edu. Research supported by NSF grant CCR 88-13632

‡Computer Science Department, Boston University, 111 Cummington St, Boston, MA 02215, email Lnd@cs.bu.edu. Research supported by NSF grant CCR-9015276.

§International Computer Science Institute, Berkeley, California, and Computer Science Division, University of California at Berkeley, email Luby@icsi.berkeley.edu. Research partially done while on leave of absence from the University of Toronto. Research supported by NSERC operating grant A8092, National Science Foundation operating grant CCR-9016468, National Science Foundation operating grant CCR-9304722, United States-Israel Binational Science Foundation grant No. 89-00312, United States-Israel Binational Science Foundation grant No. 92-00226, and ESPRIT BR Grant EC-US 030.

polynomial time computable function g that stretches a short random string x into a long string $g(x)$ that “looks” random to any feasible algorithm, called an adversary. The adversary tries to distinguish the string $g(x)$ from a random string the same length as $g(x)$. The two strings “look” the same to the adversary if the acceptance probability for both strings is essentially the same. Thus, a pseudorandom generator can be used to efficiently convert a small amount of true randomness into a much larger number of effectively random bits.

The notion of randomness tests for a string evolved over time: from set-theoretic tests to enumerable [Kol: 65], recursive and finally limited time tests. Motivated by cryptographic applications, the seminal paper [BM: 82] introduces the idea of a generator which produces its output in polynomial time such that its output passes a general polynomial time test. The fundamental paper [Yao: 82] introduced the definition of a pseudorandom generator most commonly used today, and proves that this definition and the original of [BM: 82] are equivalent.

The robust notion of a pseudorandom generator, due to [BM: 82], [Yao: 82], should be contrasted with the classical methods of generating random looking bits as described in, e.g., [Knuth: 81]. In studies of classical methods, the output of the generator is considered good if it passes a particular set of standard statistical tests. The linear congruential generator is an example of a classical method for generating random looking bits that pass a variety of standard statistical tests. However, [Boyar: 89] and [Kraw: 92] show that there is a polynomial time statistical test which the output from this generator does not pass.

The distinction between the weaker requirement that the output pass some particular statistical tests and the stronger requirement that it pass all feasible tests is particularly important in the context of many applications. As pointed out by [BM: 82], in cryptographic applications the adversary must be assumed to be as malicious as possible, with the only restriction on tests being computation time. A pseudorandom generator can be directly used to design a private key cryptosystem secure against all such adversaries.

In the context of Monte Carlo simulation applications, a typical algorithm uses long random strings, and a typical analysis shows that the algorithm produces a correct answer with high probability if the string it uses is chosen uniformly. In practice, the long random string is not chosen uniformly, as this would require more random bits than it is typically reasonable to produce (and store). Instead, a short random string is stretched into a long string using a simple generator such as a linear congruential generator, and this long string is used by the simulation algorithm. In general, it is hard to directly analyze the simulation algorithm to prove that it produces the correct answer with high probability when the string it uses is produced using such a method. A pseudorandom generator provides a generic solution to this problem. For example, [Yao: 82] shows how pseudorandom generators can be used to reduce the number of random bits needed for any

probabilistic polynomial time algorithm, and thus shows how to perform a deterministic simulation of any polynomial time probabilistic algorithm in subexponential time based on a pseudorandom generator. The results on deterministic simulation were subsequently generalized in [BH: 89], [BFNW: 96].

Since the conditions are rather stringent, it is not easy to come up with a natural candidate for a pseudorandom generator. On the other hand, there seem to be a variety of natural examples of another basic primitive; the one-way function. Informally, f is *one-way* if it is easy to compute but hard on average to invert. If $P=NP$ then there are no one-way functions, and it is not even known if $P \neq NP$ implies there are one-way functions. However, there are many examples of functions that seem to be one-way in practice and that are conjectured to be one-way. Some examples of conjectured one-way functions are the discrete logarithm problem modulo a large randomly chosen prime (see, e.g., [DH: 76]), factoring a number that is the product of two large randomly chosen primes (see, e.g., [RSA: 78]), problems from coding theory (see, e.g., [McEl: 78], [GKL: 93]), and the subset sum problem for appropriately chosen parameters (see, e.g., [IN: 96]).

The paper [BM: 82] is the first to construct a pseudorandom generator based on a one-way function. They introduce an elegant construction that shows how to construct a pseudorandom generator based on the presumed difficulty of the discrete logarithm problem. The paper [Yao: 82] substantially generalizes this result by showing how to construct a pseudorandom generator from any one-way permutation. (Some of the arguments needed in the proof were missing in [Yao: 82] and were later completed by [Levin: 87]. Also, [Levin: 87] conjectured that a much simpler construction would work for the case of one-way permutations, and this was eventually shown in [GL: 89].)

There are several important works that have contributed to the expansion of the conditions on one-way functions under which a pseudorandom generator can be constructed. [GMT: 82] and [Yao: 82] show how to construct a pseudorandom generator based on the difficulty of factoring, and this was substantially simplified in [ACGS: 88]. When f is a one-way permutation, the task of inverting $f(x)$ is to find x . In the case when f is not a permutation, the natural extension of successful inversion to finding any x' such that $f(x') = f(x)$. The paper [Levin: 87] introduces one-way functions which remain one-way after several iterations and shows them to be necessary and sufficient for the construction of a pseudorandom generator. The paper [GKL: 93] shows how to construct a pseudorandom generator from any one-way function with the property that each value in the range of the function has roughly the same number of preimages. This expanded the list of conjectured one-way functions from which pseudorandom generators can be constructed to a variety of non-number theoretic functions, including coding theory problems.

However, the general question of how to construct a pseudorandom generator from a one-way function with no structural properties was left open. This paper resolves this question. We give several successively more intricate constructions, starting with constructions for

one-way functions with a lot of structure and finishing with the constructions for one-way functions with no required structural properties.

The current paper is a combination of the results announced in the conference papers [ILL: 89] and [Hås: 90].

1.1 Concepts and tools

Previous methods, following [BM: 82], rely on constructing a function that has an output bit that is computationally unpredictable given the other bits of the output, but is nevertheless statistically correlated with these other bits. [GL: 89] provide a simple and natural input bit which is hidden from (a padded version of) any one-way function. Their result radically simplifies the previous constructions of pseudorandom generator from one-way permutations, and in addition makes all previous constructions substantially more efficient. We use their result in a fundamental way.

Our overall approach is different in spirit from previous constructions of pseudorandom generators based on one-way functions with special structure. Previous methods rely on iterating the one-way function many times, and from each iteration they extract a computationally unpredictable bit. The approach is to make sure that after many iterations the function is still one-way. In contrast, as explained below in more detail, our approach concentrates on extracting and smoothing entropy in parallel from many independent copies of the one-way function. Our overall construction combines this parallel approach with a standard method for iteratively stretching the output of a pseudorandom generator.

The notion of computational indistinguishability provides one of the main conceptual tools in our paper. Following [GM: 84] and [Yao: 82], we say that two probability distributions \mathcal{D} and \mathcal{E} are *computationally indistinguishable* if no feasible adversary can distinguish \mathcal{D} from \mathcal{E} . In these terms, a pseudorandom generator is intuitively the following: Let g be a polynomial time computable function that maps string of length n to longer strings of length $\ell_n > n$. Let X be a random variable that is uniformly distributed on strings of length n and let Y be a random variable that is uniformly distributed on strings of length ℓ_n . Then, g is a *pseudorandom generator* if $g(X)$ and Y are computationally indistinguishable.

The Shannon entropy of a distribution is a good measure of its information content. A fundamental law of information theory is that the application of a function cannot increase entropy. For example, because X has n bits of entropy, $g(X)$ can also have at most n bits of entropy (see Proposition 2.2.4). The work presented in this paper focuses on a computational analog of Shannon entropy, namely computational entropy. We say the *computational entropy* of $g(X)$ is at least the Shannon entropy of Y if $g(X)$ and Y are computationally indistinguishable. If $g(X)$ is a pseudorandom generator, the

computational entropy of $g(X)$ is greater than the Shannon entropy of its input X , and in this sense g amplifies entropy.

We introduce the following generalizations of a pseudorandom generator based on computational entropy. We say that $g(X)$ is a *pseudoentropy generator* if the computational entropy of $g(X)$ is significantly more than the Shannon entropy of X . We say that $g(X)$ is a *false entropy generator* if the computational entropy of $g(X)$ is significantly more than the Shannon entropy of $g(X)$.

We show how to construct a false entropy generator from any one-way function, a pseudoentropy generator from any false entropy generator and finally a pseudorandom generator from any pseudoentropy generator. (The presentation of these results in the paper is in reverse order.)

We use hash functions and their analysis in a fundamental way in our constructions. This approach has its roots in [GKL : 93]. In [GL : 89], it turns out that the easily computable bit that is hidden is the parity of a random subset of the input bits, i.e., the inner product of the input and a random string. This random inner product can be viewed as a hash function from many bits to one bit.

Due to its importance in such basic algorithms as primality testing, randomness has become an interesting computational resource in its own right. Recently, various studies for extracting good random bits from biased “slightly-random” sources that nevertheless possess a certain amount of entropy have been made; these sources model the imperfect physical sources of randomness, such as Geiger counter noise and Zener diodes, that would have to actually be utilized in real life. (See [Blum : 84], [SV : 86], [Vaz : 87], [VV : 85], [CG : 88], and [McIn : 87].) One of our main technical lemmas, (Lemma 4.5.1), can be viewed as a hashing lemma which is used to manipulate entropy in various ways: it can be viewed as a method for extracting close to uniform random bits from a slightly-random source using random bits as a catalyst.

1.2 Outline

An outline of the paper is as follows:

In Section 2 we give notation, especially related to probability distributions and ensembles. In Section 3, we define the basic primitives used in the paper and a general notion of reduction between primitives. We spend a little more time on this than is conventional in papers on cryptography, since we want to discuss the effects of reductions on security in quantitative terms.

Section 4 introduces the basic mechanisms for finding hidden bits and manipulating en-

tropy with hash functions. The main result of the section is a reduction from a false entropy generator to a pseudorandom generator via a pseudoentropy generator.

In Section 5, we present a construction of a pseudorandom generator from a one-way function where pre-image sizes can be estimated. Although such one-way functions are very common, and so this is an important special case, the main reason for including this is to develop intuition for general one-way functions.

Section 6 presents the most technically challenging construction, that of a false entropy generator from any one-way function. Combined with Section 4, this yields the main result of the paper, the construction of a pseudorandom generator from any one-way function.

In Section 7, we present a somewhat more direct and efficient construction of a pseudorandom generator from any one-way function. It uses the ideas from Sections 4, 5, and 6, but avoids some redundancy involved in combining three generic reductions. Section 8 concludes by placing our results in the context of modern cryptographic complexity.

2 Basic notation

\mathcal{N} is the set of natural numbers. If S is a set then $\#S$ is the number of elements in S . If S and T are sets then $S \setminus T$ is the set consisting of all elements in S that are not in T . If a is a number, then $|a|$ is the absolute value of a , $\lceil a \rceil$ is the smallest integer greater than or equal to a , and $\log(a)$ is the logarithm base two of a .

Let x and y be bit strings. We let $\langle x, y \rangle$ denote the sequence x followed by y , and when appropriate we also view this as the concatenation of x and y . If $x \in \{0, 1\}^n$ then x_i is the i^{th} bit of x , $x_{\{i, \dots, j\}}$ is $\langle x_i, \dots, x_j \rangle$, and $x \oplus y$ is $\langle x_1 \oplus y_1, \dots, x_n \oplus y_n \rangle$.

An $m \times n$ bit matrix x is indicated by $x \in \{0, 1\}^{m \times n}$. We write $x_{i,j}$ to refer to the (i, j) -entry in x . We can also view x as a sequence $x = \langle x_1, \dots, x_m \rangle$ of m strings, each of length n , where in this case x_i is the i^{th} row of the matrix, or we can view x as a bit string of length mn , which is the concatenation of the rows of the matrix.

The \odot operation indicates matrix multiplication over $\text{GF}[2]$. If $x \in \{0, 1\}^n$ appears to the left of \odot then it is considered to be a row vector, and if it appears to the right of \odot it is considered to be a column vector. Thus, if $x \in \{0, 1\}^n$ and $y \in \{0, 1\}^n$ then $x \odot y = \sum_{i=1}^n x_i \cdot y_i \pmod 2$. More generally, if $x \in \{0, 1\}^{\ell \times m}$ and $y \in \{0, 1\}^{m \times n}$ then $x \odot y$ is the $\ell \times n$ bit matrix, where the (i, j) -entry is $r \odot c$, where r is the i^{th} row of x and c is the j^{th} column of y .

2.1 Probability Notation

In general, we use capital and Greek letters to denote random variables and random events. Unless otherwise stated, all random variables are independent of all other random variables.

A distribution \mathcal{D} on a finite set S assigns a probability $\mathcal{D}(x) \geq 0$ to each $x \in S$, and thus $\sum_{x \in S} \mathcal{D}(x) = 1$. We say a random variable X is distributed according to \mathcal{D} on S if for all $x \in S$, $\Pr[X = x] = \mathcal{D}(x)$, and we indicate this by $X \in_{\mathcal{D}} S$. We write $\mathcal{D} : \{0, 1\}^{\ell_n}$ to indicate that \mathcal{D} is supported on strings of length ℓ_n . We sometimes, for convenience, blur the distinction between a random variable and its distribution. If X_1 and X_2 are random variables (that are not necessarily independent), then $(X_1|X_2 = x_2)$ denotes the random variable that takes on value x_1 with the conditional probability $\Pr[X_1 = x_1|X_2 = x_2] = \Pr[X_1 = x_1 \wedge X_2 = x_2] / \Pr[X_2 = x_2]$.

If f is a function mapping S to a set T , then $f(X)$ is a random variable that defines a distribution \mathcal{E} , where for all $y \in T$, $\mathcal{E}(y) = \sum_{x \in S, f(x)=y} \mathcal{D}(x)$. We let $f(\mathcal{D})$ indicate the distribution \mathcal{E} .

We let $X \in_{\mathcal{U}} S$ indicate that X is uniformly distributed in S , i.e., for all $x \in S$, $\Pr[X = x] = 1/|S|$. We let \mathcal{U}_n indicate the uniform distribution on $\{0, 1\}^n$, i.e., X is distributed according to \mathcal{U}_n if $X \in_{\mathcal{U}} \{0, 1\}^n$.

We sometimes want to indicate a random sample chosen from a distribution, and we do this by using the same notation as presented above for random variables except that we use lower case letters, i.e., $x \in_{\mathcal{D}} S$ indicates that x is a fixed element of S chosen according to distribution \mathcal{D} .

If X is a real-valued random variable, then $E[X]$ denotes the expected value X . If E is a probabilistic event, then $\Pr[E]$ denotes the probability that event E occurs.

Definition 2.1.1 (statistical distance) *Let \mathcal{D} and \mathcal{E} be distributions on a set S . The statistical distance between \mathcal{D} and \mathcal{E} is*

$$\mathbf{L}_1(\mathcal{D}, \mathcal{E}) = \sum_{x \in S} |\Pr[\mathcal{D}(x)] - \Pr[\mathcal{E}(x)]| / 2.$$

Proposition 2.1.2 *For any function f with domain S and for any pair of distributions \mathcal{D} and \mathcal{E} on S , $\mathbf{L}_1(f(\mathcal{D}), f(\mathcal{E})) \leq \mathbf{L}_1(\mathcal{D}, \mathcal{E})$.*

2.2 Entropy

The following definition of entropy is from [Shan: 48].

Definition 2.2.1 (information and entropy) Let \mathcal{D} be a distribution on a set S . For each $x \in S$, define the information of x with respect to \mathcal{D} to be $\mathbf{I}_{\mathcal{D}}(x) = -\log(\mathcal{D}(x))$. Let $X \in_{\mathcal{D}} S$. The (Shannon) entropy of \mathcal{D} is $\mathbf{H}(\mathcal{D}) = \mathbf{E}[\mathbf{I}_{\mathcal{D}}(X)]$. Let \mathcal{D}_1 and \mathcal{D}_2 be distributions on S that are not necessarily independent, and let $X_1 \in_{\mathcal{D}_1} S$ and $X_2 \in_{\mathcal{D}_2} S$. Then, the conditional entropy of \mathcal{D}_1 with respect to \mathcal{D}_2 , $\mathbf{H}(\mathcal{D}_1|\mathcal{D}_2)$, is $\mathbf{E}_{x_2 \in_{\mathcal{D}_2} S}[\mathbf{H}(X_1|X_2 = x_2)]$.

We sometimes refer to the entropy $\mathbf{H}(X)$ of random variable X , which is equal to $\mathbf{H}(\mathcal{D})$. We sometimes refer to the conditional entropy $\mathbf{H}(X_1|X_2)$ of X_1 conditioned on X_2 , which is equal to $\mathbf{H}(\mathcal{D}_1|\mathcal{D}_2)$.

The following variant definition of entropy is due to [Renyi: 70].

Definition 2.2.2 (Renyi entropy) Let \mathcal{D} be a distribution on a set S . The Renyi entropy of \mathcal{D} is $\mathbf{H}_{\text{Ren}}(\mathcal{D}) = -\log(\Pr[X = Y])$, where $X \in_{\mathcal{D}} S$ and $Y \in_{\mathcal{D}} S$ are independent.

There are distributions that have arbitrarily large entropy but have only a couple of bits of Renyi entropy.

Proposition 2.2.3 For any distribution \mathcal{D} , $\mathbf{H}_{\text{Ren}}(\mathcal{D}) \leq \mathbf{H}(\mathcal{D})$.

We sometimes use the following proposition implicitly. This proposition shows that a function cannot increase entropy in a statistical sense.

Proposition 2.2.4 Let f be a function and let \mathcal{D} be a distribution on the domain of f . Then, $\mathbf{H}(f(\mathcal{D})) \leq \mathbf{H}(\mathcal{D})$.

The following definition characterizes how much entropy is lost by the application of a function f to the uniform distribution.

Definition 2.2.5 (degeneracy of f) Let $f : \{0, 1\}^n \rightarrow \{0, 1\}^{\ell_n}$ and let $X \in_{\mathcal{U}} \{0, 1\}^n$. The degeneracy of f is $\mathbf{D}_n(f) = \mathbf{H}(X|f(X)) = \mathbf{H}(X) - \mathbf{H}(f(X))$.

2.3 Ensembles

We present all of our definitions and results in asymptotic form. Ensembles are used to make the asymptotic definitions, e.g., to define primitives such as one-way functions and pseudorandom generators, and to define the adversaries that try to break the primitives.

In all cases, we use $n \in \mathcal{N}$ as the index of the ensemble and implicitly the definition and/or result holds for all values of $n \in \mathcal{N}$.

In our definitions of ensembles, the input and output lengths are all polynomially related. To specify this, we use the following.

Definition 2.3.1 (polynomial parameter) *We say parameter k_n is a polynomial parameter if there is a constant $c > 0$ such that for all $n \in \mathcal{N}$,*

$$\frac{1}{cn^c} \leq k_n \leq cn^c.$$

We say k_n is \mathbf{P} -time polynomial parameter if in addition there is a constant $c' > 0$ such that, for all n , k_n is computable in time at most $c'n^{c'}$.

In many uses of a polynomial parameter k_n , k_n is integer-valued, but it is sometimes the case that k_n is real-valued.

Definition 2.3.2 (function ensemble) *We let $f : \{0, 1\}^{t_n} \rightarrow \{0, 1\}^{\ell_n}$ denote a function ensemble, where t_n and ℓ_n are integer-valued \mathbf{P} -time polynomial parameters and where f with respect to n is a function mapping $\{0, 1\}^{t_n}$ to $\{0, 1\}^{\ell_n}$. If f is injective then it is a one-to-one function ensemble. If f is injective and $\ell_n = t_n$ then it is a permutation ensemble. We let $f : \{0, 1\}^{t_n} \times \{0, 1\}^{\ell_n} \rightarrow \{0, 1\}^{m_n}$ denote a function ensemble with two inputs. In this case, we sometimes consider f as being a function of the second input for a fixed value of the first input, in which case we write $f_x(y)$ in place of $f(x, y)$.*

Definition 2.3.3 (\mathbf{P} -time function ensemble) *We say $f : \{0, 1\}^{t_n} \times \{0, 1\}^{\ell_n} \rightarrow \{0, 1\}^{m_n}$ is a T_n -time function ensemble if f is a function ensemble such that, for all $x \in \{0, 1\}^{t_n}$, for all $y \in \{0, 1\}^{\ell_n}$, $f(x, y)$ is computable in time T_n . We say f is a \mathbf{P} -time function ensemble if there is a constant c such that, for all n , $T_n \leq cn^c$. We say f is a mildly non-uniform \mathbf{P} -time function ensemble if it is a \mathbf{P} -time function ensemble except that it has an additional input \mathbf{a}_n called the advice, that is an integer-valued polynomial parameter that is not necessarily \mathbf{P} -time computable.*

These definitions generalize in a natural way to functions with more than two inputs. Sometimes we describe functions that have a variable length inputs or outputs; in these cases we implicitly assume that the string is padded out with a special blank symbol to the appropriate length.

In some of our intermediate reductions, we use certain statistical quantities in order to construct our new primitive. For example, we might use an approximation of the entropy

of a distribution in our construction of pseudoentropy generator. Although in many cases these quantities are not easy to approximate, the number of different approximation values they can take on is small. This is the reason for the definition of a mildly non-uniform \mathbf{P} -time function ensemble in the above definition. In all the definitions we give below, e.g., of one-way functions, false entropy generators, pseudoentropy generators, and pseudorandom generators, there is also an analogous mildly non-uniform version. In Proposition 4.8.1, we show how to remove mild non-uniformity in the final construction of a pseudorandom generator.

Definition 2.3.4 (range and preimages of a function) Let $f : \{0, 1\}^n \rightarrow \{0, 1\}^{\ell_n}$ be a function ensemble. With respect to n , define

$$\text{range}_f = \{f(x) : x \in \{0, 1\}^n\}.$$

For each $y \in \text{range}_f$, define

$$\text{pre}_f(y) = \{x \in \{0, 1\}^n : f(x) = y\}.$$

Definition 2.3.5 (regular function ensemble) We say function ensemble $f : \{0, 1\}^n \rightarrow \{0, 1\}^{\ell_n}$ is σ_n -regular if $\#\text{pre}_f(y) = \sigma_n$ for all $y \in \text{range}_f$.

Definition 2.3.6 ($\tilde{\mathbf{D}}_f$) Let $f : \{0, 1\}^n \rightarrow \{0, 1\}^{\ell_n}$ be a \mathbf{P} -time function ensemble. For $z \in \text{range}_f$, define the approximate degeneracy of z as

$$\tilde{\mathbf{D}}_f(z) = \left\lceil \log(\#\text{pre}_f(z)) \right\rceil.$$

Notice that $\tilde{\mathbf{D}}_f(z)$ is an approximation to within an additive factor of 1 of the quantity $n - \mathbf{I}_{f(X)}(z)$. Furthermore, $\mathbf{E}[\tilde{\mathbf{D}}_f(f(X))]$ is within an additive factor of 1 of the degeneracy of f . If f is a σ_n -regular function then, for each $z \in \text{range}_f$, $\tilde{\mathbf{D}}_f(z)$ is within an additive factor of 1 of $\log(\sigma_n)$, which is the degeneracy of f .

Definition 2.3.7 (probability ensemble) We let $\mathcal{D} : \{0, 1\}^{\ell_n}$ denote a probability ensemble, where ℓ_n is an integer-valued \mathbf{P} -time polynomial parameter and where \mathcal{D} with respect to n is a probability distribution on $\{0, 1\}^{\ell_n}$.

Definition 2.3.8 (\mathbf{P} -samplable probability ensemble) We let $\mathcal{D} : \{0, 1\}^{\ell_n}$ denote a probability ensemble that, with respect to n , is a distribution on $\{0, 1\}^{\ell_n}$ that can be generated from a random string of length r_n for some r_n , i.e., there is a function ensemble $f : \{0, 1\}^{r_n} \rightarrow \{0, 1\}^{\ell_n}$ such that if $X \in_{\mathcal{U}} \{0, 1\}^{r_n}$ then $f(X)$ has the distribution \mathcal{D} . We say \mathcal{D} is T_n -samplable probability ensemble if, for all $x \in \{0, 1\}^{r_n}$, $f(x)$ is computable in time T_n . We say \mathcal{D} is \mathbf{P} -samplable if f is a \mathbf{P} -time function ensemble, and \mathcal{D} is mildly non-uniformly \mathbf{P} -samplable if f is a mildly non-uniform \mathbf{P} -time function ensemble.

Definition 2.3.9 (copies of functions and ensembles) Let k_n be integer-valued \mathbf{P} -time polynomial parameter. If $\mathcal{D} : \{0, 1\}^{\ell_n}$ is a probability ensemble then $\mathcal{D}^{k_n} : \{0, 1\}^{\ell_n k_n}$ is the probability ensemble where, with respect to parameter n , \mathcal{D}^{k_n} consists of the concatenation of k_n independent copies of \mathcal{D} . Similarly, if $f : \{0, 1\}^{m_n} \rightarrow \{0, 1\}^{\ell_n}$ is a function ensemble then $f^{k_n} : \{0, 1\}^{m_n k_n} \rightarrow \{0, 1\}^{\ell_n k_n}$ is the function ensemble where, for $y \in \{0, 1\}^{k_n \times m_n}$,

$$f^{k_n}(y) = \langle f(y_1), \dots, f(y_{k_n}) \rangle,$$

3 Definitions of primitives and reductions

Primitives described in this paper include one-way functions and pseudorandom generators. The primitives we describe can be used in cryptographic applications, but are also useful as described in the introduction in other applications. In the definition of the primitives, we need to describe what it means for the primitive to be secure against an attack by an adversary. We first introduce adversaries and security, and then describe the basic primitives that we use thereafter.

3.1 Adversaries and security

An adversary is, for example, trying to invert a one-way function or trying to distinguish the output of a pseudorandom generator from a truly random string. The time-success ratio of a particular adversary is a measure of its ability to break the cryptographic primitive. (Hereafter, we use “primitive” in place of the more cumbersome and sometimes misleading phrase “cryptographic primitive”.) The security of a primitive is a lower bound on the time-success ratio of *any* adversary to break the primitive.

In the constructions of some primitives, we allow both private and public inputs. A *public input* is part of the output of the primitive and is known to the adversary at the time it tries to break the primitive. When we construct one primitive based on another, the constructed primitive often has public inputs. At first glance it could seem that these public inputs are not useful because an adversary knows them at the time it tries to break the constructed primitive. On the contrary, public inputs turn out to be quite useful. Intuitively, this is because their value is randomly chosen, and the adversary cannot a priori build into its breaking strategy a strategy for all possible values.

The *private input* to a primitive is not directly accessible to the adversary. The security parameter of a primitive is the length of its private input. This is because the private input to the primitive is what is kept secret from the adversary, and thus it makes sense to measure the success of the adversary in terms of this.

Definition 3.1.1 (breaking adversary and security) An adversary A is a function ensemble. The time-success ratio of A for an instance f of a primitive is defined as $\mathbf{R}_{t_n} = T_n/sp_n(A)$, where t_n is the length of the private input to f , and where T_n is the worst case expected running time of A over all instances parameterized by n , and $sp_n(A)$ is the success probability of A for breaking f . In this case, we say A is \mathbf{R} -breaking adversary for f . We say f is \mathbf{R} -secure if there is no \mathbf{R} -breaking adversary for f .

A mildly non-uniform adversary for a mildly non-uniform \mathbf{P} -time function ensemble f that has advice \mathbf{a}_n is a function ensemble A which is given \mathbf{a}_n as an additional input. The success probability and time-success ratio for a mildly non-uniform adversary is the same as for uniform adversaries.

The definition of the success probability $sp_n(A)$ for f depends on the primitive in question, i.e., this probability is defined when the primitive is defined. Intuitively, the smaller the time-success ratio of an adversary for a primitive, the better the adversary is able to break the primitive, i.e., it uses less time and/or has a larger success probability.

The above definitions are a refinement of definitions that appear in the literature. Previously, an adversary was considered to be breaking if it ran in polynomial time and had inverse polynomial success probability. The advantage of the definition introduced here is that it is a more precise characterization of the security of a primitive. This is important because different applications require different levels of security. For some applications polynomial security is enough (e.g., $\mathbf{R}_{t_n} = t_n^{10}$) and for other applications better security is crucial (e.g., $\mathbf{R}_{t_n} = 2^{\log^2(t_n)}$, or even better $\mathbf{R}_{t_n} = 2^{\sqrt{t_n}}$).

3.2 One-way function

Definition 3.2.1 (one-way function) Let $f : \{0, 1\}^{t_n} \rightarrow \{0, 1\}^{\ell_n}$ be a \mathbf{P} -time function ensemble and let $X \in_{\mathcal{U}} \{0, 1\}^{t_n}$. The success probability of adversary A for inverting f is

$$sp_n(A) = \Pr[f(A(f(X))) = f(X)].$$

Then, f is a \mathbf{R} -secure one-way function if there is no \mathbf{R} -breaking adversary for f .

A function cannot be considered to be “one-way” in any reasonable sense in case the time to invert it is smaller than the time to evaluate it in the forward direction. Thus, for example, if there is an $\mathcal{O}(t_n)$ -breaking adversary for f then it is not secure at all. On the other hand, an exhaustive adversary that tries all possible inputs to find an inverse is $t_n^{\mathcal{O}(1)} \cdot 2^{t_n}$ -breaking. Thus, the range of securities that can be hoped for fall between these two extremes.

3.3 Pseudorandom generator

The following definition can be thought of as the computationally restricted adversary definition of statistical distance. The original idea is from [GM: 84] and [Yao: 82].

Definition 3.3.1 (computationally indistinguishable) *Let $\mathcal{D} : \{0, 1\}^{\ell_n}$ and $\mathcal{E} : \{0, 1\}^{\ell_n}$ be probability ensembles. The success probability of adversary A for distinguishing \mathcal{D} and \mathcal{E} is*

$$sp_n(A) = |\Pr[A(X) = 1] - \Pr[A(Y) = 1]|$$

where X has distribution \mathcal{D} and Y has distribution \mathcal{E} . \mathcal{D} and \mathcal{E} are \mathbf{R} -secure computationally indistinguishable if there is no \mathbf{R} -breaking adversary for distinguishing \mathcal{D} and \mathcal{E} .

The following alternative definition of computationally indistinguishable more accurately reflects the tradeoff between the running time of the adversary and its success probability. In the alternative definition, success probability is defined as $sp'_n(A) = (sp_n(A))^2$. This is because it takes $1/sp'_n(A)$ trials in order to approximate $sp_n(A)$ to within a constant factor.

Definition 3.3.2 (computationally indistinguishable (alternative)) *Exactly the same as the original definition, except the success probability of adversary A is $sp'_n(A) = (sp_n(A))^2$.*

In all cases except where noted, the strength of the reduction is the same under either definition of computationally indistinguishable, and we find it easier to work with the first definition. However, there are a few places where we explicitly use the alternative definition to be able to claim the reduction is linear-preserving.

Strictly speaking, there are no private inputs in the above definition, and thus by default we use n as the security parameter. However, in a typical use of this definition, \mathcal{D} is the distribution defined by the output of a \mathbf{P} -time function ensemble (and thus \mathcal{D} is \mathbf{P} -samplable), in which case the length of the private input to this function ensemble is the security parameter. In some circumstances, it is important that both \mathcal{D} and \mathcal{E} are \mathbf{P} -samplable, e.g., this is the case for Proposition 4.6.2.

The paper [Yao: 82] originally gave the definition of a pseudorandom generator as below, except that we parameterize security more precisely.

Definition 3.3.3 (pseudorandom generator) *Let $g : \{0, 1\}^{t_n} \rightarrow \{0, 1\}^{\ell_n}$ be a \mathbf{P} -time function ensemble where $\ell_n > t_n$. Then, g is a \mathbf{R} -secure pseudorandom generator if the probability ensembles $g(\mathcal{U}_{t_n})$ and \mathcal{U}_{ℓ_n} are \mathbf{R} -secure computationally indistinguishable.*

The definition of a pseudorandom generator only requires the generator to stretch the input by at least one bit. The following proposition provides a general way to produce a pseudorandom generator that stretches by many bits from a pseudorandom generator that stretches by at least one bit. This proposition appears in [BH : 89] and is due to O. Goldreich and S. Micali.

Proposition 3.3.4 *Suppose $g : \{0, 1\}^n \rightarrow \{0, 1\}^{n+1}$ is a pseudorandom generator that stretches by one bit. Define $g^{(1)}(x) = g(x)$, and inductively, for all $i \geq 1$,*

$$g^{(i+1)}(x) = \langle g(g^{(i)}(x)_{\{1, \dots, n\}}), g^{(i)}(x)_{\{n+1, \dots, n+i\}} \rangle.$$

Let k_n be an integer-valued \mathbf{P} -time polynomial parameter. Then, $g^{(k_n)}$ is a pseudorandom generator. The reduction is linear-preserving.

On page 16 we give a formal definition of reduction and what it means to be linear-preserving, but intuitively it means that $g^{(k_n)}$ as a pseudorandom generator is almost as secure as pseudorandom generator g .

3.4 Pseudoentropy and false-entropy generators

The definitions in this subsection introduce new notions (interesting in their own right) which we use as intermediate steps in our constructions.

The difference between a pseudorandom generator and a pseudoentropy generator is that the output of a pseudoentropy generator doesn't have to be computationally indistinguishable from the uniform distribution, instead it must be computationally indistinguishable from some probability ensemble \mathcal{D} that has more entropy than the input to the generator. Thus, a pseudoentropy generator still amplifies randomness so that the output randomness is more computationally than the input randomness, but the output randomness is no longer necessarily uniform.

Definition 3.4.1 (computational entropy) *Let $f : \{0, 1\}^{t_n} \rightarrow \{0, 1\}^{\ell_n}$ be a \mathbf{P} -time function ensemble and let s_n be a polynomial parameter. Then, f has \mathbf{R} -secure computational entropy s_n if there is a \mathbf{P} -time function ensemble $f' : \{0, 1\}^{m_n} \rightarrow \{0, 1\}^{\ell_n}$ such that $f(\mathcal{U}_{t_n})$ and $f'(\mathcal{U}_{m_n})$ are \mathbf{R} -secure computationally indistinguishable and $\mathbf{H}(f'(\mathcal{U}_{m_n})) \geq s_n$.*

Definition 3.4.2 (pseudoentropy generator) *Let $f : \{0, 1\}^{t_n} \rightarrow \{0, 1\}^{\ell_n}$ be a \mathbf{P} -time function ensemble and let s_n be a polynomial parameter. Then, f is a \mathbf{R} -secure pseudoentropy generator with pseudoentropy s_n if $f(\mathcal{U}_{t_n})$ has \mathbf{R} -secure computational entropy $t_n + s_n$.*

If f is a pseudorandom generator then it is easy to see that it is also a pseudoentropy generator. This is because $f(\mathcal{U}_{t_n})$ and \mathcal{U}_{ℓ_n} are computationally indistinguishable and by definition of a pseudorandom generator, $\ell_n > t_n$. Consequently, $\mathbf{H}(\mathcal{U}_{\ell_n}) = \ell_n \geq t_n + 1$, i.e., f is a pseudoentropy generator with pseudoentropy at least 1.

A false entropy generator is a further generalization of pseudoentropy generator. A false entropy generator doesn't necessarily amplify the input randomness, it just has the property that the output randomness is computationally more than it is statistically.

Definition 3.4.3 (false entropy generator) *Let $f : \{0, 1\}^{t_n} \rightarrow \{0, 1\}^{\ell_n}$ be a \mathbf{P} -time function ensemble and let s_n be a polynomial parameter. Then, f is a \mathbf{R} -secure false entropy generator with false-entropy s_n if $f(\mathcal{U}_{t_n})$ has \mathbf{R} -secure computational entropy $\mathbf{H}(f(\mathcal{U}_{t_n})) + s_n$.*

Note that, in the definition of computational entropy, the function ensemble f' that is computationally indistinguishable from f is required to be \mathbf{P} -time computable. This is consistent with the definition of a pseudorandom generator, where the distribution that the pseudorandom generator is indistinguishable from is the uniform distribution. There is also a non-uniform version of computational entropy where f' is not necessarily \mathbf{P} -time computable, and corresponding non-uniform versions of a pseudoentropy generator and false entropy generator. It turns out to be easier to construct a false entropy generator f where f' is not necessarily \mathbf{P} -time computable from a one-way function than it is to construct a false entropy generator f where f' is \mathbf{P} -time samplable. Using this approach and a non-uniform version of Proposition 4.6.2, [ILL : 89] describe a non-uniform reduction from a one-way function to a pseudorandom generator. However, a uniform reduction using Proposition 4.6.2 requires that f' be \mathbf{P} -time computable. Thus, one of the main difficulties in our constructions below is to build a false entropy generator f where f' is \mathbf{P} -time computable.

3.5 Hidden bits

In the construction of a pseudorandom generator from a one-way function, one of the key ideas is to construct from the one-way function another function which has an output bit that is computationally unpredictable from the other output bits (it is “hidden”) and yet statistically somewhat predictable from the other output bits. This idea is used in the original construction of a pseudorandom generator from the discrete logarithm problem [BM: 82] and has been central to all such constructions since that time.

Definition 3.5.1 (hidden bit) *Let $f : \{0, 1\}^{t_n} \rightarrow \{0, 1\}^{\ell_n}$ and $b : \{0, 1\}^{t_n} \rightarrow \{0, 1\}$ be \mathbf{P} -time function ensembles. Let $\mathcal{D} : \{0, 1\}^{t_n}$ be a \mathbf{P} -samplable probability ensemble,*

let $X \in_{\mathcal{D}} \{0, 1\}^{t_n}$, and let $\beta \in_{\mathcal{U}} \{0, 1\}$. Then, $b(X)$ is \mathbf{R} -secure hidden given $f(X)$ if $\langle f(X), b(X) \rangle$ and $\langle f(X), \beta \rangle$ are \mathbf{R} -secure computationally indistinguishable.

3.6 Reductions

All of the results presented in this paper involve a reduction from one type of primitive to another.

We make the following definitions to quantify the strength of reductions. The particular parameterization of security and the different quantitative measures of the security preserving properties of a reduction are derived from [Luby: 96], [HL: 92].

Intuitively, a reduction constructs from a first primitive f on inputs of length t_n a second primitive $g^{(f)}$ on inputs of length t'_n . The reduction also specifies an oracle TM $M^{(\cdot)}$ such that if there is an adversary A for breaking $g^{(f)}$ then $M^{(A)}$ is an adversary for breaking f . How much security is preserved by the reduction is parameterized by \mathcal{S} .

Definition 3.6.1 (reduction) Let t_n and t'_n be polynomial parameters and let $\mathcal{S} : \mathcal{N} \times \mathbb{R}^+ \rightarrow \mathbb{R}^+$. An \mathcal{S} -reduction from primitive 1 to primitive 2 is a pair of oracle TMs $g^{(\cdot)}$ and $M^{(\cdot)}$ so that,

- For each \mathbf{P} -time function ensemble $f : \{0, 1\}^{t_n} \rightarrow \{0, 1\}^{\ell_n}$ that instantiates primitive 1, $g^{(f)} : \{0, 1\}^{t'_n} \rightarrow \{0, 1\}^{\ell'_n}$ instantiates primitive 2.
- $g^{(f)}$ is a \mathbf{P} -time function ensemble, and on inputs of length t'_n , only makes calls to f on inputs of length t_n .
- Suppose A is an adversary with time-success ratio $\mathbf{R}'_{t'_n}$ for $g^{(f)}$ on inputs of length t'_n . Define $\mathbf{R}_{t_n} = \mathcal{S}(n, \mathbf{R}'_{t'_n})$. Then, $M^{(A)}$ is an adversary with time-success ratio \mathbf{R}_{t_n} for f on inputs of length t_n .

To discuss the security preserving properties of the reduction, we compare how well A breaks $g^{(f)}$ to how well $M^{(A)}$ breaks f on inputs of similar size. We say the reduction is

- linear-preserving if: $\mathbf{R}_N = N^{\mathcal{O}(1)} \cdot \mathcal{O}(\mathbf{R}'_N)$.
- poly-preserving if: $\mathbf{R}_N = N^{\mathcal{O}(1)} \cdot \mathbf{R}'_{\mathcal{O}(N)}^{\mathcal{O}(1)}$.
- weak-preserving if: $\mathbf{R}_N = N^{\mathcal{O}(1)} \cdot \mathbf{R}'_{N^{\mathcal{O}(1)}}^{\mathcal{O}(1)}$.

A mildly non-uniform reduction has the same properties except that $g^{(\cdot)}$ and $M^{(\cdot)}$ are both allowed access to an integer-valued polynomial parameter \mathbf{a}_n that depends on f . The same notions of security preservation apply to mildly non-uniform reductions.

f can always be broken in time exponential in t_n . Therefore, if $\mathbf{R}'_{t'_n} \geq 2^{t_n}$, or even $\mathbf{R}'_{t'_n} \geq 2^{t_n^{\Omega(1)}} = 2^{n^{\Omega(1)}}$ in the case of a weak-preserving reduction, $M^{(A)}$ can ignore the oracle and break f by brute force. Therefore, we can assume without loss of generality that $\mathbf{R}'_{t'_n} \leq 2^{t_n}$.

Obvious from the definition of reduction are the following propositions, that say that security is preserved by reductions, and that reductions can be composed:

Proposition 3.6.2 *If $(g^{(\cdot)}, M^{(\cdot)})$ is a (mildly non-uniform) \mathcal{S} -reduction from primitive 1 to primitive 2 and f is a (mildly non-uniform) \mathbf{P} -time function ensemble that instantiates primitive 1 with security \mathbf{R}_{t_n} , then $g^{(f)}$ is a (mildly non-uniform) \mathbf{P} -time function ensemble that instantiates primitive 2 with security $\mathbf{R}'_{t'_n}$.*

Proposition 3.6.3 *If $(g_1^{(\cdot)}, M_1^{(\cdot)})$ is a (mildly non-uniform) \mathcal{S}_1 -reduction from primitive 1 to primitive 2, and if $(g_2^{(\cdot)}, M_2^{(\cdot)})$ is a (mildly non-uniform) \mathcal{S}_2 -reduction from primitive 2 to primitive 3, then $(g_1^{(g_2^{(\cdot)})}, M_1^{(M_2^{(\cdot)})})$, is a (mildly non-uniform) \mathcal{S} -reduction from primitive 1 to primitive 3, where $\mathcal{S}(N, R) = \mathcal{S}_2(N, \mathcal{S}_1(N, R))$.*

Although we phrase our definitions in terms of asymptotic complexity, one can easily interpret them for fixed length inputs in the context of an actual implementation, just as one does for algorithm analysis.

Clearly, in standard situations, $t'_n \geq t_n$ and $\mathbf{R}_{t_n} \geq \mathbf{R}'_{t'_n}$, and the closer these two inequalities are to equalities the more the security of f is transferred to g . We now describe how the slack in these inequalities affects the security preserving properties of the reduction.

The number of calls $M^{(A)}$ makes to A is invariably either a constant or depends polynomially on the time-success ratio of A , and thus \mathbf{R}_{t_n} is at most polynomial in $\mathbf{R}'_{t'_n}$. The slackness in this inequality turns out not to be the major reason for a loss in security in the reduction, instead it primarily depends on how much larger t'_n is than t_n . If t'_n is much larger than t_n then \mathbf{R}_{t_n} is much larger as a function of t_n than $\mathbf{R}'_{t'_n}$ is as a function of t'_n . We can formalize this as follows.

Proposition 3.6.4

- If $t'_n = t_n$, $M^{(A)}$ runs in time polynomial in n (not counting the running time of A), and $sp_n(M^{(A)}) = sp_n(A)/n^{\mathcal{O}(1)}$, then the reduction is linear-preserving.
- If $t'_n = \mathcal{O}(t_n)$, $M^{(A)}$ runs in time polynomial in $\mathbf{R}'_{t'_n}$, and $sp_n(M^{(A)}) = sp_n^{\mathcal{O}(1)}(A)/n^{\mathcal{O}(1)}$, then the reduction is poly-preserving.
- If $t'_n = t_n^{\mathcal{O}(1)}$, $M^{(A)}$ runs in time polynomial in $\mathbf{R}'_{t'_n}$, and $sp_n(M^{(A)}) = sp_n^{\mathcal{O}(1)}(A)/n^{\mathcal{O}(1)}$, then the reduction is weak-preserving.

It is important to design the strongest reduction possible. The techniques described in this paper can be directly used to yield poly-preserving reductions from regular or nearly regular (with polynomial time computable degree of regularity) one-way functions to pseudorandom generators [Luby: 96], and this covers almost all of the conjectured one-way functions. However, the reduction for general one-way functions is only weak-preserving.

4 Hidden bits, hash functions, and computational entropy

4.1 Constructing a hidden bit

How do we go about constructing a function such that one of its output bits is computationally unpredictable yet statistically correlated with its other output bits? The following fundamental proposition of [GL: 89] (strengthened in [Levin: 93]) provides the answer.

Proposition 4.1.1 *Let $f : \{0, 1\}^n \rightarrow \{0, 1\}^{\ell_n}$ be a one-way function. Then, $X \odot R$ is hidden given $\langle f(X), R \rangle$, where $X, R \in_{\mathcal{U}} \{0, 1\}^n$. The reduction is linear-preserving with respect to the alternative definition of computationally indistinguishable.*

Proposition 4.1.1 presents an elegant, simple and general method of obtaining a hidden bit from a one-way function. We need the following stronger proposition of [GL: 89] (see also [Levin: 93]) in some of our proofs.

Proposition 4.1.2 *There is an oracle TM M with the following properties. Let A be any adversary that accepts as input n bits and outputs a single bit. Then, $M^{(A)}$ on input parameter $\delta_n > 0$ outputs a list \mathcal{L} of n -bit strings with the following property: for any fixed $x \in \{0, 1\}^n$, if it is the case that*

$$|\Pr[A(R) = x \odot R] - \Pr[A(R) \neq x \odot R]| \geq \delta_n,$$

where $R \in_{\mathcal{U}} \{0, 1\}^n$, then, with probability at least $1/2$, it is the case that $x \in \mathcal{L}$. (The probability here only depends on the values of the random bits used by $M^{(A)}$.) The running

time of $M^{(A)}$ is polynomial in n , $1/\delta_n$ and the running time of A . Also, the number of n -bit strings in \mathcal{L} is bounded by $\mathcal{O}(1/\delta_n^2)$.

The following proposition is an immediate consequence of Propositions 4.1.1 and Definition 3.5.1.

Proposition 4.1.3 *Let $f : \{0, 1\}^n \rightarrow \{0, 1\}^{\ell_n}$ be a one-way function. Then, $\langle f(X), R, X \odot R \rangle$ and $\langle f(X), R, \beta \rangle$ are computationally indistinguishable, where $X, R \in_{\mathcal{U}} \{0, 1\}^n$ and $\beta \in_{\mathcal{U}} \{0, 1\}$. The reduction is linear-preserving with respect to the alternative definition of computationally indistinguishable.*

4.2 One-way permutation to a pseudorandom generator

We describe a way to construct a pseudorandom generator from any one-way permutation which is substantially simpler (and has stronger security preserving properties) than the original construction of [Yao: 82]. The construction and proof described here is due to [GL: 89].

Proposition 4.2.1 *Let $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a one-way permutation. Let $x, r \in \{0, 1\}^n$ and define \mathbf{P} -time function ensemble $g(x, r) = \langle f(x), r, x \odot r \rangle$. Then, g is a pseudorandom generator. The reduction is linear-preserving with respect to the alternative definition of computationally indistinguishable.*

PROOF: Let $X, R \in_{\mathcal{U}} \{0, 1\}^n$, and $\beta \in_{\mathcal{U}} \{0, 1\}$. Because f is a permutation, $\langle f(X), R, \beta \rangle$ is the uniform distribution on $\{0, 1\}^{2n+1}$. By Proposition 4.1.3, $g(X, R)$ and $\langle f(X), R, \beta \rangle$ are computationally indistinguishable, where the reduction is linear-preserving with respect to the alternative definition of computationally indistinguishable. ■

The reason Proposition 4.2.1 works when f is a permutation is because:

- (1) $f(X)$ is uniformly distributed and hence already looks random.
- (2) For any $x \in \{0, 1\}^n$, $f(x)$ uniquely determines x . So no entropy is lost by the application of f ,

For a general one-way function neither (1) nor (2) necessarily holds. *Intuitively, the rest of the paper constructs a one-way function with properties (1) and (2) from a general one-way function. This is done by using hash functions to smooth the entropy of $f(X)$ to make it more uniform, and to recapture the entropy of X lost by the application of $f(X)$.*

Proposition 4.2.1 produces a pseudorandom generator that only stretches the input by one bit. To construct a pseudorandom generator that stretches by many bits, combine this with the construction described previously in Proposition 3.3.4.

4.3 One-to-one one-way function to a pseudoentropy generator

We now describe a construction of a pseudoentropy generator from any one-to-one one-way function. This construction, together with Theorem 4.6.4, yields a pseudorandom generator from any one-to-one one-way function. The overall construction is different in spirit than the original construction of [GKL: 93]: it illustrates how to construct a pseudoentropy generator in a particularly simple way using [GL: 89]. Although the assumptions and the consequences are somewhat different, the construction is the same as described in Proposition 4.2.1.

Proposition 4.3.1 *Let $f : \{0, 1\}^n \rightarrow \{0, 1\}^{\ell_n}$ be a one-to-one one-way function. Let $x, r \in \{0, 1\}^n$ and define \mathbf{P} -time function ensemble $g(x, r) = \langle f(x), r, x \odot r \rangle$. Then, g is a pseudoentropy generator with pseudoentropy 1. The reduction is linear-preserving with respect to the alternative definition of computationally indistinguishable.*

PROOF: Let $X, R \in_{\mathcal{U}} \{0, 1\}^n$ and $\beta \in_{\mathcal{U}} \{0, 1\}$. Proposition 4.1.3 shows that $g(X, R)$ and $\langle f(X), R, \beta \rangle$ are computationally indistinguishable, where the reduction is linear-preserving with respect to the alternative definition of computationally indistinguishable. Because f is a one-to-one function and β is a random bit, $\mathbf{H}(f(X), R, \beta) = 2n + 1$, and thus $g(X, R)$ has pseudoentropy 1. ■

Note that it is not possible to argue that g is a pseudorandom generator. For example, let $f(x) = \langle 0, f'(x) \rangle$ where f' is a one-way permutation. Then, f is a one-to-one one-way function and yet $g(X, R) = \langle f(X), R, X \odot R \rangle$ is not a pseudorandom generator, because the first output bit of g is zero independent of its inputs, and thus its output can easily be distinguished from a uniformly chosen random string.

4.4 Universal hash functions

The concept of a universal hash function, introduced in [CW: 79], has proved to have far reaching and a broad spectrum of applications in the theory of computation.

Definition 4.4.1 (universal hash functions) *Let $h : \{0, 1\}^{\ell_n} \times \{0, 1\}^n \rightarrow \{0, 1\}^{m_n}$ be a \mathbf{P} -time function ensemble. Recall from definition 2.3.2 that for fixed $y \in \{0, 1\}^{\ell_n}$, we*

view y as describing a function $h_y(\cdot)$ that maps n bits to m_n bits. Then, h is a (pairwise independent) universal hash function if, for all $x \in \{0, 1\}^n$, $x' \in \{0, 1\}^n \setminus \{x\}$, for all $a, a' \in \{0, 1\}^{m_n}$,

$$\Pr[(h_Y(x) = a) \text{ and } (h_Y(x') = a')] = 1/2^{2m_n},$$

where $Y \in_{\mathcal{U}} \{0, 1\}^{\ell_n}$.

Intuitively, a universal hash function has the property that every distinct pair x and x' are mapped randomly and independently with respect to Y .

In all of our constructions of function ensembles using universal hash functions, the description of the hash function y is viewed as a public input to the function ensemble, and thus is also part of the output. The following construction is a universal hash function is due to [CW: 79].

Definition 4.4.2 (matrix construction) *Let*

$$h : \{0, 1\}^{(n+1)m_n} \times \{0, 1\}^n \rightarrow \{0, 1\}^{m_n}$$

be the following \mathbf{P} -time function ensemble. For $x \in \{0, 1\}^n$ and $y \in \{0, 1\}^{(n+1)m_n}$, $h_y(x) = \langle x, 1 \rangle \odot y$.

We concatenate a 1 to x in the above definition to cover the case when $x = 0^n$. Hereafter, whenever we refer to universal hash functions, one can think of the construction given above. However, any universal hash function that satisfies the required properties may be used. We note that there are more efficient hash functions in terms of number of bits used in specification. One such example is using Toeplitz matrices (see for example [GL: 89] or [Levin: 93]). A Toeplitz matrix is a matrix which is constant on any diagonal, and thus to specify an $n \times m$ Toeplitz matrix we can specify values for the $m + n - 1$ diagonals, This is the simplest bit-efficient construction of a universal hash function, so we adopt it as the default for the remaining paper.

4.5 Smoothing distributions with hashing

The following lemma is a key component in most of the subsequent reductions we describe.

Lemma 4.5.1 *Let $\mathcal{D} : \{0, 1\}^n$ be a probability ensemble that has Renyi entropy at least m_n . Let e_n be a positive integer valued parameter. Let $h : \{0, 1\}^{\ell_n} \times \{0, 1\}^n \rightarrow \{0, 1\}^{m_n - 2e_n}$ be a universal hash function. Let $X \in_{\mathcal{D}} \{0, 1\}^n$, $Y \in_{\mathcal{U}} \{0, 1\}^{\ell_n}$, and $Z \in_{\mathcal{U}} \{0, 1\}^{m_n - 2e_n}$. Then*

$$\mathbf{L}_1(\langle h_Y(X), Y \rangle, \langle Z, Y \rangle) \leq 2^{-(e_n+1)}.$$

This lemma is a generalization of a lemma that appears in [Sip: 83]. There, \mathcal{D} is the uniform distribution on a set $S \subseteq \{0, 1\}^n$ with $\#S = 2^{m_n}$. The papers [McIn: 87] and [BBR: 88] also proved similar lemmas. For the special case of linear hash functions, this lemma can be derived from [GL: 89] by considering unlimited adversaries. A generalization to a broader class of hash functions appears in [IZ: 89].

The lemma can be interpreted as follows: The universal hash function smooths out the Renyi entropy of X to the almost uniform distribution on bit strings of length almost m_n . The integer parameter e_n controls the tradeoff between the uniformity of the output bits of the universal hash function and the amount of entropy lost in the smoothing process. Thus, we have managed to convert almost all the Renyi entropy of X into uniform random bits while maintaining our original supply of random bits Y .

PROOF: Let $\ell = \ell_n$, $e = e_n$ and $m = m_n$ and $s = m - 2e$. For all $y \in \{0, 1\}^\ell$, $a \in \{0, 1\}^s$ and $x \in \{0, 1\}^n$, define $\chi(h_y(x) = a) = 1$ if $h_y(x) = a$ and 0 otherwise. We want to show that

$$\mathbb{E}_Y \left[\sum_{a \in \{0, 1\}^s} |\mathbb{E}_X[\chi(h_Y(X) = a)] - 2^{-s}| \right] / 2 \leq 2^{-(e+1)}.$$

We show below that for all $a \in \{0, 1\}^s$,

$$\mathbb{E}_Y [|\mathbb{E}_X[\chi(h_Y(X) = a)] - 2^{-s}|] \leq 2^{-(s+e)},$$

and from this the proof follows.

For any random variable Z , $\mathbb{E}[|Z|^2] \geq \mathbb{E}[Z]^2$ by Jensen's Inequality. Letting $Z = \mathbb{E}[\chi(h_Y(X) = a)] - 2^{-s}$, we see that it is sufficient to show for all $a \in \{0, 1\}^s$,

$$\mathbb{E}_Y [(\mathbb{E}_X[\chi(h_Y(X) = a)] - 2^{-s})^2] \leq 2^{-2(s+e)}.$$

Let $X' \in_{\mathcal{D}} \{0, 1\}^n$. Using some elementary expansion of terms, and rearrangements of summation, we can write the above as

$$\mathbb{E}_{X, X'} [\mathbb{E}_Y [(\chi(h_Y(X) = a) - 2^{-s})(\chi(h_Y(X') = a) - 2^{-s})]].$$

For each fixed value of X to x and X' to x' , where $x \neq x'$, the expectation with respect to Y is zero because of the pairwise independence property of universal hash functions. For each fixed value of X to x and X' to x' , where $x = x'$,

$$\mathbb{E} [(\chi(h_Y(x) = a) - 2^{-s})^2] = 2^{-s}(1 - 2^{-s}) \leq 2^{-s}.$$

Because the Renyi entropy of \mathcal{D} is at least m , it follows that $\Pr[X = X'] \leq 2^{-m}$. Thus, the entire sum is at most $2^{-(m+s)}$ which is equal to $2^{-2(s+e)}$ by the definition of s . \blacksquare

In this lemma, \mathcal{D} is required to have Renyi entropy at least m_n . In many of our applications, the distribution in question has at least Renyi entropy m_n , and thus the lemma

applies because of Proposition 2.2.3. For other applications, we need to work with Shannon entropy. The following technical result due to [Shan: 48] allows us to convert Shannon entropy to Renyi entropy by looking at product distributions.

Proposition 4.5.2 *Let k_n be an integer-valued polynomial parameter.*

- *Let $\mathcal{D} : \{0, 1\}^n$ be a probability ensemble. There is a probability ensemble $\mathcal{E} : \{0, 1\}^{nk_n}$ satisfying:*
 - $\mathbf{H}_{\text{Ren}}(\mathcal{E}) \geq k_n \mathbf{H}(\mathcal{D}) - nk_n^{2/3}$.
 - $\mathbf{L}_1(\mathcal{E}, \mathcal{D}^{k_n}) \leq 2^{-k_n^{1/3}}$.
- *Let $\mathcal{D}_1 : \{0, 1\}^n$ and $\mathcal{D}_2 : \{0, 1\}^n$ be not necessarily independent probability ensembles, let $\mathcal{D} = \langle \mathcal{D}_1, \mathcal{D}_2 \rangle$. There is a probability ensemble $\mathcal{E} : \{0, 1\}^{2nk_n}$, with $\mathcal{E} = \langle \mathcal{E}_1, \mathcal{E}_2 \rangle$, satisfying:*
 - *For every value $E_1 \in \{0, 1\}^{nk_n}$ such that $\Pr_{\mathcal{E}_1}[E_1] > 0$, $\mathbf{H}_{\text{Ren}}(\mathcal{E}_2 | \mathcal{E}_1 = E_1) \geq k_n \mathbf{H}(\mathcal{D}_2 | \mathcal{D}_1) - nk_n^{2/3}$.*
 - $\mathbf{L}_1(\mathcal{E}, \mathcal{D}^{k_n}) \leq 2^{-k_n^{1/3}}$.

Corollary 4.5.3 *Let k_n be an integer-valued \mathbf{P} -time polynomial parameter.*

- *Let $\mathcal{D} : \{0, 1\}^n$ be a probability ensemble, let $m_n = k_n \mathbf{H}(\mathcal{D}) - 2nk_n^{2/3}$, and let $h : \{0, 1\}^{p_n} \times \{0, 1\}^{nk_n} \rightarrow \{0, 1\}^{m_n}$ be a universal hash function. Let $X' \in_{\mathcal{D}^{k_n}} \{0, 1\}^{kn \times n}$ and let $Y \in_{\mathcal{U}} \{0, 1\}^{p_n}$. Then, $\mathbf{L}_1(\langle h_Y(X'), Y \rangle, \mathcal{U}_{m_n+p_n}) \leq 2^{1-k_n^{1/3}}$.*
- *Let $\mathcal{D}_1 : \{0, 1\}^n$ and $\mathcal{D}_2 : \{0, 1\}^n$ be not necessarily independent probability ensembles, and let $\mathcal{D} = \langle \mathcal{D}_1, \mathcal{D}_2 \rangle$. Let $m_n = k_n \mathbf{H}(\mathcal{D}_2 | \mathcal{D}_1) - 2nk_n^{2/3}$. Let $h : \{0, 1\}^{p_n} \times \{0, 1\}^{nk_n} \rightarrow \{0, 1\}^{m_n}$ be a universal hash function. Let $\langle X'_1, X'_2 \rangle \in_{\mathcal{D}^{k_n}} \{0, 1\}^{kn \times 2n}$ and let $Y \in_{\mathcal{U}} \{0, 1\}^{p_n}$. Then, $\mathbf{L}_1(\langle h_Y(X'_2), Y, X'_1 \rangle, \langle \mathcal{U}_{m_n+p_n}, X'_1 \rangle) \leq 2^{1-k_n^{1/3}}$.*

PROOF: Combine Proposition 4.5.2, Lemma 4.5.1, Proposition 2.1.2, and Proposition 2.2.3. ■

4.6 Pseudoentropy generator to a pseudorandom generator

Let $f : \{0, 1\}^n \rightarrow \{0, 1\}^{\ell_n}$ be a pseudoentropy generator with pseudoentropy s_n . In this subsection, we construct a pseudorandom generator based on f . We first start with two preliminary propositions. The following proposition is the computational analog of Proposition 2.1.2.

Proposition 4.6.1 *Let $\mathcal{D} : \{0, 1\}^n$ and $\mathcal{E} : \{0, 1\}^n$ be two probability ensembles and let $f : \{0, 1\}^n \rightarrow \{0, 1\}^{\ell_n}$ be a \mathbf{P} -time function ensemble. Let \mathcal{D} and \mathcal{E} be computationally indistinguishable. Then, $f(\mathcal{D})$ and $f(\mathcal{E})$ are computationally indistinguishable. The reduction is linear-preserving.*

The following proposition first appears in [GM: 84].

Proposition 4.6.2 *Let k_n be an integer-valued \mathbf{P} -time polynomial parameter. Let $\mathcal{D} : \{0, 1\}^{\ell_n}$ and $\mathcal{E} : \{0, 1\}^{\ell_n}$ be \mathbf{P} -samplable probability ensembles. Let \mathcal{D} and \mathcal{E} be computationally indistinguishable. Then, \mathcal{D}^{k_n} and \mathcal{E}^{k_n} are computationally indistinguishable. The reduction is weak-preserving.*

More precisely, there is a probabilistic oracle TM M with the following properties: if A is a \mathbf{R}'_{nk_n} -breaking adversary for distinguishing \mathcal{D}^{k_n} and \mathcal{E}^{k_n} then $M^{(A)}$ is a \mathbf{R}_n -breaking adversary for distinguishing \mathcal{D} and \mathcal{E} , where \mathbf{R}_n is essentially equal to $k_n \mathbf{R}'_{nk_n}$. It is crucial that \mathcal{D} and \mathcal{E} are \mathbf{P} -samplable because the sampling algorithms are used by M . The reduction is only weak-preserving because distinguishing \mathcal{D}^{k_n} and \mathcal{E}^{k_n} with respect to private inputs of length nk_n only translates into distinguishing \mathcal{D} and \mathcal{E} on private inputs of length n .

We now give the construction of a pseudorandom generator g from a pseudoentropy generator f .

Construction 4.6.3 *Let $f : \{0, 1\}^n \rightarrow \{0, 1\}^{m_n}$ be a \mathbf{P} -time function ensemble and let s_n be a \mathbf{P} -time polynomial parameter. Let $k_n = (\lceil (2m_n + 1)/s_n \rceil)^3$ and $j_n = \lfloor k_n(n + s_n) - 2m_n k_n^{2/3} \rfloor$. Let $h : \{0, 1\}^{p_n} \times \{0, 1\}^{k_n m_n} \rightarrow \{0, 1\}^{j_n}$ be a universal hash function. Let $u \in \{0, 1\}^{k_n \times n}$, $y \in \{0, 1\}^{p_n}$, and define \mathbf{P} -time function ensemble $g(u, y) = \langle h_y(f^{k_n}(u)), y \rangle$.*

Theorem 4.6.4 *Let f and g be as described in Construction 4.6.3. Let f be a pseudoentropy generator with pseudoentropy s_n . Then, g is a pseudorandom generator. The reduction is weak-preserving.*

PROOF: Let $f' : \{0, 1\}^{n'_n} \rightarrow \{0, 1\}^{m_n}$ be the \mathbf{P} -time function ensemble that witnesses the pseudoentropy generator of f as guaranteed in Definition 3.4.1 of computational entropy, i.e., $f'(X')$ and $f(X)$ are \mathbf{R} -secure computationally indistinguishable and $\mathbf{H}(f'(X')) \geq n + s_n$, where $X \in_{\mathcal{U}} \{0, 1\}^n$ and $X' \in_{\mathcal{U}} \{0, 1\}^{n'_n}$. Let $U \in_{\mathcal{U}} \{0, 1\}^{k_n \times n}$, $W \in_{\mathcal{U}} \{0, 1\}^{k_n \times n'_n}$, and $Y \in_{\mathcal{U}} \{0, 1\}^{p_n}$. By Proposition 4.6.2, $f^{k_n}(U)$ and $f^{k_n}(W)$ are computationally indistinguishable. From Proposition 4.6.1 it follows that $g(U, Y) = \langle h_Y(f^{k_n}(U)), Y \rangle$ and $g(W, Y) = \langle h_Y(f^{k_n}(W)), Y \rangle$ are computationally indistinguishable. Because

$\mathbf{H}(f'(X')) \geq n + s_n$, by choice of k_n and j_n , using Corollary 4.5.3, it follows that $\mathbf{L}_1(\langle h_Y(f^{k_n}(W)), Y \rangle, \mathcal{U}_{j_n+p_n}) \leq 2^{-k_n^{1/3}}$. Thus, it follows that $g(U, Y)$ and $\mathcal{U}_{j_n+p_n}$ are computationally indistinguishable. Note that by choice of k_n , the output length $j_n + p_n$ of g is longer than its input length $nk_n + p_n$. \blacksquare

4.7 False entropy generator to a pseudoentropy generator

Let $f : \{0, 1\}^n \rightarrow \{0, 1\}^{\ell_n}$ be a false entropy generator with false entropy s_n . In this subsection, we construct a mildly non-uniform pseudoentropy generator based on f . An idea is to extract $\tilde{\mathbf{D}}_f(f(X))$ bits of entropy out of X without compromising the false entropy. (See page 10 for the definition of $\tilde{\mathbf{D}}_f$.) Let $X \in_{\mathcal{U}} \{0, 1\}^n$. The major obstacles are that $\tilde{\mathbf{D}}_f$ is not necessarily a \mathbf{P} -time function ensemble and that f could be a very non-regular function, and thus the variance of $\tilde{\mathbf{D}}_f(f(X))$ could be quite high as a function of X , and we cannot guess its value consistently with accuracy.

Let k_n be an integer-valued \mathbf{P} -time polynomial parameter and let $U \in_{\mathcal{U}} \{0, 1\}^{k_n \times n}$. The intuition behind the following construction is that the false entropy of f^{k_n} is k_n times that of f and that the degeneracy of f^{k_n} is k_n times that of f . Furthermore, if k_n is large enough then, with high probability with respect to U , $\tilde{\mathbf{D}}_{f^{k_n}}(f^{k_n}(U))$ is close to the degeneracy of f^{k_n} . Thus, we use a universal hash function h to extract roughly the degeneracy of $f^{k_n}(U)$ bits of entropy out of U without compromising the false entropy of $f^{k_n}(U)$.

Construction 4.7.1 *Let $f : \{0, 1\}^n \rightarrow \{0, 1\}^{\ell_n}$ be a \mathbf{P} -time function ensemble. Let s_n be a \mathbf{P} -time polynomial parameter and assume for simplicity that $s_n \leq 1$. Let \mathbf{e}_n be an approximation of $\mathbf{H}(f(X))$ to within an additive factor of $s_n/8$, where $X \in_{\mathcal{U}} \{0, 1\}^n$. Fix $k_n = \lceil (4n/s_n)^3 \rceil$ and $j_n = \lceil k_n(n - \mathbf{e}_n) - 2nk_n^{2/3} \rceil$. Let $h : \{0, 1\}^{p_n} \times \{0, 1\}^{nk_n} \rightarrow \{0, 1\}^{j_n}$ be a universal hash function. For $u \in \{0, 1\}^{k_n \times n}$ and $r \in \{0, 1\}^{p_n}$, define \mathbf{P} -time function ensemble*

$$g(\mathbf{e}_n, u, r) = \langle f^{k_n}(u), h_r(u), r \rangle.$$

Lemma 4.7.2 *Let f and g be as described in Construction 4.7.1. Let f be a false entropy generator with false entropy s_n . Then, g is a mildly non-uniform pseudoentropy generator with pseudoentropy 1. The reduction is weak-preserving.*

PROOF: Let $Z \in_{\mathcal{U}} \{0, 1\}^{j_n}$. First, note that $\mathbf{H}(X|f(X)) = n - \mathbf{H}(f(X)) = n - \mathbf{e}_n$. From this and Corollary 4.5.3 (letting $X_1 = f(X)$, $X_2 = X$ in the corollary), it follows that $\mathbf{L}_1(g(\mathbf{e}_n, U, R), \langle f^{k_n}(U), Z, R \rangle) \leq 2^{-k_n^{1/3}}$.

We now prove that $g(\mathbf{e}_n, U, R)$ has computational entropy at least $p_n + nk_n + 1$. Let $\mathcal{D} : \{0, 1\}^{\ell_n}$ be the \mathbf{P} -samplable probability ensemble such that \mathcal{D} and $f(X)$ are computationally indistinguishable and such that

$$\mathbf{H}(\mathcal{D}) \geq \mathbf{H}(f(X)) + s_n.$$

Since \mathcal{D} and $f(X)$ are computationally indistinguishable, $\langle f^{k_n}(U), Z, R \rangle$ and $\langle \mathcal{D}^{k_n}, Z, R \rangle$ are computationally indistinguishable by Proposition 4.6.2, which together with the first claim implies that $g(\mathbf{e}_n, U, R)$ and $\langle \mathcal{D}^{k_n}, Z, R \rangle$ are computationally indistinguishable. Now,

$$\mathbf{H}(\mathcal{D}^{k_n}, Z, R) \geq k_n \cdot (\mathbf{H}(f(X)) + s_n) + j_n + p_n \geq k_n(\mathbf{e}_n + 7s_n/8) + j_n + p_n,$$

and because by choice of k_n and j_n this is at least $p_n + nk_n + 1$. Thus, g has computational entropy at least $p_n + nk_n + 1$, and the lemma follows. \blacksquare

4.8 Mildly non-uniform to a uniform pseudorandom generator

Proposition 4.8.1 *Let \mathbf{a}_n be any value in $\{0, \dots, k_n\}$, where k_n is an integer-valued \mathbf{P} -time polynomial parameter. Let $g : \{0, 1\}^{\lceil \log(k_n) \rceil} \times \{0, 1\}^n \rightarrow \{0, 1\}^{\ell_n}$ be a \mathbf{P} -time function ensemble, where $\ell_n > nk_n$. Let $x' \in \{0, 1\}^{k_n \times n}$ and define \mathbf{P} -time function ensemble $g'(x') = \bigoplus_{i=1}^{k_n} g(i, x'_i)$. Let g be a mildly non-uniform pseudorandom generator when the first input is set to \mathbf{a}_n . Then, g' is a pseudorandom generator. The reduction is weak-preserving.*

PROOF: Let $X \in_{\mathcal{U}} \{0, 1\}^n$, $X' \in_{\mathcal{U}} \{0, 1\}^{k_n \times n}$ and $Z \in_{\mathcal{U}} \{0, 1\}^{\ell_n}$. Suppose there is an adversary A that has distinguishing probability

$$sp_n(A) = |\Pr[A(g'(X')) = 1] - \Pr[A(Z) = 1]|.$$

We describe an oracle TM M such that, for all $i = 1, \dots, k_n$, $M^{(A)}(i)$ has $sp_n(A)$ distinguishing probability for $g(i, X)$ and Z . For all i , the running time for $M^{(A)}(i)$ is the running time for A plus the time to run compute the output of g on $k_n - 1$ inputs. Since this works with respect to all i , in particular it works when $i = \mathbf{a}_n$, from which the result follows.

For each $i = 1, \dots, k_n$, $M^{(A)}(i)$ works as follows. On input u (and i), $M^{(A)}(i)$ randomly generates $x'_1, \dots, x'_{i-1}, x'_{i+1}, \dots, x'_{k_n} \in_{\mathcal{U}} \{0, 1\}^n$ and computes $v = \bigoplus_{j \neq i} g(j, x'_j) \oplus u$. Then $M^{(A)}(i)$ runs A on input v and outputs $A(v)$. By the nature of \bigoplus , if u is chosen randomly according to Z then $M^{(A)}(i) = 1$ with probability $\Pr[A(Z) = 1]$, whereas if u is chosen randomly according to $g(i, X)$ then $M^{(A)}(i) = 1$ with probability $\Pr[A(g'(X')) = 1]$. Thus, for each value of i , $M^{(A)}(i)$ has distinguishing probability $sp_n(A)$ for $g(i, X)$ and Z . \blacksquare

Note that it may be the case that, for most fixed values of $i \in \{1, \dots, k_n\}$, $g(i, X)$ is completely predictable. On the other hand, even if there is a value \mathbf{a}_n for each n such that $g(\mathbf{a}_n, X)$ is pseudorandom, the value of \mathbf{a}_n may not be \mathbf{P} -time computable. This is exactly the case when the lemma is useful, i.e., it is useful to transform the mildly non-uniform pseudorandom generator g into a pseudorandom generator g' .

Note in the given construction, the length of the output of g' on inputs of length nk_n is $\ell_n > nk_n$, and thus g' stretches the input to a string of strictly greater length.

This reduction is only weak-preserving, and the reason is the usual one, i.e., the breaking adversary for $g'(X')$ on inputs of length nk_n is transferred into a breaking adversary for $g(i, X)$ on inputs of length only n .

If g in Proposition 4.8.1 does not satisfy the property that $\ell_n > nk_n$, then for each fixed i we can use Proposition 3.3.4 to stretch the output of $g(i, x)$ (viewed as a function of x) into a string of length longer than nk_n and then exclusive-or together the stretched outputs.

4.9 Summary

Putting together the results in this section, we have:

- A reduction from a one-way permutation to a pseudorandom generator. (From Subsection 4.2.)
- A reduction from a one-to-one one-way function to a pseudorandom generator. (Combining Subsections 4.3 and 4.6.)
- A reduction from a pseudoentropy generator to a pseudorandom generator. (From Subsection 4.6.)
- A reduction from a false entropy generator to a pseudorandom generator. (Combining Subsections 4.7, 4.6, and 4.8.)

5 Extracting entropy from one-way functions

In this section we show how to construct a pseudoentropy generator from any one-way function f with the additional property that the number of inverses of f can be computed in polynomial time, i.e., the function $\tilde{\mathbf{D}}_f$ is a \mathbf{P} -time function ensemble. Combined with the results summarized in Subsection 4.9, this gives a construction of a pseudorandom generator from a one-way function with this property.

One of the reasons for giving the first construction is because it illustrates some of the additional ideas needed for our construction of a false entropy generator from any one-way function. A general one-way function f does not necessarily have the property that $\tilde{\mathbf{D}}_f$ is a \mathbf{P} -time function ensemble, and considerably more effort is needed to construct a pseudorandom generator from it. In the subsequent section, we describe how to construct a false entropy generator from any one-way function. Combined with the results summarized in Subsection 4.9, this gives a construction of a pseudorandom generator from any one-way function.

5.1 One-way function with approximable pre-image sizes to a pseudoentropy generator

To see where we get into trouble with the construction given in Proposition 4.3.1, suppose $f : \{0, 1\}^n \rightarrow \{0, 1\}^{\ell_n}$ is a $2^{n/4}$ -regular one-way function, and let $X, R \in_{\mathcal{U}} \{0, 1\}^n$ and $\beta \in_{\mathcal{U}} \{0, 1\}$. Then, although $\langle f(X), R, X \odot R \rangle$ and $\langle f(X), R, \beta \rangle$ are computationally indistinguishable, $\mathbf{H}(f(X), R, X \odot R)$ is only about $7n/4 + 1$, and thus we have lost about $n/4$ bits of the input entropy through the application of f . Similarly, although $X \odot R$ is hidden given $\langle f(X), R \rangle$, it is also almost completely statistically uncorrelated.

The idea to overcome these problems is to create a new function which is the original one-way function concatenated with the degeneracy of the function number of bits hashed out of its input to regain the lost entropy. Then, Proposition 4.3.1 can be applied to the new function to obtain a pseudoentropy generator. We first show how to construct a pseudoentropy generator in the case when $\tilde{\mathbf{D}}_f$ is a \mathbf{P} -time function ensemble.

Construction 5.1.1 *Let $f : \{0, 1\}^n \rightarrow \{0, 1\}^{\ell_n}$ be a \mathbf{P} -time function ensemble and suppose that $\tilde{\mathbf{D}}_f$ is a \mathbf{P} -time function ensemble. Let $h : \{0, 1\}^{p_n} \times \{0, 1\}^n \rightarrow \{0, 1\}^{n+2}$ be a universal hash function. For $x \in \{0, 1\}^n$ and $y \in \{0, 1\}^{p_n}$, define \mathbf{P} -time function ensemble*

$$f'(x, y) = \langle f(x), h_y(x)_{\{1, \dots, \tilde{\mathbf{D}}_f(f(x))+2\}}, y \rangle.$$

Lemma 5.1.2 *Let f and f' be as described in Construction 5.1.1.*

- (1) *Let f be a one-way function. Then, f' is a one-way function. The reduction is poly-preserving.*
- (2) *Let $X \in_{\mathcal{U}} \{0, 1\}^n$ and $Y \in_{\mathcal{U}} \{0, 1\}^{p_n}$. Then, $\mathbf{H}(f'(X, Y)) \geq n + p_n - 1/2$.*

PROOF of (1) : Suppose adversary A inverts $f'(X, Y)$ with probability δ_n in time T_n . We prove that the following oracle TM M using A on input $z = f(x)$ finds $x' \in \text{pre}_f(z)$ with probability at least $\delta_n^3/128$ when $x \in_{\mathcal{U}} \{0, 1\}^n$.

Description of $M^{(A)}(z)$:

Compute $\tilde{\mathbf{D}}_f(z)$.

Choose $\alpha \in_{\mathcal{U}} \{0, 1\}^{\tilde{\mathbf{D}}_f(z)+2}$.

Choose $y \in_{\mathcal{U}} \{0, 1\}^{p_n}$.

If $A(z, \alpha, y)$ outputs x' with $f(x') = z$ then output x' .

Let $j_n = 2 \lceil \log(2/\delta_n) \rceil$. For all $z \in \text{range}_f$, for all $y \in \{0, 1\}^{p_n}$, define random variable

$$\beta_{z,y} = h_y(W)_{\{1, \dots, \tilde{\mathbf{D}}_f(z) - j_n\}},$$

where $W \in_{\mathcal{U}} \text{pre}_f(z)$. Then, the probability there is a $\gamma \in \{0, 1\}^{2+j_n}$ such that A inverts f on input $\langle f(X), \langle \beta_{f(X), Y}, \gamma \rangle, Y \rangle$ is at least δ_n .

Fix $z \in \text{range}_f$, and let $\beta'_z \in_{\mathcal{U}} \{0, 1\}^{\tilde{\mathbf{D}}_f(z) - j_n}$. By Lemma 4.5.1,

$$\mathbf{L}_1(\langle \beta_{z,Y}, Y \rangle, \langle \beta'_z, Y \rangle) \leq \delta_n/2.$$

Therefore, the probability there is a $\gamma \in \{0, 1\}^{2+j_n}$ such that A inverts f on input $\langle f(X), \langle \beta'_{f(X)}, \gamma \rangle, Y \rangle$ is at least $\delta_n/2$. If we choose $\gamma \in_{\mathcal{U}} \{0, 1\}^{2+j_n}$, we have the probability that A inverts $f(X)$ on input $\langle f(X), \langle \beta'_{f(X)}, \gamma \rangle, Y \rangle$ is at least

$$2^{-(2+j_n)} \cdot \frac{\delta_n}{2} \geq \frac{\delta_n^3}{128}.$$

Note that this is the input distribution in the call to A within $M^{(A)}$. Note also that the run time of $M^{(A)}$ is dominated by the time to run A . Thus, the time-success ratio of $M^{(A)}$ for inverting f' is about $128T_n/\delta_n^3$.

PROOF of (2) : Fix $z \in \text{range}_f$ and let $x, x' \in \text{pre}_f(z)$ such that $x \neq x'$. From the properties of a universal hash function,

$$\Pr[h_Y(x)_{\{1, \dots, \tilde{\mathbf{D}}_f(z)+2\}} = h_Y(x')_{\{1, \dots, \tilde{\mathbf{D}}_f(z)+2\}}] = 2^{-(\tilde{\mathbf{D}}_f(z)+2)} \leq \frac{1}{4 \cdot \#\text{pre}_f(z)}.$$

By calculating the Renyi entropy it follows that

$$\mathbf{H}(f'(X, Y)) \geq -\log\left(\frac{5}{4} \cdot 2^{-n+p_n}\right) = n + p_n + 2 - \log(5).$$

The result follows since $\log(5) \leq 5/2$. ■

Corollary 5.1.3 *Let f , h and f' be as described in Construction 5.1.1. Let $r \in \{0, 1\}^n$ and define \mathbf{P} -time function ensemble $g(x, y, r) = \langle f'(x, y), r, x \odot r \rangle$. Let f be a one-way function. Then, g is a pseudoentropy generator with pseudoentropy $1/2$. The reduction is poly-preserving.*

PROOF: The proof is the same as the proof of Proposition 4.3.1. Let $X, R \in_{\mathcal{U}} \{0, 1\}^n$, $Y \in_{\mathcal{U}} \{0, 1\}^{p_n}$, and $\beta \in_{\mathcal{U}} \{0, 1\}$. From Lemma 5.1.2, part (1) and Proposition 4.1.3 it follows that $g(X, Y, R)$ and $\langle f'(X, Y), R, \beta \rangle$ are computationally indistinguishable, where the reduction is poly-preserving. From Lemma 5.1.2, part (2) it follows that $\mathbf{H}(f'(X, Y), R, \beta) \geq 2n + p_n + 1/2$. On the other hand, the input entropy to $g(X, Y, R)$ is $2n + p_n$, and thus it follows that g has pseudoentropy $1/2$. ■

Theorem 5.1.4 *A pseudorandom generator can be constructed from a one-way function f where $\tilde{\mathbf{D}}_f$ is a \mathbf{P} -time function ensemble. The reduction is weak-preserving.*

PROOF: Combine Construction 5.1.1 with Construction 4.6.3, and use Corollary 5.1.3 and Theorem 4.6.4. ■

The following theorem, an easy corollary of Theorem 5.1.4, was previously obtained by [GKL: 93] using a different construction and proof techniques.

Theorem 5.1.5 *Let $f : \{0, 1\}^n \rightarrow \{0, 1\}^{\ell_n}$ be a σ_n -regular one-way function, where σ_n is a \mathbf{P} -time polynomial parameter. Then, a pseudorandom generator can be constructed from f . The reduction is weak-preserving.*

PROOF: Note that in this case $\tilde{\mathbf{D}}_f(f(x)) = \lceil \log(\sigma_n) \rceil$ for all $x \in \{0, 1\}^n$. Furthermore, $\lceil \log(\sigma_n) \rceil \in \{0, \dots, n\}$. Using this, and combining Construction 5.1.1 with Construction 4.6.3, and using Corollary 5.1.3 and Theorem 4.6.4 yields a mildly non-uniform pseudorandom generator. Then, Theorem 4.8 shows how to construct a pseudorandom generator from this. ■

Based on the ideas presented above, [Luby: 96] (Theorem 10.1 and Theorem 9.3) gives versions of Theorem 5.1.4 and Theorem 5.1.5 where the reduction is poly-preserving when the security parameter is \mathbf{P} -time computable.

6 Any one-way function to a false entropy generator

6.1 Finding determined hidden bits

The final step in the general construction of a pseudorandom generator from a one-way function is to construct a false entropy generator from any one-way function. This is the technically most difficult part of this paper. This construction uses some of the ideas from Construction 5.1.1. Let

$$f : \{0, 1\}^n \rightarrow \{0, 1\}^{\ell_n} \quad (1)$$

be a one-way function and let

$$h : \{0, 1\}^{p_n} \times \{0, 1\}^n \rightarrow \{0, 1\}^{n + \lceil \log(2n) \rceil} \quad (2)$$

be a universal hash function. Similar to Construction 5.1.1, for $x \in \{0, 1\}^n$, $i \in \{0, \dots, n-1\}$ and $r \in \{0, 1\}^{p_n}$, define \mathbf{P} -time function ensemble

$$f'(x, i, r) = \langle f(x), h_r(x)_{\{1, \dots, i + \lceil \log(2n) \rceil\}}, i, r \rangle. \quad (3)$$

Note that from Lemma 5.1.2, the restricted function $f'(x, \tilde{\mathbf{D}}_f(f(x)), r)$ is an almost one-to-one one-way function, except that this is not necessarily a \mathbf{P} -time function ensemble since $\tilde{\mathbf{D}}_f$ may not be a \mathbf{P} -time function ensemble, and this is the main difficulty we must overcome.

Let $X \in_{\mathcal{U}} \{0, 1\}^n$, $R \in_{\mathcal{U}} \{0, 1\}^{p(n)}$, $Y \in_{\mathcal{U}} \{0, 1\}^n$ and $\beta \in_{\mathcal{U}} \{0, 1\}$. From the proof of Lemma 5.1.2 and Corollary 5.1.3, we claim and formalize below that if $i \leq \tilde{\mathbf{D}}_f(f(X))$ then a time limited adversary cannot distinguish $X \odot Y$ from β given Y and $f'(X, i, R)$.

Let

$$\begin{aligned} \mathcal{T} &= \{ \langle x, i \rangle \mid x \in \{0, 1\}^n, i \in \{0, \dots, \tilde{\mathbf{D}}_f(f(x))\} \} \\ \bar{\mathcal{T}} &= \{ \langle x, i \rangle \mid x \in \{0, 1\}^n, i \in \{ \tilde{\mathbf{D}}_f(f(x)) + 1, \dots, n-1 \} \} \end{aligned}$$

Lemma 6.1.1 *Let $W = \langle \tilde{X}, \tilde{I} \rangle \in_{\mathcal{U}} \mathcal{T}$, $R \in_{\mathcal{U}} \{0, 1\}^{p(n)}$, $Y \in_{\mathcal{U}} \{0, 1\}^n$, and $\beta \in_{\mathcal{U}} \{0, 1\}$. Let f be a one-way function. Then,*

$$\langle f'(W, R), \tilde{X} \odot Y, Y \rangle \text{ and } \langle f'(W, R), \beta, Y \rangle$$

are computationally indistinguishable. The reduction is poly-preserving.

PROOF: Let A be a \mathbf{R}' -breaking adversary for distinguishing the two distributions. Then, A is also a \mathbf{R}' -breaking adversary for hidden bit $\tilde{X} \odot Y$ given $\langle f'(W, R), Y \rangle$. Then, from

Proposition 4.1.2, there is an oracle TM M' such that $M'^{(A)}$ is a \mathbf{R}'' -breaking adversary for inverting $f'(W, R)$, where $\mathbf{R}''(n) = n^{\mathcal{O}(1)} \cdot \mathbf{R}'(n)^{\mathcal{O}(1)}$. Finally, we use the same idea as in Lemma 5.1.2, i.e., the success probability of $M'^{(A)}$ on input $\langle f(x), \alpha, i, R \rangle$ for $\alpha \in_{\mathcal{U}} \{0, 1\}^{i + \lceil \log(2n) \rceil}$ is at least inverse polynomial in the success probability of $M'^{(A)}$ on input $f'(x, i, R)$ for each fixed $\langle x, i \rangle \in \mathcal{T}$. Consider the following oracle TM N : N^A on input $f(x)$ chooses $i \in_{\mathcal{U}} \{0, \dots, n-1\}$, $\alpha \in_{\mathcal{U}} \{0, 1\}^{i + \lceil \log(2n) \rceil}$, and $r \in_{\mathcal{U}} \{0, 1\}^{p_n}$ and runs $M'^{(A)}$ on input $\langle f(x), \alpha, i, r \rangle$. Since $\Pr[\langle x, i \rangle \in \mathcal{T}] \geq 1/n$ when $i \in_{\mathcal{U}} \{0, \dots, n-1\}$, it follows that $N^A(f(X))$ produces an inverse with probability at least $1/n$ times the probability $M'^{(A)}(f'(W, R))$ produces an inverse. \blacksquare

If $i \geq \tilde{\mathbf{D}}_f(f(X))$ then X is almost completely determined by $f'(X, i, R)$, and thus $X \odot Y$ is almost completely determined by $f'(X, i, R)$ and Y .

The interesting case is when $i = \tilde{\mathbf{D}}_f(f(X))$, in which case, from Lemma 6.1.1, the adversary is unable to distinguish $X \odot Y$ from β given Y and $f'(X, i, R)$, and yet from part (2) of Lemma 5.1.2, $X \odot Y$ is almost completely determined by $f'(X, i, R)$ and Y . It is from this case that we can extract a little bit of false entropy.

6.2 Construction and Main Theorem

We now describe the construction of a false entropy generator g based on f' . Let

$$k_n \geq 125n^3. \quad (4)$$

Part of the construction is to independently and randomly choose k_n sets of inputs to f' , and concatenate the outputs. In particular, let $X' \in_{\mathcal{U}} \{0, 1\}^{k_n \times n}$, $I' \in_{\mathcal{U}} \{0, 1\}^{k_n \times \lceil \log(n) \rceil}$, $R' \in_{\mathcal{U}} \{0, 1\}^{k_n \times p_n}$. Part of the construction is then $f'^{k_n}(X', I', R')$.

Let $I \in_{\mathcal{U}} \{0, \dots, n-1\}$, let

$$\mathbf{p}_n = \Pr[I \leq \tilde{\mathbf{D}}_f(f(X))], \quad (5)$$

and let

$$m_n = k_n \mathbf{p}_n - 2k_n^{2/3}. \quad (6)$$

We show later that it is sufficient to have an approximation of \mathbf{p}_n to within an additive factor of $1/n$ for the entire construction to work. We need this to be able to claim that g described below is mildly non-uniform. For now we assume we have the exact value of \mathbf{p}_n . Let $Y' \in_{\mathcal{U}} \{0, 1\}^{k_n \times n}$. The value of k_n is chosen to be large enough so that with high probability it is the case that $I'_j \leq \tilde{\mathbf{D}}_f(f(X'_j))$ for at least m_n of the k_n possible values of j , and in this case, from Lemma 6.1.1, $X'_j \odot Y'_j$ looks like a random bit to a time limited adversary given Y'_j and $f'(X'_j, I'_j, R'_j)$.

The problem is that we don't know for which set of m_n values of j the bit $X'_j \odot Y'_j$ looks random to a time limited adversary. Instead, the idea is to hash m_n bits out of all k_n

such bits and release the hashed bits. The intuition is that these m_n hashed bits will look random to a time limited adversary, even though there are really at most $(\mathbf{p}_n - 1/n)k_n$ bits of randomness left in these k_n bits after seeing Y' and $f'^{k_n}(X', I', R')$, and thus there are approximately $m_n - (\mathbf{p}_n - 1/n)k_n \approx n^2$ bits of false entropy. Let

$$h' : \{0, 1\}^{p'_n} \times \{0, 1\}^{k_n} \rightarrow \{0, 1\}^{m_n} \quad (7)$$

be a universal hash function, let $U \in_{\mathcal{U}} \{0, 1\}^{p'_n}$ and define \mathbf{P} -time function ensemble

$$g(\mathbf{p}_n, X', Y', I', R', U) = \langle h'_U(\langle X'_1 \odot Y'_1, \dots, X'_{k_n} \odot Y'_{k_n} \rangle), f'^{k_n}(X', I', R'), U, Y' \rangle. \quad (8)$$

Theorem 6.2.1 *Let f be a one-way function and g be as described above in equations (1) through (8). Then g is a mildly non-uniform false entropy generator with false entropy $10n^2$. The reduction is weak-preserving.*

PROOF: Let $Z \in_{\mathcal{U}} \{0, 1\}^{m_n}$, and let

$$\begin{aligned} \mathcal{D} &= \langle h'_U(\langle X'_1 \odot Y'_1, \dots, X'_{k_n} \odot Y'_{k_n} \rangle), f'^{k_n}(X', I', R'), U, Y' \rangle. \\ \mathcal{E} &= \langle Z, f'^{k_n}(X', I', R'), U, Y' \rangle. \end{aligned}$$

Note that \mathcal{D} is the distribution of the output of g , and \mathcal{E} is the same except that the m_n output bits of h' have been replaced by random bits. Lemma 6.3.1 shows that $\mathbf{H}(\mathcal{E}) \geq \mathbf{H}(\mathcal{D}) + 10n^2$. Corollary 6.3.3 shows that if f is a one-way function then \mathcal{D} and \mathcal{E} are computationally indistinguishable, where the overall reduction is weak-preserving. ■

What remains to prove Theorem 6.2.1 are the proofs of Lemma 6.3.1 and Lemma 6.3.2 of the next subsection. (Corollary 6.3.3 follows immediately from Lemma 6.3.2 and Lemma 6.1.1). Before we turn to this, we state the main result of this paper based on Theorem 6.2.1.

Theorem 6.2.2 *There are one-way functions iff there are pseudorandom generators.*

PROOF: That pseudorandom generators imply one-way functions follows from [Levin : 87]. The converse now follows from Theorem 6.2.1 and the results summarized in Subsection 4.9. ■

6.3 The Main Lemmas

Lemma 6.3.1 $\mathbf{H}(\mathcal{E}) \geq \mathbf{H}(\mathcal{D}) + 10n^2$.

PROOF: The entropy of \mathcal{D} and \mathcal{E} excluding the first m_n bits is exactly the same. The additional entropy in the first m_n bits of \mathcal{E} is equal to m_n . An upper bound on the additional entropy in the first m_n bits of \mathcal{D} is the additional entropy in $\langle X'_1 \odot Y'_1, \dots, X'_{k_n} \odot Y'_{k_n} \rangle$. For each $j \in \{1, \dots, k_n\}$ where $I'_j < \tilde{\mathbf{D}}_f(f(X'_j))$, the amount of entropy added by $X'_j \odot Y'_j$ is at most 1. On the other hand, under the condition that $I'_j \geq \tilde{\mathbf{D}}_f(f(X'_j))$, $X'_j \odot Y'_j$ is determined by $\langle f'(X'_j, I'_j, R'_j), Y'_j \rangle$ with probability at least $1 - 1/2n$, and thus the additional entropy under this condition is at most $1/2n$. Since $I'_j < \tilde{\mathbf{D}}_f(f(X'_j))$ with probability $\mathbf{p}_n - 1/n$, it follows that the additional entropy added by $X'_j \odot Y'_j$ is at most $\mathbf{p}_n - 1/2n$. Therefore, the additional entropy in the first m_n bits of \mathcal{D} is at most $k_n(\mathbf{p}_n - 1/2n) = m_n + 2k_n^{2/3} - k_n/2n < m_n - 10n^2$ by choice of k_n . ■

Lemma 6.3.2 *Let A be an adversary with distinguishing probability*

$$\delta_n = \Pr[A(\mathcal{D}) = 1] - \Pr[A(\mathcal{E}) = 1]$$

for \mathcal{D} and \mathcal{E} . (We assume without loss of generality that $\Pr[A(\mathcal{D}) = 1] > \Pr[A(\mathcal{E}) = 1]$.) Let $W = \langle \tilde{X}, \tilde{I} \rangle \in_{\mathcal{U}} \mathcal{T}$, $R \in_{\mathcal{U}} \{0, 1\}^{p(n)}$, $Y \in_{\mathcal{U}} \{0, 1\}^n$, and $\beta \in_{\mathcal{U}} \{0, 1\}$. There is an oracle TM M such that $M^{(A)}$ distinguishes between

$$\langle f'(W, R), \tilde{X} \odot Y, Y \rangle \text{ and } \langle f'(W, R), \beta, Y \rangle$$

with probability at least $\delta_n/(16k_n)$. The running time of $M^{(A)}$ is polynomial in the running time of A .

The proof of Lemma 6.3.2 is the most technically involved in this paper. Before proving this lemma, we give the main corollary to this lemma, and then give some motivation for the proof of the lemma.

Corollary 6.3.3 *Let f be a one-way function. Then, \mathcal{D} and \mathcal{E} are computationally indistinguishable. The reduction is weak-preserving.*

PROOF: Combine Lemma 6.1.1 with Lemma 6.3.2. ■

We now give some intuition to the proof of Lemma 6.3.2. The oracle TM $M^{(A)}$ will use a non-straightforward hybrid of distributions argument to be able to distinguish the two distributions in the statement of the lemma. To give some intuition about this non-straightforward hybrid, we first describe a related straightforward hybrid argument that we do not know how to implement efficiently.

Consider the following distribution. For $j \in \{1, \dots, k_n\}$, let $C_j = 1$ with probability \mathbf{p}_n and $C_j = 0$ with probability $1 - \mathbf{p}_n$. For all j , if $C_j = 1$ then let $\langle \tilde{X}'_j, \tilde{I}'_j \rangle \in_{\mathcal{U}} \mathcal{T}$, and if

$C_j = 0$ then let $\langle \tilde{X}'_j, \tilde{I}'_j \rangle \in_{\mathcal{U}} \overline{\mathcal{T}}$. Let R', Y' and U be as defined previously. If these random variables are used to define a distribution using the same construction as used to define \mathcal{D} , with $\langle \tilde{X}'_j, \tilde{I}'_j \rangle$ replacing $\langle X'_j, I'_j \rangle$, then the distribution is \mathcal{D} , except that it is described in a slightly different way. Now, suppose this distribution is altered as follows: if $C_j = 1$ then change the j^{th} input bit of h_U from $\tilde{X}'_j \odot Y'_j$ to $B_j \in_{\mathcal{U}} \{0, 1\}$. Call this distribution \mathcal{D}' .

From Lemma 6.1.1 intuitively it should be the case that a time limited adversary should not be able to distinguish \mathcal{D} from \mathcal{D}' . On the other hand, it is not hard to see using Lemma 4.5.1 that the statistical distance between \mathcal{D}' and \mathcal{E} is exponentially small in n . Thus, if adversary A can distinguish between \mathcal{D} and \mathcal{E} , we should be able to use this to distinguish $\langle f'(W, R), \tilde{X} \odot Y, Y \rangle$ and $\langle f'(W, R), \beta, Y \rangle$ as in the statement of Lemma 6.3.2.

The question is whether we can really prove that \mathcal{D}' is computationally indistinguishable from \mathcal{D} . Towards resolving this question, consider the following family of hybrid distributions. For all $j \in \{0, \dots, k_n\}$, let $\mathcal{F}^{(j)}$ be the hybrid distribution between \mathcal{D} and \mathcal{E} which is the same as \mathcal{D}' up to position j and the same as \mathcal{D} thereafter, i.e., it is the same as \mathcal{D} except that for all $i \leq j$, if $C_i = 1$ then change the i^{th} input bit of h_U from $\tilde{X}'_i \odot Y'_i$ to $B_i \in_{\mathcal{U}} \{0, 1\}$. Then, $\mathcal{F}^{(0)} = \mathcal{D}$ and $\mathcal{F}^{(k_n)} \approx \mathcal{E}$. Let $J \in_{\mathcal{U}} \{1, \dots, k_n\}$. Then, $\mathbb{E}_J[A(\mathcal{F}^{(J-1)}) - A(\mathcal{F}^{(J)})] = \delta_n/k_n$.

An inefficient oracle TM could work as follows on input $\langle f'(w, r), b, y \rangle$: The first phase chooses $j \in_{\mathcal{U}} \{1, \dots, k_n\}$, and chooses a sample from $\mathcal{F}^{(j)}$. If $c_j = 0$ then the oracle TM produces a random bit and stops. In the more interesting case, where $c_j = 1$, it replaces the inputs corresponding to the j^{th} position in the sample according to $f'(w, r)$ and y , and the j^{th} input bit of h_u is set to $b \in_{\mathcal{U}} \{0, 1\}$. Then, the second phase runs the adversary A on this input and outputs the bit produced by A . The distinguishing probability for this oracle TM is δ_n/k_n . The problem is that this is not an efficient oracle TM, because it may not be possible to efficiently uniformly sample from \mathcal{T} and $\overline{\mathcal{T}}$ as required. However, it is possible to sample uniformly from $\{0, 1\}^n \times \{0, \dots, n-1\}$, and a \mathbf{p}_n fraction of the samples will be randomly distributed in \mathcal{T} and a $1 - \mathbf{p}_n$ fraction of the samples will be randomly distributed in $\overline{\mathcal{T}}$, and this simple idea is used to construct the efficient adversary described below.

The efficient adversary $M^{(A)}$ described in detail in the proof of Lemma 6.3.2 proceeds in two phases similar to the inefficient oracle TM described above. The first phase of $M^{(A)}$ consists of k_n stages, where stage j produces a coupled pair of distributions, $\mathcal{D}^{(j)}$ and $\mathcal{E}^{(j)}$, both of which are polynomially samplable. Each stage consists of using adversary A and sampling from the distributions produced in the previous stage to produce the pair of output distributions for the current stage. Initially, $\mathcal{D}^{(0)} = \mathcal{D}$ and $\mathcal{E}^{(0)} = \mathcal{E}$, and it will turn out that $\mathcal{D}^{(k_n)} \approx \mathcal{E}^{(k_n)}$.

The first $j-1$ positions in both $\mathcal{D}^{(j)}$ and $\mathcal{E}^{(j)}$ are already fixed in essentially the same

way in $\mathcal{D}^{(j-1)}$ and $\mathcal{E}^{(j-1)}$, and these positions will be fixed the same way in $\mathcal{D}^{(j)}$ and $\mathcal{E}^{(j)}$. To fill in position j in $\mathcal{D}^{(j)}$ and $\mathcal{E}^{(j)}$, many samples of $\langle x_j, i_j \rangle$ are drawn uniformly from $\{0, 1\}^n \times \{0, \dots, n-1\}$, and then with high probability many of them will be in \mathcal{T} and many will be in $\overline{\mathcal{T}}$. We cannot directly tell for each sample whether it is in \mathcal{T} or $\overline{\mathcal{T}}$. Thus, we must use another criteria to decide which of the samples to keep to fill in position j . The criteria used is to use the sample for which the distinguishing probability of A between $\mathcal{D}^{(j)}$ and $\mathcal{E}^{(j)}$ is highest when the j^{th} position is fixed according to the sample.

Let $\delta^{(j)} = \Pr[A(\mathcal{D}^{(j)}) = 1] - \Pr[A(\mathcal{E}^{(j)}) = 1]$. Because $\delta^{(0)} = \delta_n$ and $\delta^{(k_n)} \approx 0$, it follows that

$$\mathbb{E}_{j \in_{\mathcal{U}} \{1, \dots, k_n\}} [\delta^{(j-1)} - \delta^{(j)}] \geq \delta_n / k_n.$$

It is because of this discrepancy between the value of $\delta^{(j)}$ and $\delta^{(j-1)}$ that f' can be inverted in the second phase.

Intuitively, stage j of the first phase works as follows. A bit c_j is chosen randomly to be one with probability \mathbf{p}_n and to be zero with probability $1 - \mathbf{p}_n$. In the distribution $\mathcal{D}^{(j)}$, the j^{th} input $\langle x'_j, i'_j, r'_j \rangle$ to f'^{k_n} is chosen randomly, y'_j is chosen randomly, u is chosen randomly, and then the j^{th} input bit of h'_u is set to a random bit b_j if $c_j = 1$ and to the correct inner product bit if $c_j = 0$. In the distribution $\mathcal{E}^{(j)}$, the j^{th} input of f'^{k_n} is set the same way it is set in $\mathcal{D}^{(j)}$, and thus the two distributions $\mathcal{D}^{(j)}$ and $\mathcal{E}^{(j)}$ are correlated. The choice of the j^{th} inputs is done several times (c_j is chosen only once at the beginning, i.e., it is not rechosen for each of the times) and each time the distinguishing probability of A for $\mathcal{D}^{(j)}$ and the corresponding $\mathcal{E}^{(j)}$ is approximated, and the choice that maximizes the difference between these accepting probabilities determines how $\mathcal{D}^{(j)}$ and $\mathcal{E}^{(j)}$ are finally set.

The second phase of $M^{(A)}$ chooses $j \in_{\mathcal{U}} \{1, \dots, k_n\}$ and then uses the pair of distributions $\mathcal{D}^{(j)}$ and $\mathcal{E}^{(j)}$ produced in the first stage. The idea is to choose a random sample from both $\mathcal{D}^{(j)}$ and $\mathcal{E}^{(j)}$, modify portions of the $\mathcal{D}^{(j)}$ part according to the input to $M^{(A)}$, and run A on both the modified $\mathcal{D}^{(j)}$ sample and the $\mathcal{E}^{(j)}$ sample and based on the outputs produce a one bit output. The intuition is that the distinguishing probability will be $\delta^{(j)}$, which on average over all j is at least δ_n / k_n .

We now turn to the formal proof of Lemma 6.3.2.

PROOF: (Of Lemma 6.3.2)

The oracle TM $M^{(A)}$ works as follows on input $\langle f'(w, r), b, y \rangle$:

Phase 1 : Define $\mathcal{D}^{(0)} = \mathcal{D}$ and $\mathcal{E}^{(0)} = \mathcal{E}$. Let $B \in_{\mathcal{U}} \{0, 1\}^{k_n}$. Let $\rho = \delta_n / (16k_n)$ and

$\tau = 64n^2/\rho$. Stage $j = 1, \dots, k_n$ works as follows: Randomly choose $c_j \in \{0, 1\}$ so that $c_j = 1$ with probability \mathbf{p}_n . Choose $\hat{x}_1, \dots, \hat{x}_\tau \in_{\mathcal{U}} \{0, 1\}^n$ and $\hat{i}_1, \dots, \hat{i}_\tau \in_{\mathcal{U}} \{0, \dots, n-1\}$. For each $m \in \{1, \dots, \tau\}$, define $w_m = \langle \hat{x}_m, \hat{i}_m \rangle$ and let $\mathcal{D}_{c_j}^{(j-1)}(w_m)$ be the same as $\mathcal{D}^{(j-1)}$ except that $\langle X'_j, I'_j \rangle$ is fixed to w_m and the j^{th} input bit of h' is set to

$$\begin{cases} \hat{x}_m \odot Y'_j & \text{if } c_j = 0 \\ B_j & \text{if } c_j = 1. \end{cases}$$

Similarly, define $\mathcal{E}^{(j-1)}(w_m)$ to be the same as $\mathcal{E}^{(j-1)}$ except that $\langle X'_j, I'_j \rangle$ is fixed to w_m . Let

$$\delta_{c_j}^{(j-1)}(w_m) = \Pr[A(\mathcal{D}_{c_j}^{(j-1)}(w_m)) = 1] - \Pr[A(\mathcal{E}^{(j-1)}(w_m)) = 1].$$

Using A and sampling $\mathcal{O}(n/\rho^2)$ times from $\mathcal{D}_{c_j}^{(j-1)}(w_m)$ and $\mathcal{E}^{(j-1)}(w_m)$, produce an estimate $\Delta_{c_j}^{(j-1)}(w_m)$ so that

$$\Pr[|\Delta_{c_j}^{(j-1)}(w_m) - \delta_{c_j}^{(j-1)}(w_m)| > \rho] \leq 2^{-n}.$$

Let $m_0 \in \{1, \dots, \tau\}$ be the index for which $\Delta_{c_j}^{(j-1)}(w_{m_0})$ is maximized. Set $\langle x'_j, i'_j \rangle = w_{m_0}$, $\mathcal{D}^{(j)} = \mathcal{D}_{c_j}^{(j-1)}(w_{m_0})$, $\mathcal{E}^{(j)} = \mathcal{E}^{(j-1)}(w_{m_0})$ and go to the next stage.

Phase 2 : Pick $j \in_{\mathcal{U}} \{0, \dots, k_n - 1\}$. Let $\mathcal{D}^{(j)}(w, r, b, y)$ be the distribution $\mathcal{D}^{(j)}$ except that $f'(X'_{j+1}, I'_{j+1}, R'_{j+1})$ is set to $f'(w, r)$ and the $j+1^{\text{st}}$ input bit of h' is set to b and Y'_{j+1} is set to y . Let $\mathcal{E}^{(j)}(w, r, y)$ be the same as $\mathcal{E}^{(j)}$ except that $f'(X'_{j+1}, I'_{j+1}, R'_{j+1})$ is set to $f'(w, r)$ and Y'_{j+1} is set to y . Let $\beta \in_{\mathcal{U}} \{0, 1\}$, let D be a sample of $\mathcal{D}^{(j)}(w, r, b, y)$ and let E be a sample of $\mathcal{E}^{(j)}(w, r, y)$. If $A(D) = A(E)$ then output β else output $A(D)$.

We now prove that the oracle adversary $M^{(A)}$ as just described distinguishes as claimed in the lemma. Let $w = \langle x, i \rangle$, $d^{(j)}(w, r, b, y) = \mathbb{E}[A(\mathcal{D}^{(j)}(w, r, b, y))]$ and $e^{(j)}(w, r, y) = \mathbb{E}[A(\mathcal{E}^{(j)}(w, r, y))]$. Then,

$$\Pr[M^{(A)}(f'(w, r), b, y) = 1] = 1/2 + (d^{(j)}(w, r, b, y) - e^{(j)}(w, r, y))/2.$$

Also, it follows directly from the definitions that,

$$\mathbb{E}[d^{(j)}(w, R, x \odot Y, Y) - e^{(j)}(w, R, Y)] = \delta_0^{(j)}(w),$$

and

$$\mathbb{E}[d^{(j)}(w, R, \beta, Y) - e^{(j)}(w, R, Y)] = \delta_1^{(j)}(w).$$

Let $\epsilon^{(j)} = \mathbb{E}[\delta_0^{(j)}(W) - \delta_1^{(j)}(W)]$. Thus, the distinguishing probability of $M^{(A)}$ is

$$\mathbb{E}[M^{(A)}(f'(W, R), \tilde{X} \odot Y, Y)] - \mathbb{E}[M^{(A)}(f'(W, R), \beta, Y)] = \mathbb{E}_j[\delta_0^{(j)}(W) - \delta_1^{(j)}(W)]/2 = \mathbb{E}_j[\epsilon^{(j)}]/2,$$

where $j \in_{\mathcal{U}} \{0, \dots, k_n - 1\}$ in the last two expectations. To prove the lemma, it is sufficient to show that $\mathbb{E}_j[\epsilon^{(j)}]/2 \geq \rho$, or equivalently,

$$\mathbb{E}\left[\sum_{j \in \{0, \dots, k_n - 1\}} \epsilon^{(j)}\right] \geq 2\rho k_n = \delta_n/8. \quad (9)$$

The expectation here is over the random choices of $M^{(A)}$ in the first phase. Let $\delta^{(j)} = \Pr[A(\mathcal{D}^{(j)}) = 1] - \Pr[A(\mathcal{E}^{(j)}) = 1]$. We prove (9) by showing below that

- (a) $\mathbb{E}[\delta^{(k_n)}] \leq 2^{-n}$. The expectation is over the random choices of $M^{(A)}$ in the first phase.
- (b) $\mathbb{E}[\delta^{(j)} - \delta^{(j+1)}] \leq \epsilon^{(j)} + 4\rho$. The expectation is over random choices in the $j + 1^{rst}$ stage of phase 1 conditional on any set of choices in the previous stages.

From (a) and (b), and because $\delta^{(0)} = \delta_n$, it follows that

$$\begin{aligned} \delta_n/2 &< \delta_n - \mathbb{E}[\delta^{(k_n)}] \\ &= \sum_{j \in \{0, \dots, k_n - 1\}} \mathbb{E}[\delta^{(j)} - \delta^{(j+1)}] \\ &\leq 4k_n\rho + \mathbb{E}\left[\sum_{j \in \{0, \dots, k_n - 1\}} \epsilon^{(j)}\right] \\ &= \delta_n/4 + \mathbb{E}\left[\sum_{j \in \{0, \dots, k_n - 1\}} \epsilon^{(j)}\right], \end{aligned}$$

and this proves the bound in Equation 9. Thus, it suffices to prove (a) and (b) above.

PROOF of (a) : Since $\Pr[c_j = 1] = \mathbf{p}_n$, applying Chernoff bounds (e.g., see [MR: 95]), we get that, with probability at least $1 - 2^{-n}$,

$$\sum_{j \in \{0, \dots, k_n - 1\}} c_j \geq k_n \mathbf{p}_n - k_n^{2/3} = m_n + k_n^{2/3}.$$

The entropy of the input to h^l conditional on the rest of the bits of $\mathcal{D}^{(k_n)}$ is at least $\sum_{j \in \{0, \dots, k_n - 1\}} c_j$. So, if this sum is at least $m_n + k_n^{2/3}$, applying Lemma 4.5.1, $\mathbf{L}_1(\mathcal{D}^{(k_n)}, \mathcal{E}^{(k_n)}) \leq 2^{-n}$. Thus, $\delta^{(k_n)} = \mathbb{E}[A(\mathcal{D}^{(k_n)})] - \mathbb{E}[A(\mathcal{E}^{(k_n)})] \leq 2^{-n}$.

PROOF of (b) : Let $\bar{W} \in_{\mathcal{U}} \bar{\mathcal{T}}$, and recall that $W \in_{\mathcal{U}} \mathcal{T}$. Then, since the $j+1$ 'st input of h' is always $X'_{j+1} \odot Y'_{j+1}$ in $\mathcal{D}^{(j)}$,

$$\begin{aligned}
\delta^{(j)} &= \mathbf{p}_n \mathbb{E}[\delta_0^{(j)}(W)] + (1 - \mathbf{p}_n) \mathbb{E}[\delta_0^{(j)}(\bar{W})] \\
&= \mathbf{p}_n \mathbb{E}[\delta_1^{(j)}(W)] + \mathbf{p}_n (\mathbb{E}[\delta_0^{(j)}(W)] - \mathbb{E}[\delta_1^{(j)}(W)]) + (1 - \mathbf{p}_n) (\mathbb{E}[\delta_0^{(j)}(\bar{W})]) \\
&= \mathbf{p}_n \mathbb{E}[\delta_1^{(j)}(W)] + \mathbf{p}_n \epsilon^{(j)} + (1 - \mathbf{p}_n) (\mathbb{E}[\delta_0^{(j)}(\bar{W})]) \\
&< \epsilon^{(j)} + \mathbf{p}_n \mathbb{E}[\delta_1^{(j)}(W)] + (1 - \mathbf{p}_n) (\mathbb{E}[\delta_0^{(j)}(\bar{W})])
\end{aligned}$$

We now show that $\mathbb{E}[\delta^{(j+1)}] \geq \mathbf{p}_n \mathbb{E}[\delta_1^{(j)}(W)] + (1 - \mathbf{p}_n) (\mathbb{E}[\delta_0^{(j)}(\bar{W})]) - 4\rho$, and this concludes the proof. Let $c \in \{0, 1\}$ and consider stage j in phase 1. From our choice of τ and the fact that $1/n \leq \mathbf{p}_n \leq 1 - 1/n$, it follows that, with probability at least $1 - 2^{-n}$, at least n/ρ of the w_m 's are in \mathcal{T} , and at least n/ρ of the w_m 's are in $\bar{\mathcal{T}}$. It then follows using Chernoff bounds that

$$\Pr[\max_{1 \leq m \leq \tau} \{\delta_c^{(j)}(w_m)\} \geq \max\{\mathbb{E}[\delta_c^{(j)}(W)], \mathbb{E}[\delta_c^{(j)}(\bar{W})]\} - \rho]$$

is at least $1 - 2^{-n}$. Also, with probability at least $1 - 2^{-n}$, $\Delta_c^{(j)}(w_m)$ is within ρ of the corresponding $\delta_c^{(j)}(w_m)$, and thus (recalling how w_{m_0} is chosen above in stage j)

$$\begin{aligned}
\delta_c^{(j)}(w_{m_0}) &\geq \Delta_c^{(j)}(w_{m_0}) - \rho \\
&= \max_{m \in \{1, \dots, \tau\}} \{\Delta_c^{(j)}(w_m)\} - \rho \\
&\geq \max_{m \in \{1, \dots, \tau\}} \{\delta_c^{(j)}(w_m)\} - 2\rho \\
&\geq \max\{\mathbb{E}[\delta_c^{(j)}(W)], \mathbb{E}[\delta_c^{(j)}(\bar{W})]\} - 3\rho
\end{aligned}$$

with probability at least $1 - 3 \cdot 2^{-n}$. Let $\delta_c^{(j+1)}$ be the value of $\delta^{(j+1)}$ conditional on $c_{j+1} = c$. From this we can conclude that

$$\mathbb{E}[\delta_c^{(j+1)}] \geq \max\{\mathbb{E}[\delta_c^{(j)}(W)], \mathbb{E}[\delta_c^{(j)}(\bar{W})]\} - 4\rho.$$

Since $c_{j+1} = 1$ with probability \mathbf{p}_n ,

$$\begin{aligned}
\mathbb{E}[\delta^{(j+1)}] &= \mathbf{p}_n \mathbb{E}[\delta_1^{(j+1)}] + (1 - \mathbf{p}_n) \mathbb{E}[\delta_0^{(j+1)}] \\
&\geq \mathbf{p}_n \mathbb{E}[\delta_1^{(j)}(W)] + (1 - \mathbf{p}_n) (\mathbb{E}[\delta_0^{(j)}(\bar{W})]) - 4\rho.
\end{aligned}$$

■

Before we continue let us just check that a good approximation of \mathbf{p}_n is sufficient. Suppose that $\mathbf{p}_n \leq \tilde{\mathbf{p}}_n \leq \mathbf{p}_n + \frac{1}{n}$ and we do the entire construction with $\tilde{\mathbf{p}}_n$ replacing \mathbf{p}_n . Enlarge \mathcal{T} to density $\tilde{\mathbf{p}}_n$ by making it contain some elements $\langle x, i \rangle$ with $i = \tilde{\mathbf{D}}_f(f(x)) + 1$. Lemma 6.1.1 is easily seen to remain valid and Lemma 6.3.1 just becomes more true in that the entropy of \mathcal{D} decreases. This implies that it is sufficient to try $\mathcal{O}(n)$ different values of \mathbf{p}_n .

7 A Direct Construction

We have shown how to construct a false entropy generator from an arbitrary one-way function, a pseudoentropy generator from a false entropy generator and finally a pseudorandom generator from pseudoentropy generator. The combinations of these constructions gives a pseudorandom generator from an arbitrary one-way function as stated in Theorem 6.2.2. By literally composing the reductions given in the preceding parts of this paper, we construct a pseudorandom generator with inputs of length n^{34} from a one-way function with inputs of length n . This is obviously not a suitable reduction for practical applications. In this subsection, we use the concepts developed in the rest of this paper, but provide a more direct and efficient construction. However, this construction still produces a pseudorandom generator with inputs of length n^{10} , which is clearly still not suitable for practical applications. (A sharper analysis can reduce this to n^8 , which is the best we could find using the ideas developed in this paper.) The result could only be considered practical if the pseudorandom generator had inputs of length n^2 , or perhaps even close to n . (However, in many special cases of one-way functions, the ideas from this paper are practical, see e.g., [Luby: 96].)

The improvement in the direct construction given here comes from the observation that more than one of the reductions involves a product distribution, whereas only one product distribution is needed for the overall proof.

We start with a one-way function $f : \{0, 1\}^n \rightarrow \{0, 1\}^{\ell_n}$. We construct f' as in equation (3), and let \mathbf{p}_n be the probability that $I \leq \tilde{\mathbf{D}}_f(f(X))$ as in the previous section. Let $\mathcal{X} = \langle X, I, R \rangle$ represent the input distribution to f' , and let c_n be the length of \mathcal{X} and c'_n the length of $f'(\mathcal{X})$. Let $\mathbf{e}_n = \mathbf{H}(f'(\mathcal{X}))$. Let $b(\chi, y) = x \odot y$. Set $k_n = 2000n^6$.

Intuitively, we generate pseudo-random bits as follows: Let $\mathcal{X}' = \mathcal{X}^{k_n}$ and $Y' = Y^{k_n}$. We first compute $f'^{k_n}(\mathcal{X}')$ and $b^{k_n}(\mathcal{X}', Y')$. Intuitively, we are entitled to recapture

$$k_n c_n - \mathbf{H}\langle f'^{k_n}(\mathcal{X}'), b^{k_n}(\mathcal{X}', Y') \rangle$$

bits from \mathcal{X}' , because this is the conditional entropy left after we have computed f'^{k_n} and

b^{k_n} . We are entitled to recapture $k_n \mathbf{p}_n$ bits from the $b^{k_n}(\mathcal{X}', Y')$ (since we get a hidden bit out of each copy whenever $I \leq \tilde{\mathbf{D}}_f(f(X))$). Finally, we should be able to extract $\mathbf{e}_n k_n$ bits from $f^{k_n}(\mathcal{X}')$, since \mathbf{e}_n is the entropy of $f'(\mathcal{X})$. Since $b(n)$ is almost totally predictable for almost all inputs where $I \geq \tilde{\mathbf{D}}_f(f(X))$,

$$\mathbf{H}(f'(\mathcal{X}), b(\mathcal{X}, Y)) \leq \mathbf{e}_n + \mathbf{p}_n - 1/n + 1/(2n).$$

(See the proof of Lemma 6.3.1.) Thus, if we add up all the output bits, we are entitled to $k_n(c_n + 1/(2n))$, or $k_n/(2n)$ more bits than the input to f^{k_n} . However, our methods of extracting entropy are not perfect, so we need to sacrifice some bits at each stage; to use Corollary 4.5.3, we need to sacrifice $2nk_n^{2/3}$ at each stage, so we chose k_n to satisfy $k_n/(2n) > 6nk_n^{2/3}$

Formally, let $m_n = k_n(c_n - \mathbf{e}_n - \mathbf{p}_n + 1/(2n)) - 2nk_n^{2/3}$, $m'_n = k_n \mathbf{p}_n - 2nk_n^{2/3}$, and $m''_n = k_n \mathbf{e}_n - 2nk_n^{2/3}$. Let R_1, R_2 , and R_3 be indices of hash functions so that h_{R_1} maps $k_n c_n$ bits to m_n bits, h_{R_2} maps k_n bits to m'_n bits and h_{R_3} maps $k_n \mathbf{e}'_n$ bits to m''_n bits. Our construction is as follows:

Construction 7.0.4

$$g(\mathcal{X}', Y', R_1, R_2, R_3) = \langle h_{R_1}(\mathcal{X}'), h_{R_2}(b^{k_n}(\mathcal{X}', Y')), h_{R_3}(f^{k_n}(\mathcal{X}')), Y', R_1, R_2, R_3 \rangle.$$

Theorem 7.0.5 *If f is a one-way function and g is as in Construction 7.0.4, then g is a mildly non-uniform pseudorandom generator. The reduction is weak-preserving.*

PROOF: It is easy to check that g outputs more bits than it inputs.

As noted above, the conditional entropy of \mathcal{X} given $f'(\mathcal{X})$ and $b(\mathcal{X}, Y)$ is at least $c_n - \mathbf{e}_n - \mathbf{p}_n + (1/2n)$. Thus, from Corollary 4.5.3, we have that $\langle h_{R_1}(\mathcal{X}'), R_1 \rangle$ is statistically indistinguishable from random bits given $\langle f^{k_n}(\mathcal{X}'), b^{k_n}(\mathcal{X}', Y'), Y' \rangle$. Hence, $g(\mathcal{X}', Y', R_1, R_2, R_3)$ is statistically indistinguishable from $\langle Z_1, h_{R_2}(b^{k_n}(\mathcal{X}', Y')), h_{R_3}(f^{k_n}(\mathcal{X}')), Y', R_1, R_2, R_3 \rangle$, where $Z_1 \in_{\mathcal{U}} \{0, 1\}^{m_n}$. Now, from Lemmas 6.3.2 and 6.1.1, it follows that $h_{R_2}(b^{k_n}(\mathcal{X}', Y'))$ is computationally indistinguishable from random bits given $\langle f^{k_n}(\mathcal{X}'), R_2, Y' \rangle$. Thus, $g(\mathcal{X}', Y', R_1, R_2, R_3)$ is computationally indistinguishable from

$$\langle Z_1, Z_2, h_{R_3}(f^{k_n}(\mathcal{X}')), Y', R_1, R_2, R_3 \rangle,$$

where $Z_2 \in_{\mathcal{U}} \{0, 1\}^{m'_n}$. Finally, from Corollary 4.5.3, $\langle h_{R_3}(f^{k_n}(\mathcal{X}')), R_3 \rangle$ is statistically indistinguishable from $\langle Z_3, R_3 \rangle$, where $Z_3 \in_{\mathcal{U}} \{0, 1\}^{m''_n}$. Thus, the output of g is computationally indistinguishable from a truly random output of the same length. ■

If we use hash functions constructed as Toeplitz matrices then $\mathcal{O}(m)$ bits is sufficient to construct a hash function on m bits and the inputs needed for the hash function is just a constant fraction of all inputs. Then, the input length to g is $\mathcal{O}(nk_n) = \mathcal{O}(n^7)$.

We still need to use Proposition 4.8.1 to get rid of the mild non-uniformity. From the arguments above, it is clear that an approximation of both \mathbf{e}_n and \mathbf{p}_n that is within $1/(8n)$ of their true values is sufficient. Since $0 \leq \mathbf{e}_n \leq n$, and $0 \leq \mathbf{p}_n < 1$, there are at most $\mathcal{O}(n^3)$ cases of pairs to consider. This means that we get a total of $\mathcal{O}(n^3)$ generators, each needing an input of length $\mathcal{O}(n^7)$. Thus the total input size to the pseudorandom generator is $\mathcal{O}(n^{10})$, as claimed.

8 Conclusions

A general problem is to characterize the conditions under which cryptographic applications are possible. By conditions we mean complexity theoretic conditions, e.g., $P \neq NP$, the existence of one-way functions, etc. Examples of cryptographic applications are private key cryptography, identification/authentication, digital signatures, bit commitment, exchanging secrets, coin flipping over the telephone, etc.

For a variety of cryptographic applications it is known that a secure protocol can be constructed from a pseudorandom generator, e.g., the work of [GGM: 86], [LR: 88], [GMR: 89], [Naor: 88], [GMW: 91], show that applications ranging from private key encryption to zero-knowledge proofs can be based on a pseudorandom generator. The results presented in this paper show that these same protocols can be based on any one-way function. The paper [NY: 89] gives a signature scheme that can be based on any one-way permutation, and [Rom: 90], substantially improves this by basing such a scheme on any one-way function.

Using the notion of a false entropy generator, [G: 89] shows that the existence of pseudorandom generators is equivalent to the existence of a pair of \mathbf{P} -samplable distributions which are computationally indistinguishable but statistically very different.

The paper [IL: 89] provides complementary results; a one-way function can be constructed from a secure protocol for any one of a variety of cryptographic applications, including private key encryption, identification/authentication, bit commitment and coin flipping by telephone. The paper [OW: 93] shows that a one-way function can be constructed from any non-trivial zero-knowledge proof protocol. Thus, secure protocols for any of these applications is equivalent to the existence of one-way functions.

The results described in this paper and the previous three paragraphs show that the existence of a one-way function is *central* to modern complexity based cryptography.

Some applications seem unlikely to be shown possible based on any one-way function, e.g., [IR: 89] give strong evidence that exchanging secrets over a public channel is an application of this kind.

A fundamental issue is that of efficiency, both in size and time; the general construction we give for a pseudorandom generator based on any one-way function increases the size of the input by a large polynomial amount and thus is only weak-preserving. This is not good news for practical applications; it would be nice to have a general poly-preserving or a linear-preserving reduction.

9 Acknowledgements

This research evolved over a long period of time and was greatly influenced by many people. We thank Amos Fiat, Moni Naor, Ronitt Rubinfeld, Manuel Blum, Steven Rudich, Noam Nisan, Lance Fortnow, Umesh Vazirani, Charlie Rackoff, Oded Goldreich, Hugo Krawczyk, and Silvio Micali for their insights and contributions to this work. We in particular thank Charlie, Umesh and Manuel for their advice and enthusiasm, and Oded and Hugo for exposing the fourth author to their wealth of insights on this problem. Finally, Oded's insightful comments on every aspect of earlier versions of this paper has improved the presentation in many ways.

References

- [ACGS: 88] Alexi, W., Chor, B., Goldreich, O., Schnorr, C.P., "RSA Rabin Functions: Certain Parts Are As Hard As the Whole", *SIAM J. on Computing*, Vol. 17, 1988, pp. 194–209.
- [BFNW: 96] L. Babai, L. Fortnow, N. Nisan, A. Wigderson, "BPP has Subexponential Time Simulations unless EXPTIME has Publishable Proofs", *Complexity Theory*, Vol 3, 1993, pp. 307–318.
- [BBR: 88] Bennett, C., Brassard, G., Robert, J., "Privacy Amplification by Public Discussion", *Siam J. on Computing*, Vol. 17, No. 2, 1988, pp. 210–229.
- [Blum: 84] Blum, M., "Independent Unbiased Coin Flips From a Correlated Biased Source: A Finite State Markov Chain", *25th IEEE Symposium on Foundations of Computer Science*, 1984, pp. 425–433.
- [BM: 82] Blum, M., and Micali, S., "How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits", *SIAM J. on Computing*, Vol. 13, 1984, pp. 850–864.
- [BH: 89] Boppana, R., Hirschfeld, R., "Pseudo-random generators and complexity classes", S. Micali, ed., *Advances in Computer Research*, vol. 5, pp. 1–26, JAI Press, 1989.

- [Boyar : 89] Boyar, J. “Inferring Sequences Produced by Pseudo-Random Number Generators”, *Jour. of ACM*, Vol. 36, No. 1, 1989, pp.129–141.
- [CW : 79] Carter, L., and M. Wegman, “Universal Classes of Hash Functions”, *JCSS*, 1979, Vol. 18, pp. 143–154.
- [CG : 88] Chor, B., and O. Goldreich, “Unbiased Bits from Sources of Weak Randomness and Probabilistic Communication Complexity”, *SIAM J. on Computing*, Vol. 17, 1988, pp. 230–261.
- [DH : 76] Diffie, D., and Hellman, M., “New directions in cryptography”, *IEEE Trans. Inform. Theory*, Vol. 22, 1976, pp. 644–654.
- [G : 89] Goldreich, O., “A Note on Computational Indistinguishability”, *ICSI Tech Report TR-89-051*, July 1989.
- [GGM : 86] Goldreich, O., S. Goldwasser, and S. Micali, “How to Construct Random Functions”, *J. of ACM*, Vol. 33, No. 4, 1986, pp. 792–807.
- [GKL : 93] Goldreich, O., Krawczyk, H. and Luby, M., “On the Existence of Pseudorandom Generators”, *SIAM J. on Computing*, Vol. 22, No. 6, December, 1993, pp. 1163–1175.
- [GL : 89] Goldreich, O., and L.A. Levin, “A Hard-Core Predicate for any One-way Function”, *21st ACM Symposium on Theory of Computing*, 1989, pp. 25–32.
- [GMW : 91] Goldreich, O., Micali, S., and Wigderson, A., “Proofs that Yield Nothing But their Validity or All Languages in NP have Zero-Knowledge Proofs”, *J. of the ACM*, Vol. 38, No. 3, July 1991, pp. 691–729.
- [GM : 84] Goldwasser, S. and Micali, S., “Probabilistic Encryption,” *JCSS*, Vol. 28, No. 2, April 1984, pp. 270–299.
- [GMR : 89] Goldwasser, S., Micali, S. and Rackoff, C., “The Knowledge Complexity of Interactive Proof Systems,” *SIAM J. on Computing*, Vol. 18, No. 1, 1989, pp. 186–208.
- [GMT : 82] Goldwasser, S., Micali, S. and Tong, P., “Why and how to establish a private code on a public network,” *23^d IEEE Symposium on Foundations of Computer Science*, 1982, pp. 134–144.
- [Hås : 90] Håstad, J. “Pseudo-Random Generators under Uniform Assumptions”, *22nd ACM Symposium on Theory of Computing*, 1990, pp. 395–404.
- [HL : 92] Herzberg, A., Luby, M., “Public Randomness in Cryptography”, proceedings of *CRYPTO 1992*, *ICSI technical report TR-92-068*, October, 1992.

- [IL : 89] Impagliazzo, R. and Luby, M., “One-way functions are essential for information based cryptography,” 30th *IEEE Symposium on Foundations of Computer Science*, 1989, pp. 230–235.
- [ILL : 89] Impagliazzo, R., Levin, L. and Luby, M., “Pseudo-random number generation from one-way functions”, 21^{rst} *ACM Symposium on Theory of Computing*, 1989, pp. 12–24.
- [IN : 96] Impagliazzo, R., Naor, M., “Efficient Cryptographic Schemes Provably as Secure as Subset Sum”, *J. of Cryptology*, Vol. 9, No. 4, pp. 192-216, 1996.
- [IR : 89] Impagliazzo, R. and Rudich, S., “Limits on the Provable Consequences of One-way Functions”, 21^{rst} *ACM Symposium on Theory of Computing*, 1989, pp. 44–56.
- [IZ : 89] Impagliazzo, R., and Zuckerman, D. “How to recycle random bits”, 30th *IEEE Symposium on Foundations of Computer Science*, 1989, pp. 248–253.
- [Knuth : 81] Knuth, D., *Semi-Numerical Algorithm, The Art of Computer Programming*, Addison-Wesley, Second Edition, Vol. 2, Chapter 3, 1981. (Third Edition, expected 1997).
- [Kol : 65] Kolmogorov, A. N., “Three Approaches to the Concept of the Amount of Information”, *Probl. Inf. Transm.*, Vol 1, (1), 1965, pp. 1–7.
- [Kraw : 92] Krawczyk, H., “How to Predict Congruential Generators”, *Journal of Algorithms*, Vol. 13, 1992. pp. 527–545.
- [Levin : 87] Levin, L.A., “One-way Function and Pseudorandom Generators”, *Combinatorica*, Vol. 7, No. 4, 1987, pp. 357–363.
- [Levin : 93] Levin, L.A., “Randomness and Non-determinism”, “One-way Function and Pseudorandom Generators”, *J. of Symb. Logic*, Vol. 58, No. 3, 1993, pp.1102–1103.
- [Luby : 96] Luby, M., **Pseudorandomness and Cryptographic Applications**, Princeton Computer Science Notes, Princeton University Press, 1996.
- [LR : 88] Luby M., and Rackoff, C., “How to Construct Pseudorandom Permutations From Pseudorandom Functions”, *SIAM J. on Computing*, Vol. 17, No. 2, 1988, pp. 373–386.
- [McEl : 78] McEliece, R. J., “A public key cryptosystem based on algebraic coding theory”, *DSN progress report*, Jan.-Feb., 1978, Jet Propulsion Laboratory, California Institute of Technology.

- [McIn: 87] McInnes, J., “Cryptography Using Weak Sources of Randomness,” *Tech. Report 194/87*, U. of Toronto, 1987.
- [MR: 95] Motwani, R. and Raghavan, P., **Randomized Algorithms**, Cambridge University Press, 1995.
- [Naor: 88] Naor, M., “Bit Commitment using Pseudorandom Generators”, *J. of Cryptology*, Vol. 4, pp. 151–158, 1991. Preliminary version appears in *Crypto 89*, pp. 123–132, 1990.
- [NY: 89] Naor, M. and Yung, M., “Universal One-way Hash Functions and Their Applications”, *21st ACM Symposium on Theory of Computing*, 1989, pp. 33–43.
- [OW: 93] Ostrovsky, R and Wigderson, A., “One-way Functions are Essential for Non-Trivial Zero-Knowledge”, *2nd Israel Symposium on the Theory of Computing and Systems*, 1993, pp. 3–17.
- [Renyi: 70] Renyi, A., **Probability Theory**, North-Holland, Amsterdam, 1970.
- [RSA: 78] Rivest, R., Shamir, A., and Adleman, L., “A method for obtaining digital signatures and public-key cryptosystems”, *Comm. of the ACM*, Vol. 21, 1978, pp. 120–126.
- [Rom: 90] Rompel, J., “One-way Functions are Necessary and Sufficient for Secure Signatures”, *22nd ACM Symposium on Theory of Computing*, 1990, pp. 387–394.
- [SV: 86] Santha, M. and Vazirani, U., “Generating Quasi-random Sequences from Slightly-random Sources”, *JCSS*, Vol. 33, No. 1, 1986, pp. 75–87.
- [Shan: 48] Shannon, C., “A Mathematical Theory of Communication”, *Bell Systems Technical Journal*, 27, 1948, pp. 379–423 and pp. 623–656.
- [Sip: 83] Sipser, M., “A Complexity Theoretic Approach to Randomness”, *15th ACM Symposium on Theory of Computing*, 1983, pp. 330–335.
- [Vaz: 87] Vazirani, U., “Towards a Strong Communication Complexity Theory or Generating Quasi-random Sequences from Two Communicating Slightly-random Sources”, *Combinatorica*, Vol. 7, No.4, 1987, pp. 375–392.
- [VV: 85] Vazirani, U. and Vazirani, V., “Random Polynomial Time is Equal to Slightly-random Polynomial Time”, *26th IEEE Symposium on Foundations of Computer Science*, 1985, pp. 417–428.
- [Yao: 82] Yao, A.C., “Theory and Applications of Trapdoor Functions”, *23rd IEEE Symposium on Foundations of Computer Science*, 1982, pp. 80–91.