Construction of a pseudo-random generator from any one-way function

Johan Håstad*
Royal Institute of Technology
Stockholm, Sweden

Russell Impagliazzo†
Department of Computer Science
University of California at San Diego

Leonid A. Levin‡
Computer Science Department
Boston University

Michael Luby§
International Computer Science Institute
Berkeley, California

Abstract

There are many natural examples of conjectured one-way functions, whereas pseudo-random generators have a number of important applications, including the construction of a private key cryptosystem that is provably secure. We show how to construct a pseudo-random generator from any one-way function. Our constructions make extensive use of hashing to extract and smooth entropy from a one-way function.

1 Introduction

One of the basic primitives in cryptography and other areas of computer science is a pseudo-random generator. The usefulness of a pseudo-random generator is demonstrated, e.g., by the facts that it can be used to construct a private key cryptosystem that is secure even against chosen plaintext attack, and can be used to save random bits in Monte Carlo simulation experiments. Intuitively, a *pseudo-random generator* is a polynomial time computable function g that stretches a short random string x into a long string $g(x)$ that "looks" like a random string to *any* feasible algorithm (hereafter called the *adversary*) that is allowed to examine $g(x)$, i.e., the probability the algorithm outputs 1 when the input is a long random string versus the probability it outputs 1 when the input is $g(x)$ is

*Research supported by the Swedish National Board for Technical Development

†Research done while at U.C.Berkeley, research partially supported by NSF grant CCR 88-13632

‡Research supported by NSF grant DCR-8607492

§Research partially done while on leave of absence from the University of Toronto, research supported in part by NSF Grant CCR-9016468, grant No. 89-00312 from the United States-Israel Binational Science Foundation (BSF), Jerusalem, Israel, and by NSERC operating grant A8092

close to zero. Thus, a pseudo-random generator can be used to efficiently convert a small amount of true randomness into a much larger number of seemingly random bits.

The robust notion of a pseudo-random generator, due to [4, Blum, Micali], [41, Yao], should be contrasted with the classical methods of generating random looking bits as described in, e.g., [22, Knuth]. 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, [31, Plumstead] and [23, Krawczyk] 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 distinction in the context of cryptography, where the adversary must be assumed to be as malicious as possible, with the only restriction on tests being computation time. This is a primary motivation for the seminal paper of [4, Blum, Micali], which is the first to define a generator as pseudo-random if it passes all feasible tests, and shows there is a pseudo-random generator based on the presumed difficulty of the discrete logarithm problem.

The fundamental paper [41, Yao] modified this original definition of a pseudo-random generator to the generally used form, shows the two forms are equivalent, and shows how to to construct a pseudo-random generator from any one-way permutation (i.e., f is length preserving and injective). Furthermore, [41, Yao] shows how pseudo-random 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 pseudo-random generator. The results on deterministic simulation were subsequently considerably generalized in [5, Boppana, Hirschfeld].

Since the conditions are rather stringent, it is not easy to come up with a natural candidate for a pseudo-random 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 provably one-way. Some examples of conjectured one-way functions are the discrete logarithm problem modulo a large randomly chosen prime (see, e.g., [8, Diffie, Hellman]), quadratic residuosity (see, e.g., [14, Goldwasser, Micali]), factoring a number that is the product of two large randomly chosen primes (see, e.g., [33, Rivest, Shamir, Adleman]), problems from coding theory (see, e.g., [11, Goldreich, Krawczyk, Luby]), and the subset sum problem for appropriately chosen parameters (see, e.g., [20, Impagliazzo, Naor]).

There are several important works that have contributed to the expansion of the conditions on one-way functions under which a pseudo-random generator can be constructed. [1, Alexi, Chor, Goldreich, Schnorr] show how to construct a pseudo-random generator based on the difficulty of factoring. Up to this point, attention focused only on one-way permutations, in which case the task of inverting $f(x)$ is to find x uniquely. In the case when $f(x)$ is not a permutation, the natural extension of successful inversion to finding any x' such that $f(x') = f(x)$. [24, Levin] introduces one-way functions which remain one-way after several iterations and shows them to be necessary and sufficient for the construction of a pseudo-random generator. [11, Goldreich, Krawczyk, Luby] show how to construct a pseudo-random 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 pseudo-random 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 pseudo-random 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 [19, Impagliazzo, Levin, Luby] and [16, Håstad].

1.1 Concepts

Previous methods, following [4, Blum, Micali], rely on constructing a function that has a "hidden" but "meaningful" bit of the output from one-way functions with special structural properties. An output bit of a function is *hidden* if it is unpredictable by an algorithm that runs in a reasonable amount of time given all the other output bits, and the bit is *meaningful* if it can be predicted from the other output bits by an algorithm without any time restrictions. [12, Goldreich, Levin] 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 pseudo-random 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 pseudo-random 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 hidden but meaningful 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 a single iteration of the one-way function.

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 $\ell(n)$. 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 on page 9).

The notion of a pseudo-random generator is intuitively the following. Following [14, Goldwasser, Micali] and [41, Yao], we say that $g(X)$ and Y are *computationally indistinguishable* if no feasible adversary can distinguish $g(X)$ from Y, in which case we say that g is a *pseudo-random generator*.

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 pseudo-random generator, the computational entropy of $g(X)$ is greater than the Shannon entropy of its input X, and in this sense g amplifies entropy. The notion of computational entropy provides one of the main conceptual tools in our paper.

We introduce the following generalizations of a pseudo-random generator based on computational entropy. We say that $g(X)$ is a *pseudo-entropy 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 pseudo-entropy generator from any false entropy generator and finally a pseudo-random generator from any pseudo-entropy generator. (The presentation of these results in the paper is in reverse order.) We use in a fundamental way the work of [12, Goldreich, Levin].

We use hashing functions and their analysis in a fundamental way in our constructions. This approach has its roots in [11, Goldreich, Krawczyk, Luby]. In [12, Goldreich, Levin], 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 [3, Blum], [36, Santha, Vazirani], [38, Vazirani], [40, Vazirani, Vazirani], [7, Chor, Goldreich] [27, McInnes].) One of our main technical lemmas, (Lemma 1 on page 16), 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.

2 Basic notation

$\mathcal{N}$ is the set of natural numbers. If S is a set then $\sharp S$ 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, $\log(a)$ is the logarithm base two of a and $\mathrm{ilog}(a)$ is the logarithm base two of a rounded up to the nearest integer. If a number is an input to or an output of an algorithm, the assumption is that it is presented in binary notation.

Let x and y be bit strings. We let $\langle x, y \rangle$ denote the sequence of two strings x followed by y, and when appropriate we also view this as the concatenation of x and y. When $\langle x, y \rangle$ is the input to a function f, we write this as $f(x, y)$. We let x_i denote the i^{th} bit of x. Let $x \in \{0,1\}^n$ and let $S \subset \{1, \ldots, n\}$. Then, x_S denotes the subsequence of bits in x indexed by S, e.g., $x_{\{1,\ldots,i\}}$ denotes the first i bits of x, $x_{\{i+1,\ldots,n\}}$ denotes all but the first i bits of x, and thus $x = \langle x_{\{1,\ldots,i\}}, x_{\{i+1,\ldots,n\}} \rangle$. If x and y are bit strings, each of length l, then $x \oplus y$ is the vector sum mod 2 (i.e., bitwise parity) of x and y, i.e., $(x \oplus y)_i = (x_i + y_i) \bmod 2$. The string 1^n denotes the concatenation of n ones, and similarly 0^n denotes the concatenation of n zeroes.

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 $x = \langle x_1, \ldots, x_m \rangle$ of m strings, each of length n, where in this case x_i is the i^{th} row of the matrix, or as a string $x \in \{0,1\}^{mn}$ of length mn, which is the concatenation of the rows of the matrix.

For two equal length strings $x \in \{0,1\}^n$ and $y \in \{0,1\}^n$, define

$$x \cdot y = \sum_{i \in \{1,\ldots,n\}} x_i \cdot y_i \bmod 2.$$

If $x \in \{0,1\}^{l \times m}$ and $y \in \{0,1\}^{m \times n}$ are matrices then $x \cdot y$ indicates matrix multiplication over GF[2], i.e., $x \cdot y$ is the $l \times n$ bit matrix, where the (i,j) entry is $r_i \cdot c_j$, where $r_i \in \{0,1\}^m$ is the i^{th} row of x and $c_j \in \{0,1\}^m$ is the j^{th} column of y.

2.1 Standard functions

Definition 2.1 (standard function) *A function $f(x)$ is a* $(n; \ell(n))$-standard function *if there is a constant $c > 0$ such that for each $n \in \mathcal{N}$, f is a function mapping $\{0,1\}^n$ to $\{0,1\}^{\ell(n)}$ that is computable in time at most n^c. If f is injective for each value of n then f is a* one-to-one standard function. *If f is injective and $\ell(n) = n$ then f is a* standard permutation. *If, for each* $y \in \{0,1\}^{\ell(n)}$ *in the range of* f, $\sharp f^{-1}(y) = \mathrm{reg}(n)$, *then* $f(x)$ *is* $\mathrm{reg}(n)$-regular standard function.

The default input length to f is n when not explicitly stated. These definitions extend in a natural way to functions that have more than one input. In this case, we use commas to separate the lengths of the inputs and a semi-colon to separate the list of lengths of the inputs from the output length. Sometimes in our descriptions we describe standard 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.

2.2 Adversaries and Security

An adversary is, for example, trying to invert a one-way function or trying to distinguish the output of a pseudo-random generator from a truly random string. The security of a primitive is a lower bound on the ability of any adversary to break the primitive. The achievement ratio of an adversary is a measure of its ability to break the primitive.

DEFINITION 2.2 (breaking adversary and security) *An* adversary *A is a probabilistic Turing machine. The* achievement ratio *A for an instance f of a primitive is defined as $T(t)/\delta(t)$, where $T(t)$ is the running time of A and $\delta(t)$ is the success probability of A for breaking f on inputs to f of length t. A is* $\mathbf{R}$-breaking *for f if the achievement ratio $T(t)/\delta(t)$ of A for f satisfies $T(t)/\delta(t) \leq \mathbf{R}(t)$ for infinitely many $t \in \mathcal{N}$. We say f is* $\mathbf{R}$*-secure if there is no* $\mathbf{R}$*-breaking adversary for f.*

The definition of the success probability $\delta(t)$ of A for f depends on the primitive in question, i.e., this probability is defined when the primitive is defined. Intuitively, the smaller the achievement 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 advesary 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 makes it easier to characterize the security of a primitive and to quantify the security preserving properties of reductions for fixed input lengths used in practice.

2.3 Probability ensembles

In general, we use capital and Greek letters to denote random variables and random events. When S is a finite set we use the notation $X \in_{\mathcal{U}} S$ to mean that X is a random variable uniformly distributed in S, and $x \in_{\mathcal{U}} S$ indicates that x is a fixed element of S chosen uniformly. We also view random variable X as defining a probability distribution. If

X is a real-valued random variable, then $\mathrm{Exp}[X]$ denotes the expected value X. If E is a probabilistic event, then $\Pr[E]$ denotes the probability that event E occurs. Unless otherwise stated, all random variables are independent of all other random variables.

The *support* of a probability distribution $\mathcal{D}_n$ is the set of elements that have non-zero probability. We use the notation $X \in_{\mathcal{D}_n} \{0,1\}^n$ to mean that random variable X is randomly distributed in $\{0,1\}^n$ according to distribution $\mathcal{D}_n$, and we use the notation $x \in_{\mathcal{D}_n} \{0,1\}^n$ to mean that string x is randomly chosen in $\{0,1\}^n$ according to $\mathcal{D}_n$.

Definition 2.3 (probability ensemble) *We call $\mathcal{D}_n$ a* probability ensemble *with* support *contained in $\{0,1\}^{\ell(n)}$ if for each $n \in \mathcal{N}$, $\mathcal{D}_n$ is a probability distribution with support contained in $\{0,1\}^{\ell(n)}$.*

Definition 2.4 (polynomial samplable) *Let $\mathcal{D}_n$ be a probability ensemble with support contained in $\{0,1\}^{\ell(n)}$. $\mathcal{D}_n$ is* polynomial samplable *if there is a $(k(n);\ell(n))$-standard function $f(x)$ such that $f(X) \in_{\mathcal{D}_n} \{0,1\}^{\ell(n)}$ when $X \in_{\mathcal{U}} \{0,1\}^{k(n)}$.*

2.4 Miscellaneous notation

Definition 2.5 (at least inverse polynomial) *A real-valued function $s(n)$ is* at least inverse polynomial *if there is a constant $c \geq 0$ such that $s(n) \geq 1/n^c$.*

Definition 2.6 (length parameter) *A positive integer-valued function $\ell(n)$ is a* length parameter *if $\ell(n)$ is polynomial in n and if $\ell(n)$ is computable in time polynomial in n.*

Definition 2.7 (exponentially small in n) *If there is a constant $c > 0$ such that for all sufficiently large $n \in \mathcal{N}$, $a(n) \leq 2^{-n^c}$, then for notational convenience we use* $\mathrm{small}(n)$ *in place of $a(n)$.*

Definition 2.8 (copies of functions and ensembles) *Let $k(n)$ be length parameter and let $\mathcal{D}_n$ be a probability ensemble with support contained in $\{0,1\}^n$. Define ${\mathcal{D}_n}^{k(n)}$ to be the ensemble with support contained in $\{0,1\}^{nk(n)}$, which is obtained by independently sampling $k(n)$ times from $\mathcal{D}_n$ and concatenating the results. Similarly, if $f(x)$ is a $(n;\ell(n))$-standard function then $f^{k(n)}(y) = \langle f(y_1),\ldots,f(y_{k(n)})\rangle$ is a $(nk(n);\ell(n)k(n))$-standard function.*

3 Definitions of the primitives

3.1 One-way function

Definition 3.1 (one-way function) *Let $f(x)$ be a $(n;\ell(n))$-standard function and let $X \in_{\mathcal{U}} \{0,1\}^n$. The* success probability *of adversary A for inverting f is $\delta(n) = \Pr[f(A(f(X))) = f(X)]$. Then, f is a $\mathbf{R}$-secure one-way function if there is no $\mathbf{R}$-breaking adversary for f.*

If there is an $O(n)$-breaking adversary for f then it is not secure at all, whereas an exhaustive adversary that tries all possible inputs to find an inverse is $n^{O(1)} \cdot 2^n$-breaking. Thus, the range of securities that can be hoped for fall between these two extremes. We would like the one-way function to be harder to break than it is to use.

Allowing the security of a primitive to be parameterized is important because different applications require different levels of security. For some applications polynomial security is enough (e.g., $\mathbf{R}(n) = n^{10}$) and for other applications better security is crucial (e.g., $\mathbf{R}(n) = 2^{\log^2(n)}$, or even better $\mathbf{R}(n) = 2^{\sqrt{n}}$).

3.2 Indistinguishability

DEFINITION 3.2 (statistical distance, indistinguishability and quasi-random) *Let X and Y be random variables with support contained in $\{0,1\}^n$. The* statistical distance *between X and Y is*

$$\mathrm{dist}(X,Y) = \sum_{x \in \{0,1\}^n} |\Pr[X = x] - \Pr[Y = x]|/2.$$

X and Y are statistically indistinguishable within $\delta(n)$ *if* $\mathrm{dist}(X,Y) \leq \delta(n)$. *We say X is* quasi-random within $\delta(n)$ *if $Y \in_{\mathcal{U}} \{0,1\}^n$.*

Proposition 1 *For any function f with domain $\{0,1\}^n$,* $\mathrm{dist}(f(X), f(Y)) \leq \mathrm{dist}(X,Y)$.

The following definitions and propositions are the computational analogs of statistical definitions and propositions. For these definitions, let $\mathcal{D}_n$ and $\mathcal{E}_n$ be polynomial samplable ensembles with support contained in $\{0,1\}^n$, let $X \in_{\mathcal{D}_n} \{0,1\}^n$, $Y \in_{\mathcal{E}_n} \{0,1\}^n$, and let $f(x)$ be a $(n;\ell(n))$-standard function.

DEFINITION 3.3 (adversary distinguishing probability for distributions) *If A is an adversary then the* success probability *of A for distinguishing X and Y is*

$$\delta(n) = |\Pr[A(X) = 1] - \Pr[A(Y) = 1]|.$$

Note that X and Y are statistically indistinguishable within $\delta(n)$ if the maximum success probability over *all* adversaries A (with no bound on the computational resources of A) for distinguishing them is at most $\delta(n)$. The notion of computational indistinguishability, originally defined by [14, Goldwasser, Micali] and [41, Yao], restricts the computational resources available to adversaries.

DEFINITION 3.4 (computationally indistinguishable) *X and Y are $\mathbf{R}$-secure* computationally indistinguishable *if there is no $\mathbf{R}$-breaking adversary for distinguishing X and Y.*

3.3 Pseudo-random generator

[4, Blum, Micali], [41, Yao] originally gave the definition of a pseudo-random generator as below, except that we parameterize security more precisely.

DEFINITION 3.5 (pseudo-random generator) *Let $g(x)$ be a $(n; \ell(n))$-standard function where $\ell(n) > n$. The* stretching parameter *of $g(x)$ is $\ell(n) - n \geq 1$. Let $X \in_{\mathcal{U}} \{0,1\}^n$ and $Z \in_{\mathcal{U}} \{0,1\}^{\ell(n)}$. Then, g is a* **R***-secure pseudo-random generator if $g(X)$ and Z are* **R***-secure computationally indistinguishable.*

3.4 Entropy

The following definition of entropy is from [35, Shannon]. Let $\mathcal{D}_n$ be an ensemble with support contained in $\{0,1\}^n$, and let $X \in_{\mathcal{D}_n} \{0,1\}^n$.

DEFINITION 3.6 (information and entropy) *For each $x \in \{0,1\}^n$, define the* information *of x with respect to $\mathcal{D}_n$ to be $\mathrm{Inf}_{\mathcal{D}_n}(x) = -\log(\Pr[X = x])$. The* (Shannon) entropy *of $\mathcal{D}_n$* $\mathrm{Ent}(\mathcal{D}_n) = \mathrm{Exp}[\mathrm{Inf}_{\mathcal{D}_n}(X)]$.

Equivalently, we sometimes refer to the entropy $\mathrm{Ent}(X)$ of random variable X, which is defined to be equal to $\mathrm{Ent}(\mathcal{D}_n)$.

Let $f(x)$ be a $(n; \ell(n))$-function. We sometimes use the following proposition implicitly. This proposition shows that a function f cannot increase entropy in a statistical sense.

Proposition 2 $\mathrm{Ent}(f(X)) \leq \mathrm{Ent}(X)$.

DEFINITION 3.7 (degeneracy of f) *The* degeneracy *of f is $\mathrm{Ent}(X) - \mathrm{Ent}(f(X))$, where $X \in_{\mathcal{U}} \{0,1\}^n$.*

We also use a variant definition of entropy used in [7, Chor, Goldreich].

DEFINITION 3.8 (min-entropy) *The* min-entropy *of $\mathcal{D}_n$ is $\min_{x \in \{0,1\}^n} \mathrm{Inf}_{\mathcal{D}_n}(x)$.*

Intuitively, if a distribution has min-entropy k, it is "at least as random" as the uniform distribution on k bit strings. There are distributions that have arbitrarily large entropy but have only one bit of min-entropy.

The following definition of entropy is due to [32, Renyi].

DEFINITION 3.9 (Renyi entropy) *The* Renyi entropy *of $\mathcal{D}_n$ is $-\log(\Pr[X = Y])$, where $X \in_{\mathcal{D}_n} \{0,1\}^n$ and $Y \in_{\mathcal{D}_n} \{0,1\}^n$ are independent.*

It can be easily verified that the Renyi entropy of a distribution is at most the Shannon entropy and at least the min-entropy but at most twice the min-entropy.

3.5 Pseudo-entropy and false-entropy generators

The difference between a pseudo-random generator and a pseudo-entropy generator is that the output of a pseudo-entropy generator doesn't have to be computationally indistinguishable from the uniform distribution, instead it must be computationally indistinguishable from some distribution $\mathcal{E}_n$ that has more entropy than the input to the generator. Thus, a pseudo-entropy 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 3.10 (computational entropy) *Let $f(x)$ be a $(n; m(n))$-standard function. Then, f has* **R***-secure* computational entropy $s(n)$ *if there is a $(n'(n); \ell(n))$-standard function $f'(x')$ such that $f(X)$ and $f'(X')$ are* **R***-secure computationally indistinguishable and* $\mathrm{Ent}(f'(X') \geq s(n)$*, where $X \in_{\mathcal{U}} \{0,1\}^n$ and $X' \in_{\mathcal{U}} \{0,1\}^{n'(n)}$.*

Definition 3.11 (pseudo-entropy generator) *Let $f(x)$ be a $(n; \ell(n))$-standard function and let $X \in_{\mathcal{U}} \{0,1\}^n$. Then, f is a* **R***-secure* pseudo-entropy generator *(with pseudo-entropy $s(n)$) if $f(X)$ has* **R***-secure computational entropy $n + s(n)$.*

Note that this definition implies that $\ell(n) > n$. Let $Z \in_{\mathcal{U}} \{0,1\}^{\ell(n)}$. If $f(X)$ is a pseudo-random generator then it is easy to see that it is also a pseudo-entropy generator. This is because $f(X)$ and Z are computationally indistinguishable and by definition of a pseudo-random generator, $\ell(n) > n$. Consequently, $\mathrm{Ent}(Z) = \ell(n) \geq n+1$, i.e., f is a pseudo-entropy generator with pseudo-entropy 1.

A false entropy generator is a further generalization of pseudo-entropy 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 3.12 (false entropy generator) *Let $f(x)$ be a $(n; \ell(n))$-standard function and let $X \in_{\mathcal{U}} \{0,1\}^n$. Then, f is a* **R***-secure* false entropy generator *(with false-entropy $s(n)$) if $f(X)$ has* **R***-secure computational entropy* $\mathrm{Ent}(f(X)) + s(n)$.

3.6 Hidden and meaningful bit

In the construction of a pseudo-random 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 (it is meaningful). The idea of a function that hides a meaningful output bit is used implicitly in the original construction of a pseudo-random generator from the discrete logarithm problem [4, Blum, Micali] and has been central to all such constructions since that time.

DEFINITION 3.13 (hidden and meaningful bit) *Let $f(x)$ be a $(n;\ell(n))$-standard function and let $b(x)$ be a $(n;1)$-standard function. Let $\mathcal{D}_n$ be a polynomial samplable probability ensemble with support contained in $\{0,1\}^n$ and let $X \in_{\mathcal{D}_n} \{0,1\}^n$. The* guessing *(or success) probability of adversary A for $b(X)$ given $f(X)$ is*

$$\delta(n) = |\Pr[A(f(X)) = b(X)] - \Pr[A(f(X)) \neq b(X)]|.$$

- *$b(X)$ is $\mathbf{R}$-secure* hidden *given $f(X)$ if there is no $\mathbf{R}$-breaking adversary A for guessing $b(X)$ given $f(X)$.*
- *$b(X)$ is $s(n)$-*meaningful *given $f(X)$ if there is an adversary A (with no resource bounds on A) with guessing probability at least $s(n)$ for $b(x)$ given $f(x)$.*

The following proposition shows a relationship between how meaningful $b(X)$ is given $f(X)$ and the conditional entropy of $b(X)$ given $f(X)$.

Proposition 3 *There is a constant $c > 0$ such that if $b(X)$ is $s(n)$-*meaningful *given $f(X)$ then* $\mathrm{Ent}(b(X), f(X)) \leq \mathrm{Ent}(f(X)) + 1 - c \cdot s(n)^2$.

4 Definitions of reductions

All of the results presented in this paper involve a reduction from one type of primitive to another. Central to the definition of a reduction is the notion of an oracle Turing machine.

DEFINITION 4.1 (oracle Turing machine) *An* oracle Turing machine *is a randomized Turing machine M whose behavior is not fully specified. The behavior is not fully specified in the sense that M, in the course of its execution, interactively makes calls (hereafter described as oracle calls) to and receives corresponding outputs from an algorithm that is not part of the description of M. We let M^A denote the fully specified Turing machine described by M using algorithm A to compute the oracle calls.*

Note that although the running time of M is not defined, the running time of M^A is defined. Also, if A is a Turing machine then so is M^A.

We need 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 taken from [25, Luby], [17, Herzberg, Luby].

DEFINITION 4.2 (comparison measures for achievement ratios) *Let $\mathbf{R}(t)$ and $\mathbf{R}'(t)$ be achievement ratios. We say that:*

- *$\mathbf{R} = \mathcal{O}_{\mathcal{L}}(\mathbf{R}')$ if there is a constant c such that $\mathbf{R}(t) \leq t^c \cdot \mathbf{R}'(t)$.*

- $\mathbf{R} = \mathcal{O}_{\mathcal{P}}(\mathbf{R}')$ *if there are constants* β *and* c *such that* $\mathbf{R}(t) \leq t^c \cdot \mathbf{R}'(t)^\beta$.
- $\mathbf{R} = \mathcal{O}_{\mathcal{S}}(\mathbf{R}')$ *if there are constants* α, β *and* c *such that* $\mathbf{R}(t) \leq t^c \cdot \mathbf{R}'(t^\alpha)^\beta$.

We say $\mathbf{R}' = \Omega_{\mathcal{L}}(\mathbf{R})$, $\mathbf{R}' = \Omega_{\mathcal{P}}(\mathbf{R})$, $\mathbf{R}' = \Omega_{\mathcal{S}}(\mathbf{R})$, *if* $\mathbf{R} = \mathcal{O}_{\mathcal{L}}(\mathbf{R}')$, $\mathbf{R} = \mathcal{O}_{\mathcal{P}}(\mathbf{R}')$, $\mathbf{R} = \mathcal{O}_{\mathcal{S}}(\mathbf{R}')$, *respectively.*

Definition 4.3 (reduction) *There is a* reduction *from primitive 1 to primitive 2 if there is a Turing machine* P *and an oracle Turing machine* M *with the following properties.*

- *Given a description of an instance* f *of primitive 1,* P *produces a description of an instance* g *of primitive 2.*
- *Given any breaking adversary* A *for* g, M^A *is a breaking adversary for* f.

Suppose that M *has the property that if* A *is a* $\mathbf{R}'$*-breaking adversary for* g *then* M^A *is a* $\mathbf{R}$*-breaking adversary for* f. *The reduction is* linear-preserving, polynomial-preserving, slight-preserving *if* $\mathbf{R} = \mathcal{O}_{\mathcal{L}}(\mathbf{R}')$, $\mathbf{R} = \mathcal{O}_{\mathcal{P}}(\mathbf{R}')$, $\mathbf{R} = \mathcal{O}_{\mathcal{S}}(\mathbf{R}')$, *respectively.*

Intuitively, P efficiently converts f into g and M is the guarantee that the security of f is passed on to g.

A linear-preserving reduction is more desirable than a polynomial-preserving reduction which in turn is more desirable than a slight-preserving reduction. Consider a reduction from a one-way function f to a pseudo-random generator g and suppose we want the reduction to guarantee that g is $\mathbf{R}'$-secure. The difference between these types of guarantees isn't dramatic when $\mathbf{R}'$ is not too large. For example, if $\mathbf{R}'(t)$ is polynomial in t, then all three definitions of preserving guarantee that $\mathbf{R}(t)$ is polynomial in t.

However, the difference between these types of guarantees is marked if $\mathbf{R}'(t)$ grows at a faster rate. To see the differences between the strengths of the reductions, suppose we want to conclude that g is $2^{t^{1/2}}$-secure, To be particular, suppose that $\alpha = \beta = 2$ and $c = 0$ are the parameter settings for the three types of reductions. For a linear-preserving reduction, g is $2^{t^{1/2}}$-secure if there is no $2^{t^{1/2}}$-breaking adversary for f. For a polynomial-preserving reduction, g is $2^{t^{1/2}}$-secure if there is no $2^{2t^{1/2}}$-breaking adversary for f. For a slight-preserving reduction, g is $2^{t^{1/2}}$-secure if there is no 2^{2t}-breaking adversary for f. The brute force adversary is a 2^{2t}-breaking adversary for f, and thus the slight-preserving reduction does not guarantee that g is $2^{t^{1/2}}$-secure no matter how secure f is.

It turns out that the primary quantity that determines the strength of the reduction is the ratio t'/t, where t' is the length of the input for g and t is the length of the input for calls to f when computing g. The bigger this ratio the more the loss in security. The reason for this is that typically $\mathbf{R}(t)$ is either linear or polynomial in $\mathbf{R}'(t')$. Thus, if t is much smaller than t' then $\mathbf{R}(t)$ is much larger as a function of t than $\mathbf{R}'(t')$ as a function of t'.

We now give some concrete examples of different strengths of reductions. If $t = t'$ and $\mathbf{R}(t) = \mathbf{R}'(t')$ then the reduction is linear-preserving. Slightly weaker, if $t = \epsilon t'$ for some constant $\epsilon > 0$ and $\mathbf{R}(t) = \mathbf{R}'(t')^{\beta}$ for some constant $\beta > 1$ then the reduction is polynomial-preserving. This can be seen as follows. In the most extreme case, when $\mathbf{R}'(t') = 2^{t'}$, it is easy to verify that $\mathbf{R}'(t') = \mathbf{R}'(t/\epsilon) \leq \mathbf{R}'(t)^{1/\epsilon}$. Thus, $\mathbf{R}(t) \leq \mathbf{R}'(t)^{\beta/\epsilon}$. If t' is substantially larger than t (but still polynomial in t), then the reduction is typically only slight-preserving.

Because of the tremendous superiority of a linear-preserving over a polynomial-preserving over a slight-preserving reduction, it is important to design the strongest reduction possible. Some of the most important work left to be done is to find stronger preserving reductions between primitives than are currently known, e.g., the reductions given in this paper from a one-way function to a pseudo-random generator and from a weak one-way function to a one-way function (in the general case) are only slight-preserving. Note that, although the general construction presented here is only slight-preserving, the techniques described in this paper can yield polynomial-preserving reductions for several special cases of one-way function which cover almost all of the conjectured one-way functions used in practice. (Almost all conjectured one-way functions are either regular or nearly regular.)

5 Constructing a hidden and meaningful bit

How do we go about constructing a function that hides a meaningful bit from a one-way function? The following fundamental proposition of [12, Goldreich, Levin] provides the answer. Let $f(x)$ be a $(n; \ell(n))$-standard function. Let $X \in_{\mathcal{U}} \{0,1\}^n$, $R \in_{\mathcal{U}} \{0,1\}^n$ and $\beta \in_{\mathcal{U}} \{0,1\}^n$.

Proposition 4 *If f is a $\mathbf{R}$-secure one-way function then $X \cdot R$ is $\Omega_{\mathcal{P}}(\mathbf{R})$-secure hidden given $\langle f(X), R \rangle$.*

Proposition 4 presents an elegant, simple and general method of obtaining a hidden bit from a one-way function. We need the following stronger proposition of [12, Goldreich, Levin] in some of our proofs.

Proposition 5 *There is an oracle Turing Machine M with the following properties. Let A be any adversary that accepts as input a string $r \in \{0,1\}^n$ and outputs a single bit, and let $R \in_{\mathcal{U}} \{0,1\}^n$. Then, M^A on input error parameter $\delta(n) > 0$ outputs a list L of n-bit strings with the following property: for any fixed $x \in \{0,1\}^n$, if it is the case that*

$$|\Pr[A(R) = x \cdot R] - \Pr[A(R) \neq x \cdot R]| \geq \delta(n)$$

then, with probability $1 - \mathrm{small}(n)$, it is the case that $x \in L$. (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.

Proposition 6 *If $b(X)$ is $\mathbf{R}$-secure hidden given $f(X)$ then $\langle f(X), b(X)\rangle$ and $\langle f(X), \beta\rangle$ are $\Omega_{\mathcal{L}}(\mathbf{R})$-secure computationally indistinguishable.*

PROOF: Let M be the oracle machine which with respect to adversary A and input $y \in \{0,1\}^n$ works as follows: choose $\alpha \in_{\mathcal{U}} \{0,1\}$; if $A(y,\alpha) = 1$ then output α else output $1-\alpha$. It can be easily verified that if A is an adversary with distinguishing probability

$$\delta(n) = \Pr[A(f(X), b(X)) = 1] - \Pr[A(f(X), \beta) = 1],$$

then $M^A(f(X))$ is an adversary with guessing probability $2\delta(n)$ for $b(X)$ given $f(X)$. ■

Proposition 7 *If f is a $\mathbf{R}$-secure one-way function then $\langle f(X), X \cdot R\rangle$ and $\langle f(X), \beta\rangle$ are $\Omega_{\mathcal{P}}(\mathbf{R})$-secure computationally indistinguishable.*

PROOF: By Proposition 6 there is an oracle machine M such that if A is $\mathbf{R}'$-breaking adversary for $\langle f(X), X \cdot R\rangle$ and $\langle f(X), \beta\rangle$ then M^A is a $\mathbf{R}''$-breaking adversary that distinguishes $X \cdot R$ given $\langle f(X), R\rangle$, where $\mathbf{R}'' = \mathcal{O}_{\mathcal{L}}(\mathbf{R}')$. By Proposition 4, there is an oracle machine N such that N^A is a $\mathbf{R}$-breaking adversary for inverting $f(X)$, where $\mathbf{R} = \mathcal{O}_{\mathcal{P}}(\mathbf{R}'')$. The corollary follows because this implies that $\mathbf{R}' = \Omega_{\mathcal{P}}(\mathbf{R})$. ■

5.1 One-way permutation to a pseudo-random generator

We describe a way to construct a pseudo-random generator from any one-way permutation which is substantially simpler (and has stronger security preserving properties) than the original construction of [41, Yao]. The construction and proof described here is due to [12, Goldreich, Levin].

Proposition 8 *Let $f(x)$ be a $\mathbf{R}$-secure one-way permutation. Let $r \in \{0,1\}^n$ and define standard function $g(x,r) = \langle f(x), r, x \cdot r\rangle$. Then g is $\Omega_{\mathcal{P}}(\mathbf{R})$-secure pseudo-random generator.*

PROOF: Let $X \in_{\mathcal{U}} \{0,1\}^n$, $R \in_{\mathcal{U}} \{0,1\}^n$ 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 7, $g(X,R)$ and $\langle f(X), R, \beta\rangle$ are $\Omega_{\mathcal{P}}(\mathbf{R})$-secure computationally indistinguishable. ■

When f is a permutation then: (1) for any $x \in \{0,1\}^n$, $f(x)$ uniquely determines x; (2) $x \cdot r$ is completely determined by $\langle f(x), r\rangle$. For a general one-way function neither of these two properties necessarily holds. Intuitively, the complications introduced below are used to construct a one-way function with these two properties from a general one-way function.

6 Manipulating entropy via hashing

We provide the main definitions and technical tools for manipulations of entropy using hashing.

6.1 Universal hash functions

The concept of a universal hash function, introduced in [6, Carter, Wegman], has proved to have far reaching and a broad spectrum of applications in the theory of computation.

Definition 6.1 (universal hash functions) *Let $h(x,y)$ be a $(n,\ell(n);m(n))$-standard function. Then, $h(x,y)$ is a (pairwise independent)* universal hash function *if, for all $x \in \{0,1\}^n$, $x' \in \{0,1\}^n \setminus \{x\}$, for all $a,a' \in \{0,1\}^{m(n)}$,*

$$\Pr[(h(x,Y)=a) \wedge (h(x',Y)=a')] = 1/2^{2m(n)},$$

where $Y \in_{\mathcal{U}} \{0,1\}^{\ell(n)}$.

For fixed $y \in \{0,1\}^{\ell(n)}$, we view $h(x,y)$ as a function $h_y(x)$ of x mapping n bits to $m(n)$ bits. Intuitively, a universal hash function $h_y(x)$ has the property that, for a randomly chosen y, every distinct pair x and x' are mapped randomly and independently of each other by h_y.

Definition 6.2 (inner product based hash function) *Define $(n,(n+1)m(n);m(n))$-universal hash function h as follows. Let $x \in \{0,1\}^n$, $y \in \{0,1\}^{(n+1)\times m(n)}$ and then $h_y(x) = \langle x,1\rangle \cdot y$.*

Definition 6.3 (finite field based hash function) *Define $(n,2n;n)$-universal hash function h as follows. There are known efficiently computable bijections between* $\mathrm{GF}[2^n]$ *and $\{0,1\}^n$ and efficient ways to compute field operations given this representation. Define $h_y(x)$ by first computing y_1x+y_2, where x, y_1 and y_2 are considered as elements of* $\mathrm{GF}[2^n]$ *and the expression is computed over* $\mathrm{GF}[2^n]$*, and then letting $h_y(x) \in \{0,1\}^n$ be the bit string that corresponds to the result of the expression. A hash function that maps $\{0,1\}^n$ to $\{0,1\}^{m(n)}$ is the following. Let $\ell(n) = \max\{n,m(n)\}$ and $y \in \{0,1\}^{2\times\ell(n)}$. The hash function is $h_y(x)_{\{1,\ldots,m(n)\}}$, where x is considered as an $\ell(n)$ bit input (padded out with $\ell(n)-n$ zeroes).*

The advantage of the second scheme over the first is that it uses fewer bits. A small disadvantage is that we need a representation of the finite field in question. This is not an essential point, since we can randomly construct such a representation in expected time polynomial in the field size.

Hereafter, whenever we refer to universal hash functions, we mean one of the constructions given above. However, any universal hash function that satisfies the required properties may be used.

6.2 Smoothing distributions with hashing

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

Lemma 1 *Let $\mathcal{D}_n$ be a distribution that has Renyi entropy at least $m(n)$, and let $X \in_{\mathcal{D}_n} \{0,1\}^n$. Let $e(n)$ be a positive integer valued parameter. Let $h_y(x) = h(x,y)$ be a $(n, \ell(n); m(n)-2e(n))$-universal hash function. Let $Y \in_{\mathcal{U}} \{0,1\}^{\ell(n)}$ and let $Z \in_{\mathcal{U}} \{0,1\}^{m(n)-2e(n)}$. Then,*

$$\mathrm{dist}(\langle h_Y(X), Y\rangle, \langle Z, Y\rangle) \leq 2^{-(e(n)+1)}.$$

This lemma is a generalization of a lemma that appears in [37, Sipser]. There, $\mathcal{D}_n$ is the uniform distribution on a set $S \subseteq \{0,1\}^n$ with $\sharp S = 2^{m(n)}$. [27, McInnes] and [2, Bennett, Brassard, Robert] also proved similar lemmas.

The lemma can be interpreted as follows: The Renyi entropy of X is smoothed out to the almost uniform distribution on bit strings of length almost $m(n)$ by using the hash function h_Y as a catalyst. The integer parameter $e(n)$ controls the tradeoff between the uniformity of the output bits of the 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

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

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

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

and from this the proof follows.

For any random variable Z, $\mathrm{Exp}[|Z|^2] \geq \mathrm{Exp}[|Z|]^2$ from Jensen's Inequality. From this it follows that $\mathrm{Exp}[|Z|] \leq \mathrm{Exp}[Z^2]^{1/2}$. Letting $Z = \mathrm{Exp}[\chi(h_Y(X) = a)] - 2^{-s}$, we see that it is sufficient to show for all $a \in \{0,1\}^s$,

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

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

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

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 $x = x'$,

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

Because $\mathrm{Ren\_ent}(\mathcal{D}_n) \geq m$, 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. ■

In this lemma, $\mathcal{D}_n$ is required to have Renyi entropy at least $m(n)$. In many of our applications, the distribution in question has at least min-entropy $m(n)$, and thus the lemma applies because the Renyi entropy of a distribution is at least the min-entropy.

6.3 Converting arbitrary entropy into uniform entropy

Lemma 1 is useful in some parts of our constructions as it is, but in other parts it has a major drawback. In some situations, we want to be able to transform an arbitrary distribution $\mathcal{D}_n$ into a uniform distribution without much loss in Shannon entropy. The drawback is that Lemma 1 says that the Renyi entropy, not the Shannon entropy, of $\mathcal{D}_n$ can be transformed into uniform Shannon entropy. Can we replace the condition that $\mathcal{D}_n$ has a specified amount of Renyi entropy in Lemma 1 by the weaker condition that $\mathcal{D}_n$ has a specified amount of Shannon entropy? Not directly. For example, a distribution can have high Shannon entropy yet still have one element output with probability 1/2. Thus, any function computed based on *one* sample from this distribution generates some output with probability at least 1/2, and therefore is highly non-random. This hints at a solution: take multiple independent samples from the $\mathcal{D}_n$ and apply a randomly chosen hash function to the concatenation of the samples.

Lemma 2 *Let $k(n)$ be a length parameter. For every probability ensemble $\mathcal{D}_n$ with support contained in $\{0,1\}^n$ there is a probability ensemble $\mathcal{E}_n$ with support contained in $\{0,1\}^{nk(n)}$ satisfying:*

- *The min-entropy of $\mathcal{E}_n$ is at least $k(n)\mathrm{Ent}(\mathcal{D}_n) - nk(n)^{2/3}$.*
- *$\mathcal{E}_n$ and ${\mathcal{D}_n}^{k(n)}$ are statistically indistinguishable within* $\mathrm{small}(k(n))$.

PROOF: Let $X \in_{\mathcal{D}_n} \{0,1\}^n$ and let $X' \in_{{\mathcal{D}_n}^{k(n)}} \{0,1\}^{k(n)\times n}$. Then, $\mathrm{Inf}_{{\mathcal{D}_n}^{k(n)}}(X') = \sum_{i\in\{1,\ldots,k(n)\}} \mathrm{Inf}_{\mathcal{D}_n}(X'_i)$ is the sum of $k(n)$ independent non-negative real-valued random variables, each with expected value $\mathrm{Ent}(X) \leq n$. Using Chernoff bounds,

$$\Pr[\mathrm{Inf}_{{\mathcal{D}_n}^{k(n)}}(X') \leq k(n)\mathrm{Ent}(X) - nk(n)^{2/3}/2] \leq \mathrm{small}(k(n)).$$

Let

$$S = \{x' \in \{0,1\}^{k(n)\times n} : \mathrm{Inf}_{{\mathcal{D}_n}^{k(n)}}(x') \leq k(n)\mathrm{Ent}(X) - nk(n)^{2/3}/2\},$$

and let $\bar{S} = \{0,1\}^{k(n)\times n} \setminus S$. Let $\mathcal{E}_n$ be the conditional distribution of ${\mathcal{D}_n}^{k(n)}$ restricted to $\bar{S}$ and let $Z \in_{\mathcal{E}_n} \{0,1\}^{k(n)\times n}$. Since $\Pr[X' \in \bar{S}] \geq 1 - \mathrm{small}(k(n))$, the min-entropy of $\mathcal{E}_n$ is at least $\min_{x' \in \bar{S}} \mathrm{Inf}_{{\mathcal{D}_n}^{k(n)}}(x') + \log(1 - \mathrm{small}(k(n)) \geq k(n)\mathrm{Ent}(\mathcal{D}_n) - nk(n)^{2/3}$. Because $\Pr[X' \in S] \leq \mathrm{small}(k(n))$ it follows that Z and X' are statistically indistinguishable within $\mathrm{small}(k(n))$. ■

Corollary 3 *Let $k(n) \geq n$ be a length parameter. Let $h_y(x) = h(x,y)$ be a $(nk(n), p(n); m(n))$-universal hash function, where $m(n) = k(n)\mathrm{Ent}(\mathcal{D}_n) - 2nk(n)^{2/3}$. Let $X \in_{{\mathcal{D}_n}^{k(n)}} \{0,1\}^{k(n)\times n}$ and let $Y \in_{\mathcal{U}} \{0,1\}^{p(n)}$. The distribution $\langle h_Y(X), Y\rangle$ is quasi-random within* $\mathrm{small}(n)$.

PROOF: Use Lemma 2 (page 17), Lemma 1 (page 16) and Proposition 1 (page 8), noting that the Renyi entropy of a distribution is at least its min-entropy. ■

7 The pseudo-entropy generator link

7.1 Pseudo-entropy generator to a pseudo-random generator

Proposition 10 (page 18), together with Lemma 2 (page 17) via Corollary 3 (page 18), suggests a way of converting a pseudo-entropy generator into a pseudo-random generator; first make several copies of the pseudo-entropy generator so that the Shannon entropy and the min-entropy are approximately equal, then use hashing to convert the min-entropy into uniform entropy. We first start with two preliminary propositions. The following proposition, which is the computational analog of Proposition 1, is used implicitly in several places.

Proposition 9 *Let X and Y be two polynomial samplable ensembles with support in $\{0,1\}^n$ and let f be a standard function. If X and Y are* $\mathbf{R}$*-secure computationally indistinguishable then $f(X)$ and $f(Y)$ are* $\Omega_{\mathcal{L}}(\mathbf{R})$*-secure computationally indistinguishable.*

The following proposition first appears in [14, Goldwasser, Micali].

Proposition 10 *Let $k(n)$ be a length parameter. If $\mathcal{D}_n$ and $\mathcal{E}_n$ are* $\mathbf{R}$*-secure computationally indistinguishable then ${\mathcal{D}_n}^{k(n)}$ and ${\mathcal{E}_n}^{k(n)}$ are* $\Omega_{\mathcal{S}}(\mathbf{R})$*-secure computationally indistinguishable.*

More precisely, there is a probabilistic oracle Turing machine M with the following properties: if A is a $\mathbf{R}'(nk(n))$-breaking adversary for distinguishing ${\mathcal{D}_n}^{k(n)}$ and ${\mathcal{E}_n}^{k(n)}$ then M^A is a $\mathbf{R}(n)$-breaking adversary for distinguishing $\mathcal{D}_n$ and $\mathcal{E}_n$, where $\mathbf{R}(n)$ is essentially equal to $\mathbf{R}'(nk(n))$.

In Proposition 10 it is crucial that both probability ensembles are polynomial samplable because the sampling algorithms are used by M. Because we want to be able to *uniformly* convert a breaking adversary for distinguishing ${\mathcal{D}_n}^{k(n)}$ from ${\mathcal{E}_n}^{k(n)}$ into a breaking adversary for distinguishing $\mathcal{D}_n$ from $\mathcal{E}_n$, we required $\mathcal{E}_n$ to be polynomial samplable in the definition of computational entropy. There is a weaker form of Proposition 10 where neither $\mathcal{D}_n$ nor $\mathcal{E}_n$ are required to be polynomial samplable, but then the much weaker conclusion of the proposition is there is a non-uniform way of converting an adversary for distinguishing ${\mathcal{D}_n}^{k(n)}$ and ${\mathcal{E}_n}^{k(n)}$ into an adversary for distinguishing $\mathcal{D}_n$ and $\mathcal{E}_n$.

Theorem 1 *Let $f(x)$ be a $\mathbf{R}$-secure $(n; m(n))$-pseudo-entropy generator with pseudo-entropy $s(n)$, where $s(n)$ is at least inverse polynomial and computable in time polynomial in n. Let $k(n) = (4n/s(n))^3$ and $j(n) = k(n)(n + s(n)) - 2nk(n)^{2/3}$ be length parameters. Let $h(z, r) = h_r(z)$ be a $(k(n) \cdot m(n), p(n); j(n))$-universal hash function. Let $u \in \{0, 1\}^{k(n)\times n}$ and define standard function $g(u, r) = \langle h_r(f^{k(n)}(u)), r\rangle$. Then, g is a $\Omega_{\mathcal{S}}(\mathbf{R})$-secure pseudo-random generator.*

PROOF: Let $f'(x')$ be the $(n'(n); m(n))$-standard function as described in the definition of computational entropy (page 10), i.e., f' has the following properties. Let $X \in_{\mathcal{U}} \{0, 1\}^n$ and $X' \in_{\mathcal{U}} \{0, 1\}^{n'(n)}$. $f'(X')$ and $f(X)$ are $\mathbf{R}$-secure computationally indistinguishable and $\mathrm{Ent}(f'(X')) \geq n + s(n)$. Let $U \in_{\mathcal{U}} \{0, 1\}^{k(n)\times n}$, $W \in_{\mathcal{U}} \{0, 1\}^{k(n)\times n'(n)}$, and $R \in_{\mathcal{U}} \{0, 1\}^{p(n)}$. Because $\mathrm{Ent}(f'(X')) \geq n + s(n)$, by choice of $k(n)$ and $j(n)$, using Corollary 3 (page 18) with respect to $f'^{k(n)}(W)$, it follows that $\langle h_R(f'^{k(n)}(W)), R\rangle$ is quasi-random within $\mathrm{small}(n)$. By Proposition 10 (page 18), $f^{k(n)}(U)$ and $f'^{k(n)}(W)$ are $\Omega_{\mathcal{S}}(\mathbf{R})$-secure computationally indistinguishable. From Proposition 9 (page 18) it follows that $g(U, R)$ and $\langle h_R(f'^{k(n)}(W)), R\rangle$ are $\Omega_{\mathcal{S}}(\mathbf{R})$-secure computationally indistinguishable. Note that by choice of $k(n)$, the length $j(n)$ of the output of g is strictly longer than the length $nk(n)$ of the input to g. Thus, g is a pseudo-random generator. ■

7.2 One-to-one one-way function to a pseudo-entropy generator

We now describe a construction of a pseudo-entropy generator from any one-to-one one-way function. This construction, together with Theorem 1 (page 19), yields a pseudo-random generator from any one-to-one one-way function. The overall construction is quite different in spirit than the original construction of [11, Goldreich, Krawczyk, Luby]: it illustrates how to construct a pseudo-entropy generator in a particularly simple way using [12, Goldreich, Levin]. The construction is the same as described in Proposition 8 (page 14).

Lemma 4 *Let $f(x)$ be a $\mathbf{R}$-secure one-to-one $(n; \ell(n))$ one-way function. Let $y \in \{0, 1\}^n$ and define standard function $g(x, y) = \langle f(x), x \cdot y, y\rangle$. Then, g is $\Omega_{\mathcal{P}}(\mathbf{R})$-secure pseudo-entropy generator.*

PROOF: Let $X \in_{\mathcal{U}} \{0,1\}^n$, $Y \in_{\mathcal{U}} \{0,1\}^n$ and $\beta \in_{\mathcal{U}} \{0,1\}$. Proposition 7 (page 14 shows that $g(X,Y)$ and $\langle f(X), \beta; Y\rangle$ are $\Omega_{\mathcal{P}}(\mathbf{R})$-secure computationally indistinguishable. Because f is a one-to-one function and β is a random bit, $\mathrm{Ent}(f(X), \beta, Y) = 2n+1$, and thus $g(X,Y)$ has pseudo-entropy 1. ■

Note that even if f is a one-way function it is not possible to argue that g is a pseudo-random 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 $g(X,Y) = \langle f(X), X \cdot Y, Y\rangle$ is not a pseudo-random 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.

7.3 Regular one-way function to a pseudo-entropy generator

The main result of this subsection shows how to construct a pseudo-entropy generator from any one-way function f with the property that the number of inverses of f can be computed in polynomial time. A general one-way function does not necessarily have this property, and considerably more effort is needed to construct a pseudo-entropy generator. The primary reason for giving the construction here is because it illustrates some of the additional ideas needed later for our construction of a false entropy generator from any one-way function.

An easy corollary of the main result is the construction of a pseudo-random generator from a regular one-way function. This corollary was previously obtained by [11, Goldreich, Krawczyk, Luby] using a different construction and proof techniques.

Throughout this subsection, let $f(x)$ be a $(n; \ell(n))$-standard function and let $X \in_{\mathcal{U}} \{0,1\}^n$.

To see where we get into trouble with the construction given in Proposition 8 (page 14), suppose f is a $2^{n/4}$-regular one-way function. Let $Y \in_{\mathcal{U}} \{0,1\}^n$ and $\beta \in_{\mathcal{U}} \{0,1\}$. Then, although $\langle f(X), X \cdot Y, Y\rangle$ and $\langle f(X), \beta, Y\rangle$ are computationally indistinguishable, $\mathrm{Ent}(f(X), X \cdot Y, Y)$ is only about $7n/4 + 1$, and thus we have lost about $n/4$ bits of the input entropy through the application of f. Furthermore, although $X \cdot Y$ is hidden given $\langle f(X), Y\rangle$, it is not a meaningful bit.

The idea to overcome these problems is to create a new function which is the original one-way function $f(x)$ concatenated with bits extracted from x using hashing to regain the lost entropy. Then, Proposition 4 (page 13) can be applied to the new function to obtain a pseudo-entropy generator. This technique of extracting entropy out of the input to f equal to the degeneracy of f using techniques similar to that of Lemma 1 (page 16) has many applications in [18, Impagliazzo, Luby].

For the remainder of this subsection, we make the assumption that the function d defined below is computable in time polynomial in n given $f(x)$. Based on this assumption, we show how to construct a pseudo-random generator.

DEFINITION 7.1 (the function d) *For $x \in \{0,1\}^n$, define $d(f(x)) = \mathrm{ilog}(\sharp f^{-1}(f(x)))$,*

Notice that $d(f(x))$ is an approximation to within an additive factor of 1 of the quantity $n - \mathrm{Inf}_{f(X)}(f(x))$. Furthermore, $\mathrm{Exp}[d(f(X))]$ is within an additive factor of 1 of the degeneracy of f. If f is a $\mathrm{reg}(n)$-regular function then, for each $x \in \{0,1\}^n$, $d(f(x))$ is within an additive factor of 1 of $\log(\mathrm{reg}(n))$, which is the degeneracy of f.

Lemma 5 *Let $h(x,r) = h_r(x)$ be a $(n, p(n); n + \mathrm{ilog}(2n))$-universal hash function. Define $f'(x,r) = \langle f(x), h_r(x)_{\{1,\ldots,d(f(x))+\mathrm{ilog}(2n)\}}, r\rangle$. Let $X \in_{\mathcal{U}} \{0,1\}^n$ and $R \in_{\mathcal{U}} \{0,1\}^{p(n)}$.*

(1) If f is a $\mathbf{R}$-secure one-way function then f' is a $\Omega_{\mathcal{P}}(\mathbf{R})$-secure one-way function.

(2) $\mathrm{Ent}(f'(X,R)) \geq n + p(n) - 1/2$.

PROOF of (1) : Suppose adversary A inverts $f'(X,R)$ with probability $\delta(n)$. We prove that the following oracle Turing machine M using A on input $y = f(x)$ finds $x' \in f^{-1}(y)$ with probability at least $\delta(n)^3/16n$ when $x \in_{\mathcal{U}} \{0,1\}^n$.

Description of $M^A(y)$:

Compute $d(y)$.
Choose $\alpha \in_{\mathcal{U}} \{0,1\}^{d(y)+\mathrm{ilog}(2n)}$.
Choose $r \in_{\mathcal{U}} \{0,1\}^{p(n)}$.
If $A(y,\alpha,r)$ outputs x' with $f(x') = y$ then output x'.

Let $j = j(n) = 2\mathrm{ilog}(2/\delta(n))$. Let β be the random variable $h_R(X)_{\{1,\ldots,d(f(X))-j\}}$. Then, the probability there is a $\gamma \in \{0,1\}^{\mathrm{ilog}(2n)+j}$ such that A inverts f on input $\langle f(X), \langle\beta,\gamma\rangle, R\rangle$ is at least $\delta(n)$.

Let $\beta' \in_{\mathcal{U}} \{0,1\}^{d(f(X))-j}$. By Lemma 1, for any fixed $y = f(x)$, given R, β and β' are statistically indistinguishable within $\delta(n)/2$. Therefore, the probability there is a $\gamma \in \{0,1\}^{\mathrm{ilog}(2n)+j}$ such that A inverts f on input $\langle f(X), \langle\beta',\gamma\rangle, R\rangle$ is at least $\delta(n)/2$. If we choose $\gamma \in_{\mathcal{U}} \{0,1\}^{\mathrm{ilog}(2n)+j}$, we have the probability that A inverts $f(X)$ on input $\langle f(X), \langle\beta',\gamma\rangle, R\rangle$ is at least $\delta(n)^3/16n$. But this is the input distribution in the call to A within M^A.

PROOF of (2) : Fix y in the range of f and let $x \in f^{-1}(y)$. We claim and prove below that $\langle y, h_R(x)_{\{1,\ldots,d(y)+\mathrm{ilog}(2n)\}}, R\rangle$ uniquely determines x with probability at least $1 - 1/2^{\mathrm{ilog}(2n)} \geq 1 - 1/2n$, and from this it follows that

$$\mathrm{Ent}(f'(X,R)) \geq p(n) + n(1 - 1/2n) = n + p(n) - 1/2,$$

There is ambiguity in determining x from the given information if and only if there is some $x' \in f^{-1}(y) \setminus \{x\}$ such that

$$h_R(x')_{\{1,\ldots,d(y)+\mathrm{ilog}(2n)\}} = h_R(x)_{\{1,\ldots,d(y)+\mathrm{ilog}(2n)\}}.$$

By the properties of a universal hash function, for any $x' \in \{0,1\}^n \setminus \{x\}$,

$$\Pr[h_R(x)_{\{1,\ldots,d(y)+\mathrm{ilog}(2n)\}} = h_R(x')_{\{1,\ldots,d(y)+\mathrm{ilog}(2n)\}}] = 2^{-(d(y)+\mathrm{ilog}(2n))}.$$

The claim follows because

$$\Pr[\exists x' \in f^{-1}(y) \setminus \{x\} : h_R(x')_{\{1,\ldots,d(y)+\mathrm{ilog}(2n)\}} = h_R(x)_{\{1,\ldots,d(y)+\mathrm{ilog}(2n)\}}]$$

is upper bounded by
$$\frac{\sharp f^{-1}(y) - 1}{2^{d(y)+\mathrm{ilog}(2n)}} < \frac{1}{2^{\mathrm{ilog}(2n)}}.$$

■

As the following corollary shows, the preceding lemma can be used to construct a pseudo-entropy generator from f'. The outline of the proof is the same as the proof of Lemma 4 (page 19),

Corollary 6 *Let $y \in \{0,1\}^n$ and define standard function $g(x,y,r) = \langle f'(x,r), x \cdot y, y\rangle$. If f is a $\mathbf{R}$-secure one-way function then g is a $\Omega_{\mathcal{P}}(\mathbf{R})$-secure pseudo-entropy generator.*

PROOF: Let $Y \in_{\mathcal{U}} \{0,1\}^n$ and $\beta \in_{\mathcal{U}} \{0,1\}$. From Lemma 5, part (1) and Proposition 7 (page 14) it follows that $g(X,Y,R)$ and $\langle f'(X,R), \beta, Y\rangle$ are $\Omega_{\mathcal{P}}(\mathbf{R})$-secure computationally indistinguishable. From Lemma 5, part (2) it follows that $\mathrm{Ent}(f'(X,R),\beta,Y) \geq 2n + p(n) + 1/2$. On the other hand, the input entropy to $g(X,Y,R)$ is $2n + p(n)$, and thus it follows that g has pseudo-entropy $1/2$. ■

Theorem 2 *If f is a $\mathbf{R}(n)$-secure one-way function and the function d is polynomial time computable, then a $\Omega_{\mathcal{S}}(\mathbf{R})$-secure pseudo-random generator can be constructed from f.*

PROOF: Combine Corollary 6 with Theorem 1 (page 19). ■

The following theorem follows directly from Theorem 2.

Theorem 3 *If f is a $\mathbf{R}$-secure* $\mathrm{reg}(n)$*-regular one-way function and* $\mathrm{ilog}(\mathrm{reg}(n))$ *can be computed in polynomial time then a $\Omega_{\mathcal{S}}(\mathbf{R})$-secure pseudo-random generator can be constructed from f.*

Based on the ideas presented above, [25, Luby] gives versions of both Theorem 2 and Theorem 3 where the reduction is polynomial-preserving.

Theorem 4 below is a stronger version of Theorem 3: it does not assume that $\mathrm{ilog}(\mathrm{reg}(n))$ can be computed in polynomial time. The basic idea is to define a polynomial number of generators, one for each possible value of $\mathrm{ilog}(\mathrm{reg}(n))$, and then the pseudo-random generator consists of the "exclusive-or" of the polynomial number of outputs of the generators. The following "exclusive-or lemma" provides this key technical component in the proof of the theorem.

Lemma 7 *Let $g(i,x)$ be a $(\mathrm{ilog}(k(n)), n; \ell(n))$-standard function, where $k(n)$ is polynomial in n and where $\ell(n) > nk(n)$. Let $x' \in \{0,1\}^{k(n)\times n}$ and define standard function $g'(x') = \oplus_{i=1}^{k(n)} g(i, x'_i)$. There is an oracle machine M such that if A is an $\mathbf{R}'(nk(n))$-breaking adversary for g' as a pseudo-random generator then, for all $i = 1, \ldots, k(n)$, $M^A(i)$ is an $\mathbf{R}(n)$-breaking adversary for $g(i,\cdot)$ as a pseudo-random generator, where $\mathbf{R}(n)$ is linearly related to $\mathbf{R}'(nk(n))$.*

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

$$\delta(n) = \left|\Pr[A(g'(X')) = 1] - \Pr[A(Z) = 1]\right|.$$

We describe an oracle machine M such that, for all $i = 1, \ldots, k(n)$, $M^A(i)$ has $\delta(n)$ 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, from which the result follows.

For each $i = 1, \ldots, k(n)$, $M^A(i)$ works as follows. On input u (and i), $M^A(i)$ randomly generates $x'_1, \ldots, x'_{i-1}, x'_{i+1}, \ldots, x'_{k(n)} \in_{\mathcal{U}} \{0,1\}^n$ and computes $v = \oplus_{j\neq i} g(j, x'_j) \oplus u$. Then $M^A(i)$ runs A on input v and outputs $A(v)$. By the nature of $\oplus$, 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 $\delta(n)$ for $g(i,X)$ and Z. ■

Note that it may be the case that, for most fixed values of $i \in \{1, \ldots, k(n)\}$, $g(i,X)$ is completely predictable. On the other hand, even if there is a value $s(n)$ for each n such that $g(s(n), X)$ is pseudo-random, the value of $s(n)$ may not be computable in time polynomial in n. This is exactly the case when the lemma is useful, i.e., it is useful to transform the somewhat non-uniform pseudo-randomness of g into a pseudo-random generator g'. By 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 slight-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 Lemma 7 does not satisfy the property that $\ell(n) > nk(n)$, then for each fixed i we can use the following proposition 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. This proposition appears in [5, Boppana, Hirschfeld] and is due to O. Goldreich and S. Micali.

Proposition 11 *There is a linear-preserving reduction that takes a pseudo-random generator g with stretching parameter one and produces a pseudo-random generator g' with stretching parameter polynomial in the input length.*

Theorem 4 *Let f be a* $\mathbf{R}$*-secure* $\mathrm{reg}(n)$*-regular one-way function (where* $\mathrm{reg}(n)$ *is not necessarily computable in polynomial time). Let $m(n) = n(n+1)+1$. Define* $(\mathrm{ilog}(n), n, p'(n); m(n))$*-standard function $g'(i,x,r')$, where $g'(i,x,r')$ is the generator obtained by assuming that* $\mathrm{ilog}(\mathrm{reg}(n))$ *is equal to i, combining the constructions in Theorem 3 with that in Proposition 11 so that the output is of length $m(n)$. Let $y \in \{0,1\}^{(n+1)\times n}$ and define standard function $g(y,r') = \oplus_{i=0}^{n} g'(i, y_{i+1}, r')$. Then, g is a $\Omega_{\mathcal{S}}(\mathbf{R})$-secure pseudo-random generator.*

PROOF: Define function $g''(x,r') = g'(\mathrm{ilog}(\mathrm{reg}(n)), x, r')$. Let $R' \in_{\mathcal{U}} \{0,1\}^{p'(n)}$. Although g'' may not be computable in polynomial time, it can nevertheless be shown, by combining Theorem 3 and Proposition 11, that $g''(X,R')$ is pseudo-random. It follows from Lemma 7 (page 23) that g is a pseudo-random generator. ■

8 The false-entropy generator link

8.1 False-entropy generator to a pseudo-random generator

Let $f(x)$ be a $\mathbf{R}$-secure $(n;\ell(n))$ false entropy generator with false entropy $s(n)$, where $s(n)$ is at least inverse polynomial and computable in time polynomial in n. Our goal is to construct a pseudo-random generator based on f when the function d is not necessarily computable in polynomial time (see page 21 for the definition of d). Let $X \in_{\mathcal{U}} \{0,1\}^n$. The analogous idea to that used in Subsection 7.3 would be to extract $d(f(X))$ bits of entropy out of X without compromising the false entropy. The major obstacle is that f could be a many-to-one non-regular function, and thus the variance of $d(f(X))$ could be quite high.

Let $k(n)$ and $j(n)$ be length functions. Let $h_r(u) = h(u,r)$ be a $(nk(n), p(n); j(n))$-universal hash function. Define standard function

$$g(u,r) = \langle f^{k(n)}(u), h_r(u), r\rangle.$$

Let $U \in_{\mathcal{U}} \{0,1\}^{k(n)\times n}$ and $R \in_{\mathcal{U}} \{0,1\}^{p(n)}$. It turns out that g is a pseudo-entropy generator for appropriate choices of $k(n)$ and $j(n)$. The intuition 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 $d(f^{k(n)}(U))$ is close to the degeneracy of $f^{k(n)}$. We set $j(n)$ roughly equal to the degeneracy of $f^{k(n)}$, and the intuition is that h_R extracts entropy $j(n)$ from U without compromising the false entropy of $f^{k(n)}$.

Lemma 8 *Let $a(n)$ be an approximation of* $\mathrm{Ent}(f(X))$ *to within an additive factor of $s(n)/8$. Fix $k(n) = (4n/s(n))^3$ and $j(n) = k(n)(n-a(n)) - 2nk(n)^{2/3}$. Let $Z \in_{\mathcal{U}} \{0,1\}^{j(n)}$. Then, $\langle f^{k(n)}(U), h_R(U), R\rangle$ and $\langle f^{k(n)}(U), Z, R\rangle$ are statistically indistinguishable within* $\mathrm{small}(n)$.

PROOF: For all $i = 1, \ldots, k(n)$ define random variable $Y(u_i) \in_{\mathcal{U}} f^{-1}(f(u_i))$, and thus the min-entropy of $Y(u_i)$ is $MY(u_i) = \log(\sharp f^{-1}(f(u_i)))$. Let $MY(u) = \sum_{i\in\{1,\ldots,k(n)\}} MY(u_i)$ be the min-entropy of $Y^{k(n)}(u)$. For all $u_i \in \{0,1\}^n$, $MY(u_i) \in [0,n]$, and $\mathrm{Exp}[MY(U_i)] = n - \mathrm{Ent}(f(X))$. Thus,

$$\mathrm{Exp}[MY(U)] = k(n)(n - \mathrm{Ent}(f(X))),$$

and using Chernoff bounds,

$$\Pr\left[MY(U) \leq k(n)(n - \mathrm{Ent}(f(X))) - nk(n)^{2/3}\right] \leq \mathrm{small}(n).$$

By the definitions of $j(n)$, $a(n)$ and $k(n)$,

$$k(n)(n - \mathrm{Ent}(f(X))) - nk(n)^{2/3} \geq j(n) - k(n)s(n)/8 + nk(n)^{2/3} \geq j(n) + 8n^3.$$

For a fixed setting u, if $MY(u) \geq j(n) + 8n^3$, then Lemma 1 (page 16) shows that $\langle h_R(Y(u)), R\rangle$ is quasi-random within $\mathrm{small}(n)$. From these two facts, it is not hard to see that $\langle f^{k(n)}(U), h_R(U), R\rangle$ and $\langle f^{k(n)}(U), Z, R\rangle$ are statistically indistinguishable within $\mathrm{small}(n)$. ■

Lemma 9 *If f is a* $\mathbf{R}$*-secure false entropy generator with false-entropy $s(n)$ and $a(n)$ is computable in time polynomial in n then g is a $\Omega_{\mathcal{S}}(\mathbf{R})$-secure pseudo-entropy generator.*

PROOF: We claim that $g(U,R)$ has computational entropy $nk(n)+1$. From Lemma 8, $g(U,R)$ and $\langle f^{k(n)}(U), Z, R\rangle$ are statistically indistinguishable within $\mathrm{small}(n)$. Let $\mathcal{D}_n$ be the ensemble such that $\mathcal{D}_n$ and $f(X)$ are computationally indistinguishable and such that

$$\mathrm{Ent}(\mathcal{D}_n) \geq \mathrm{Ent}(f(X)) + s(n) \geq a(n) + 7s(n)/8.$$

Since $\mathcal{D}_n$ and $f(X)$ are computationally indistinguishable, $\langle f^{k(n)}(U), Z, R\rangle$ and $\langle {\mathcal{D}_n}^{k(n)}, Z, R\rangle$ are computationally indistinguishable by Proposition 10 (page 18), which implies that $g(U,R)$ and $\langle {\mathcal{D}_n}^{k(n)}, Z, R\rangle$ are computationally indistinguishable. The proof follows because

$$\mathrm{Ent}({\mathcal{D}_n}^{k(n)}, Z, R) \geq p(n) + k(n)(a(n) + 7s(n)/8) + j(n),$$

and because by choice of $k(n)$ and $j(n)$ this is at least $p(n) + nk(n) + 1$. ■

Theorem 5 *If f is a* **R***-secure false entropy generator with false-entropy $s(n)$ then a $\Omega_{\mathcal{S}}(\mathbf{R})$-secure pseudo-random generator can be constructed from f.*

PROOF: Let $m'(n) = k'(n)(\lceil 8n/s(n) \rceil + 1) + 1$. Let $g'(i, x', r')$ be the generator obtained by assuming in Lemma 9 that $\mathrm{Ent}(f(X))$ is equal to $is(n)/8n$ when defining $a(n)$, then using the construction described in Theorem 1 (page 19, then using the construction of Proposition 11 (page 24) so that the output is of length $m'(n)$. Then, g' is a $(\mathrm{ilog}(8n/s(n)), k'(n)p'(n); m'(n))$-standard function. Define function

$$g''(x', r') = g'(\lceil 8\mathrm{Ent}(f(X))/s(n) \rceil, x', r').$$

Let $X' \in_{\mathcal{U}} \{0,1\}^{k'(n)}$ and $R' \in_{\mathcal{U}} \{0,1\}^{p'(n)}$. Combining Lemma 9, Theorem 1 and Proposition 11, it can be seen that g'' is pseudo-random. Let $y \in \{0,1\}^{(\lceil 8n/s(n) \rceil + 1) \times k'(n)}$ and define standard function

$$g'''(y, r') = \oplus_{i=0}^{\lceil 8n/s(n) \rceil} g'(i, y_{i+1}, r').$$

It follows from Lemma 7 (page 23) that g''' is a pseudo-random generator. ∎

8.2 One-way function to a false entropy generator

Let $f(x)$ be a **R**-secure $(n; \ell(n))$ one-way function. Let $h_r(x) = h(x, r)$ be a $(n, p(n); n + \mathrm{ilog}(2n))$-universal hash function. Let $i \in \{0, \ldots, n-1\}$, $y \in \{0,1\}^n$,

$$f'(x, i, r) = \langle f(x), h_r(x)_{\{1,\ldots,i+\mathrm{ilog}(2n)\}}, i, r \rangle,$$

$$\lambda(n) = \Pr[I \leq d(f(X))].$$

We assume for now that $\lambda(n)$ is computable in time polynomial in n. Let $k(n) = 125n^3$ and $m(n) = k(n)\lambda(n) - 2k(n)^{2/3}$. Let $h'_u(s) = h'(s, u)$ be a $(k(n), p'(n); m(n))$-universal hash function. Let $x' \in \{0,1\}^{k(n) \times n}$, $y' \in \{0,1\}^{k(n) \times n}$, $i' \in \{0,1\}^{k(n) \times \mathrm{ilog}(n)}$ and $r' \in \{0,1\}^{k(n) \times p(n)}$. $u \in \{0,1\}^{p'(n)}$. Define standard function

$$g(x', y', i', r', u) = \langle h'_u(x'_1 \cdot y'_1, \ldots, x'_{k(n)} \cdot y'_{k(n)}), f'^{k(n)}(x', i', r'), u \rangle$$

Theorem 6 *If f is a one-way function and $\lambda(n)$ is polynomial time computable then g is a false entropy generator with false entropy $10n^2$. The reduction is slight-preserving.*

PROOF: This follows directly by combining Lemma 10 and Lemma 11 below. ∎

Let $X' \in_{\mathcal{U}} \{0,1\}^{k(n) \times n}$, $Y' \in_{\mathcal{U}} \{0,1\}^{k(n) \times n}$, $I' \in_{\mathcal{U}} \{0,1\}^{k(n) \times \mathrm{ilog}(n)}$, $R' \in_{\mathcal{U}} \{0,1\}^{k(n) \times p(n)}$, $U \in_{\mathcal{U}} \{0,1\}^{p'(n)}$ and $Z \in_{\mathcal{U}} \{0,1\}^{m(n)}$. Let

- $\mathcal{D}_n = g(X', Y', I', R', U)$, and
- $\mathcal{E}_n$ is the same as $\mathcal{D}_n$ except that the first $m(n)$ bits are replaced with Z.

Lemma 10 $\text{Ent}(\mathcal{E}_n) \geq \text{Ent}(\mathcal{D}_n) + 10n^2$.

PROOF: The entropy of $\mathcal{D}_n$ and $\mathcal{E}_n$ excluding the first $m(n)$ bits is exactly the same. The additional entropy in the first $m(n)$ bits of $\mathcal{E}_n$ is equal to $m(n)$. An upper bound on the additional entropy in the first $m(n)$ bits of $\mathcal{D}_n$ is the additional entropy in $\langle X'_1 \cdot Y'_1, \ldots, X'_{k(n)} \cdot Y'_{k(n)} \rangle$. For each $j \in \{1, \ldots, k(n)\}$ where $I'_j < d(f(X'_j))$, the amount of entropy added by $X'_j \cdot Y'_j$ is at most 1. On the other hand, as in the proof of part (2) of the proof of Lemma 5 (see page 21), under the condition that $I'_j \geq d(f(X'_j))$, $X'_j \cdot Y'_j$ 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 $I'_j < d(f(X'_j))$ with probability $\lambda(n) - 1/n$, it follows that the additional entropy added by $X'_j \cdot Y'_j$ is at most $\lambda(n) - 1/2n$. Therefore, the additional entropy in the first $m(n)$ bits of $\mathcal{D}_n$ is at most $k(n)(\lambda(n) - 1/2n) = m(n) + 2k(n)^{2/3} - k(n)/2n \leq m(n) - 10n^2$ by choice of $k(n)$. ■

Lemma 11 *If f is a one-way function then $\mathcal{D}_n$ and $\mathcal{E}_n$ are computationally indistinguishable. The reduction is slight-preserving.*

PROOF: Let A be a distinguishing adversary with distinguishing probability $\delta(n)$, i.e., $|\text{Exp}[A(\mathcal{D}_n)] - \text{Exp}[A(\mathcal{E}_n)]| = \delta(n)$. We assume without loss of generality that $\text{Exp}[A(\mathcal{D}_n)] > \text{Exp}[A(\mathcal{E}_n)]$. Consider the oracle machine O which works as follows with respect to A.

Phase 1 : This consists of running $k(n)$ 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. These $k(n)$ pairs of distributions are used to invert f in the second phase.

Define $\mathcal{D}^0 = \mathcal{D}_n$ and $\mathcal{E}^0 = \mathcal{E}_n$. Let $B \in_{\mathcal{U}} \{0,1\}^{k(n)}$. Let $\rho = \delta(n)/(16k(n))$ and $t = 64n^3k(n)/(\rho \cdot \delta(n))$. Stage $j = 1, \ldots, k(n)$ works as follows: Randomly choose $c_j \in \{0,1\}$ so that $c_j = 1$ with probability $\lambda(n)$. Choose $\hat{x}_1, \ldots, \hat{x}_t \in_{\mathcal{U}} \{0,1\}^n$ and $\hat{i}_1, \ldots, \hat{i}_t \in_{\mathcal{U}} \{0, \ldots, n-1\}$. For each $m \in \{1, \ldots, t\}$, define $w_m = \langle \hat{x}_m, \hat{i}_m \rangle$ and let $\mathcal{D}^{j-1}_{c_j}(w_m)$ 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 to h' is set to

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

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 $\Pr[|\Delta_{c_j}^{j-1}(w_m) - \delta_{c_j}^{j-1}(w_m)| > \rho]$ is small(n). Let $m_0 \in \{1,\ldots,t\}$ 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 : This phase works as follows on input $\langle f'(w,r), b, y\rangle$. Pick $j \in_{\mathcal{U}} \{0,\ldots,k(n)-1\}$. 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 $j+1^{rst}$ input bit to h' is set to b and 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 $A(D)$.

Let $X \in_{\mathcal{U}} \{0,1\}^n$, $I \in_{\mathcal{U}} \{0,\ldots,n-1\}$, $R \in_{\mathcal{U}} \{0,1\}^{p(n)}$, $Y \in_{\mathcal{U}} \{0,1\}^n$, and $\beta \in_{\mathcal{U}} \{0,1\}$. Let $\mathcal{T}_n = \{\langle x,i\rangle | x \in \{0,1\}^n, i \in \{1,\ldots,d(f(x)\}\}$ and $W = \langle\tilde{X},\tilde{I}\rangle \in_{\mathcal{U}} \mathcal{T}_n$. The following two claims prove the lemma.

Claim 1 *If f is a $\mathbf{R}$-secure one-way function then $\langle f'(W,R), \tilde{X}\cdot Y, Y\rangle$ and $\langle f'(W,R), \beta, Y\rangle$ are $\Omega_{\mathcal{P}}(\mathbf{R})$-secure computationally indistinguishable, where $\beta \in_{\mathcal{U}} \{0,1\}$.*

PROOF: Let A be a $\mathbf{R}'$-breaking adversary for distinguishing the two distributions. Then, L^A is a $\mathbf{R}''$-breaking adversary for guessing $\tilde{X}\cdot Y$ given $\langle f'(W,R), Y\rangle$, where L is the oracle machine described in Proposition 6 and where $\mathbf{R}'' = \mathcal{O}_{\mathcal{L}}(\mathbf{R}')$. Then, from Proposition 5, there is an oracle machine M such that M^A is a $\mathbf{R}'''$-breaking adversary for inverting f' on input $f'(W,R)$, where $\mathbf{R}''' = \mathcal{O}_{\mathcal{P}}(\mathbf{R}'')$. Finally, we use the same idea as in Lemma 5, i.e., the success probability of M^A on input $\langle f(x), \alpha, i, R\rangle$ for $\alpha \in_{\mathcal{U}} \{0,1\}^{i+\mathrm{ilog}(2n)}$ is at least inverse polynomial in the success probabilty of M^A on input $f'(x,i,R)$ for each fixed $\langle x,i\rangle \in \mathcal{T}_n$. Consider the following oracle machine N: N^A on input $f(x)$ chooses $i \in_{\mathcal{U}} \{0,\ldots,n-1\}$, $\alpha \in_{\mathcal{U}} \{0,1\}^{i+\mathrm{ilog}(2n)}$, and $r \in_{\mathcal{U}} \{0,1\}^{p(n)}$ and runs M^A on input $\langle f(x), \alpha, i, r\rangle$. Since $\Pr[\langle X,I\rangle \in \mathcal{T}_n] \geq 1/n$, 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. ■

Claim 2 *O^A distinguishes between $\langle f'(W,R), \tilde{X}\cdot Y, Y\rangle$ and $\langle f'(W,R), \beta, Y\rangle$ with probability at least ρ.*

PROOF: Let $w = \langle x,i\rangle$, $D^j(w,r,b,y) = \mathrm{Exp}[\mathcal{D}^j(w,r,b,y)]$ and $E^j(w,r,y) = \mathrm{Exp}[\mathcal{E}^j(w,r,y)]$. Then,

$$\Pr[O^A(f'(w,r), b, y) = 1] = 1/2 + (D^j(w,r,b,y) - E^j(w,r,y))/2.$$

Also, it follows directly from the definitions that,

$$\mathrm{Exp}[D^j(w, R, x \cdot Y, Y) - E^j(w, R, Y)] = \delta_0^j(w),$$

and

$$\mathrm{Exp}[D^j(w, R, \beta, Y) - E^j(w, R, Y)] = \delta_1^j(w).$$

Thus, the distinguishing probability of O^A is

$$\mathrm{Exp}[O^A(f'(W, R), \tilde{X} \cdot Y, Y)] - \mathrm{Exp}[O^A(f'(W, R), \beta, Y)] = \mathrm{Exp}[\delta_0^j(W) - \delta_1^j(W)]/2.$$

Let $\epsilon^j = \mathrm{Exp}[\delta_0^j(W) - \delta_1^j(W)]$. To prove the claim, it is sufficient to show that, for $j \in_{\mathcal{U}} \{0, \ldots, k(n) - 1\}$, $\mathrm{Exp}[\epsilon^j] \geq 2\rho$, or equivalently,

$$\mathrm{Exp}[\sum_{j \in \{0,\ldots,k(n)-1\}} \epsilon^j] \geq 2\rho \cdot k(n) = \delta(n)/8. \tag{1}$$

The expectation is over the random choices of O^A in the first phase. We prove this by showing below that

(a) $\mathrm{Exp}[\delta^{k(n)})] = \mathrm{small}(n)$. The expectation is over the random choices of O^A in the first phase.

(b) $\mathrm{Exp}[\delta^j - \delta^{j+1}] \leq \epsilon^j + 4\rho$. 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), it follows that

$$\begin{aligned}
\delta(n)/2 &< \delta(n) - \mathrm{Exp}[\delta^{k(n)}] \\
&= \sum_{j \in \{0,\ldots,k(n)-1\}} \mathrm{Exp}[\delta^j - \delta^{j+1}] \\
&\leq 4k(n)\rho + \mathrm{Exp}[\sum_{j \in \{0,\ldots,k(n)-1\}} \epsilon^j] \\
&= \delta(n)/4 + \mathrm{Exp}[\sum_{j \in \{0,\ldots,k(n)-1\}} \epsilon^j],
\end{aligned}$$

and this proves the bound in Equation 1. Thus, it suffices to prove (a) and (b) above.

PROOF of (a) : Since $\Pr[c_j = 1] = \lambda(n)$, applying Chernoff bounds we get that, with probability at least $1 - \mathrm{small}(n)$, $\sum_{j \in \{0,\ldots,k(n)-1\}} c_j \geq k(n)\lambda(n) - (k(n))^{2/3} = m(n) + k(n)^{2/3}$. The min-entropy of the input to h' conditional on the rest of the bits of $\mathcal{D}^{k(n)}$ is at least $\sum_{j \in \{0,\ldots,k(n)-1\}} c_j$. So, if this sum is at least $m(n) + k(n)^{2/3}$, applying Lemma 1 (page 16), the two distributions $\mathcal{D}^{k(n)}$ and $\mathcal{E}^{k(n)}$ are statistically indistinguishable within small(n). Thus, $\delta^{k(n)} = \text{Exp}[A(\mathcal{D}^{k(n)})] - \text{Exp}[A(\mathcal{E}^{k(n)})] = \text{small}(n)$.

PROOF of (b) : Let $\bar{W} \in_{\mathcal{U}} \bar{\mathcal{T}}_n$. Then,

$$\begin{aligned}
\delta^j &= \lambda(n)\text{Exp}[\delta_0^j(W)] + (1-\lambda(n))\text{Exp}[\delta_0^j(\bar{W})] \\
&= \lambda(n)\text{Exp}[\delta_1^j(W)] + \lambda(n)(\text{Exp}[\delta_0^j(W)] - \text{Exp}[\delta_1^j(W)]) \\
&\quad + (1-\lambda(n))(\text{Exp}[\delta_0^j(\bar{W})]) \\
&= \lambda(n)\text{Exp}[\delta_1^j(W)] + \lambda(n)\epsilon^j + (1-\lambda(n))(\text{Exp}[\delta_0^j(\bar{W})]) \\
&\leq \epsilon^j + \lambda(n)\text{Exp}[\delta_1^j(W)] + (1-\lambda(n))(\text{Exp}[\delta_0^j(\bar{W})])
\end{aligned}$$

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

$$\Pr[\max_{1\leq j\leq t}\{\delta_c^j(w_m)\} \geq \max\{\text{Exp}[\delta_c^j(W)], \text{Exp}[\delta_c^j(\bar{W})]\} - \rho] \geq 1 - \text{small}(n).$$

Then, with probability $1-\text{small}(n)$, $\Delta_c^j(w_m)$ is within ρ of the corresponding $\delta_c^j(w_m)$, and thus

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

with probability $1-\text{small}(n)$. Let δ_c^{j+1} be the value of δ^{j+1} conditional on $c_{j+1} = c$. From this we can conclude that

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

Since $c_{j+1} = 1$ with probability $\lambda(n)$,

$$\begin{aligned}
\text{Exp}[\delta^{j+1}] &= \lambda(n)\text{Exp}[\delta_1^{j+1}] + (1-\lambda(n))\text{Exp}[\delta_0^{j+1}] \\
&\geq \lambda(n)\text{Exp}[\delta_1^j(W)] + (1-\lambda(n))(\text{Exp}[\delta_0^j(\bar{W})]) + 4\rho.
\end{aligned}$$

This complete the proof of part (b). Claim (2) follows from (a) and (b), and then Lemma 11 follows from Claims (1) and (2). ■ ■

Theorem 7 *There are one-way functions iff there are pseudo-random generators.*

PROOF: That pseudo-random generators imply one-way functions follows from [24, Levin]. The converse now follows from Theorem 6 and Theorem 5, except for the assumption about knowing the exact value of $\lambda(n)$. However we can deal with this problem exactly as we dealt with the unknown false entropy in Theorem 5 (page 26), using Proposition 11 (page 24), and Lemma 7 (page 23) ■

9 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 pseudo-random generator, e.g., the work of ([9, Goldreich, Goldwasser, Micali], [26, Luby, Rackoff], [13, Goldreich, Micali, Wigderson], [28, Naor]), show that applications ranging from private key encryption to zero-knowledge proofs can be based on a pseudo-random generator. The results presented in this paper show that these same protocols can be based on any one-way function. [29, Naor, Yung], give a signature scheme that can be based on any one-way permutation, and [34, Rompel], substantially improves this by basing such a scheme on any one-way function.

[18, Impagliazzo, Luby] 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. [30, Ostrovsky, Wigderson] show 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.

Some applications seem unlikely to be shown possible based on any one-way function, e.g., [21, Impagliazzo, Rudich] 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 pseudo-random generator based on any one-way function increases the size of the input by a large polynomial amount and thus is only slight-preserving. This is not good news for practical applications; it would be nice to have a general polynomial-preserving or a linear-preserving reduction.

10 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, Lance for suggesting a version of Lemma 2 and Oded and Hugo for exposing the fourth author to their wealth of insights on this problem.

References

[1] 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. A preliminary version appeared in 25^{th} *FOCS*, 1984, pp. 449-457.

[2] Bennett, C., Brassard, G., Robert, J., "Privacy Amplification by Public Discussion", *Siam J. on Computing*, Vol. 17, No. 2, 1988, pp. 210-229.

[3] Blum, M., "Independent Unbiased Coin Flips From a Correlated Biased Source: A Finite State Markov Chain", 25^{th} *FOCS*, 1984, pp. 425-433.

[4] 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. A preliminary version appeared in 23^{rd} *FOCS* 1982.

[5] 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.

[6] Carter, J., and M. Wegman, "Universal Classes of Hash Functions", *JCSS*, 1979, Vol. 18, pp. 143-154.

[7] Chor, B., and O. Goldreich, "Unbiased Bits from Sources of Weak Randomness and Probabilistic Communication Complexity", *SIAM J. on Computing*, Vol. 17, 1988. A preliminary version appeared in 25^{th} *FOCS* 1985.

[8] W. Diffie and M. Hellman, "New directions in cryptography", *IEEE Trans. Inform. Theory*, Vol. 22, 1976, pp. 644-654.

[9] Goldreich, O., S. Goldwasser, and S. Micali, "How to Construct Random Functions", *J. of ACM*, Vol. 33, No. 4, 1986, pp. 792-807. A preliminary version appeared in 24^{th} *FOCS*, 1984.

[10] Goldreich, O., Impagliazzo, R., Levin, L., Venketesan, R., Zuckerman, D., "Security Preserving Amplification of Hardness", Proceedings of the 31st IEEE Symposium on Foundations of Computer Science, pp. 318-326, 1990.

[11] Goldreich, O., Krawczyk, H. and Luby, M., "On the Existence of Pseudorandom Generators", preliminary version in 29^{th} *FOCS*, 1988, pp. 12-24. journal version in *SIAM J. on Computing*, Vol. 22, No. 6, December, 1993, pp. 1163-1175.

[12] Goldreich, O., and L.A. Levin, "A Hard-Core Predicate for any One-way Function", 21^{rst} *STOC*, 1989, pp 25-32.

[13] Goldreich, O., Micali, M. and Wigderson, A., "Proofs that Yield Nothing but Their Validity and a Methodology of Cryptographic Protocol Design", 27^{th} *FOCS*, 1986, pp. 174-187, *Tech. Report TR498*, Comp. Sci. Dept., Technion, submitted to *JACM*.

[14] Goldwasser, S. and Micali, S., "Probabilistic Encryption," *JCSS*, Vol. 28, No. 2, April 1984, pp. 270-299. A preliminary version appeared in 14^{th} *STOC*, 1982.

[15] 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. A preliminary version appeared in 17^{th} *STOC*, 1985.

[16] Håstad, J. "Pseudo-Random Generators under Uniform Assumptions", 22^{nd} *STOC*, 1990, pp 395-404.

[17] Herzberg, A., Luby, M., "Public Randomness in Cryptography", proceedings of *CRYPTO 1992*, *ICSI technical report* TR-92-068, October, 1992.

[18] Impagliazzo, R. and Luby, M., "One-way functions are essential for information based cryptography," 30^{th} *FOCS*, 1989.

[19] Impagliazzo, R., Levin, L. and Luby, M, "Pseudo-random number generation from one-way functions", 21^{rst} *STOC*, 1989, pp 12-24.

[20] Impagliazzo, R., Naor, M., "Efficient Cryptographic Schemes Provably as Secure as Subset Sum", 30^{th} *FOCS*, 1989, pp. 236-241.

[21] Impagliazzo, R. and Rudich, S., "Limits on the Provable Consequences of One-way Functions", 21^{rst} *STOC*, 1989, pp 44-56.

[22] Knuth, D., *Semi-Numerical Algorithm, The Art of Computer Programming*, Addison-Wesley, Second Edition, Vol. 2, 1981.

[23] Krawczyk, H., "How to Predict Congruential Generators," accepted for publication in *J. of Algorithms*, presented at *CRYTPO*, 1989.

[24] Levin, L.A., "One-way Function and Pseudorandom Generators", *Combinatorica*, Vol. 7, No. 4, 1987, pp. 357-363. A preliminary version appeared in 17^{th} *STOC*, 1985.

[25] Luby, M., "Pseudo-randomness and Applications", monograph in preparation for publication by Princeton University Press.

[26] 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. A preliminary version appeared in 18^{th} *STOC*, 1986.

[27] McInnes, J., "Cryptography Using Weak Sources of Randomness," *Tech. Report 194/87*, U. of Toronto, 1987.

[28] Naor, M., personal communication, 1988.

[29] Naor, M. and Yung, M., "Universal One-way Hash Functions and Their Applications", 21^{rst} *STOC*, 1989, pp 33-43.

[30] Ostrovsky, R and Wigderson, A., "One-way Functions are Essential for Non-Trivial Zero-Knowledge", 2^{nd} *Israel Symposium on the Theory of Computing and Systems*, 1993, pp. 3-17.

[31] Plumstead, J., "Inferring a Sequence Generated by a Linear Congruence", 23^{rd} *FOCS*, 1982, pp 153-159.

[32] Renyi, A., **Probability Theory**, North-Holland, Amsterdam, 1970.

[33] R. Rivest, A. Shamir and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", *Comm. of the ACM*, Vol. 21, 1978, pp. 120-126.

[34] Rompel, J., "One-way Functions are Necessary and Sufficient for Secure Signatures", 22^{nd} *STOC*, 1990, pp 387-394.

[35] Shannon, C., "A Mathematical Theory of Communication", *Bell Systems Technical Journal*, 27, 1948, pp. 379-423 and pp. 623-656.

[36] Santha, M. and Vazirani, U., "Generating Quasi-random Sequences from Slightly-random Sources", *JCSS*, Vol. 33, No. 1, 1986. A preliminary version appeared in 25^{th} *FOCS*, 1984, pp. 434-440,

[37] Sipser, M., "A Complexity Theoretic Approach to Randomness", 15^{th} *STOC*, 1983, pp. 330-335.

[38] 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. A preliminary version appeared in 17^{th} *STOC*, 1985, pp. 366-378.

[39] Vazirani, U. and Vazirani, V., "Efficient and Secure Pseudo-Random Number Generation", 25^{th} *FOCS*, 1984, pp. 417-428.

[40] Vazirani, U. and Vazirani, V., "Random Polynomial Time is Equal to Slightly-random Polynomial Time", 26^{th} *FOCS*, 1985, pp. 417-428. Submitted to *JACM*.

[41] Yao, A.C., "Theory and Applications of Trapdoor Functions", 23^{rd} *FOCS*, 1982, pp. 80-91.