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

Johan Håstad* Russell Impagliazzo†

Leonid A. Levin‡ Michael Luby§

TR-91-068

December, 1991

Abstract

We show how to construct a pseudo-random generator from any one-way function. In contrast, previous works have constructed pseudo-random generators only from one-way functions with special structural properties. Our overall approach is different in spirit from previous work; we concentrate on extracting and smoothing entropy from a single iteration of the one-way function using universal hash functions.

*Royal Institute of Technology, Stockholm, Sweden. Research supported by the Swedish National Board for Technical Development

†Department of Computer Science, University of California at San Diego. Research done while at U.C.Berkeley, research partially supported by NSF grant CCR 88-13632

‡Computer Science Department, Boston University. Research supported by NSF grant DCR-8607492

§International Computer Science Institute, Berkeley, California. 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

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 by the fact that it can be used to construct a private key cryptosystem that is secure even against chosen plaintext attack, and it can be used to save random bits and allows reproducibility of results 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 much longer string $g(X)$ that "looks" just like a random string to *any* polynomial time adversary that is allowed to examine $g(X)$. Thus, a pseudo-random number generator can be used to efficiently convert a small amount of true randomness into a much longer string that is indistinguishable from a truly random string of the same length to any polynomial time adversary.

The notion given here of a pseudo-random generator should be contrasted with the classical notion of a pseudo-random generator, e.g. [20, Knuth]. A classical pseudo-random generator is required to pass a particular set of statistical tests, but does not necessarily satisfy the more general requirement that it pass all polynomial time tests. This is a 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 [5, Blum Micali], where they define the notion of pseudo-random generator used here and show that there is a pseudo-random generator based on the difficulty of the discrete log problem. Another example of the central importance of pseudo-random generators is that they can be used to reduce the number of random bits needed for any probabilistic polynomial time algorithm, and they allow deterministic simulation of any polynomial time probabilistic algorithm in subexponential time [35, Yao]. In contrast, [27, Plumstead] and [21, Krawczyk] show that there is a polynomial time statistical test which the classical linear congruential generator does not pass, although it does pass a variety of standard statistical tests. In a similar vein, [4, Blum Blum Shub] describe two natural and seemingly similar generators. They show that one of the generators passes standard statistical tests but is not pseudo-random in the sense used here, whereas the other generator is pseudo-random in the sense used here if it turns out that factoring the product of a pairs of randomly chosen primes is difficult.

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. Intuitively, a function f is *one-way* if: (1) f is polynomial time computable; (2) given $f(X)$ for a randomly distributed X, it is not possible on the average to find an inverse X' such that $f(X') = f(X)$ in polynomial time. It has not been proven that there are any one-way functions, but there are a number of problems from number theory (e.g. factoring and the discrete log problem), coding theory, graph theory and combinatorial theory that are candidates for problems that might eventually be proven to be one-way functions.

[5, Blum Micali] are the first to introduce a definition of a pseudo-random generator. [35, Yao] introduces the notion of a pseudo-random generator we use in this paper, which seems more natural than the definition in [5, Blum Micali], and shows that the two definitions are equivalent.

[5, Blum Micali] shows that a specific conjectured one-way function, the discrete log problem, can be used to construct a pseudo-random number generator. Their construction has the property that if the generator is not pseudo-random then the function from which it is constructed can be efficiently inverted and consequently is not one-way. All subsequent constructions share this

property.

Subsequent to [5, Blum Micali], several results show how to construct pseudo-random number generators based on general classes of one-way functions with special properties. [35, Yao] generalizes the results of [5, Blum Micali] by showing how to construct a pseudo-random generator from any one-way permutation (i.e. for each y in the range there is a unique x such that $f(x) = y$). [1, Alexi Chor Goldreich Schnorr] show how to construct a pseudo-random generator based either on the difficulty of factoring or on the difficulty of inverting the RSA function. Their pseudo-random generators have trapdoors, which is crucial for constructing public key cryptosystems. [22, Levin] introduces one-way functions with special properties and shows that these one-way functions are necessary and sufficient for the construction of a pseudo-random generator.[1] Up till this point, the conjectured one-way functions that fulfill the special properties are all number theoretic in nature. [9, 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.

In this paper we show how to construct a pseudo-random generator from *any* one-way function.

Previous methods implicitly rely on constructing a function that has a "hidden" but "meaningful" bit of the output from one-way functions with special structural properties. Intuitively, an output bit of a function is *hidden* if it is unpredictable by a polynomial time algorithm given all the other output bits, and the bit is *meaningful* if it can be predicted from the other output bits by an all-powerful algorithm. [10, Goldreich Levin] provide a simple and elegant way of constructing a function with a hidden bit from any one-way function. Their result radically simplifies the previous constructions of pseudo-random number generators 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 quite different in spirit from previous constructions of pseudo-random generators from 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 from a single iteration of the one-way function.

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 information, i.e. if $\mathcal{D}$ is a probability distribution on inputs to a function g then the Shannon entropy of $g(\mathcal{D})$ is at most the Shannon entropy of $\mathcal{D}$. On the other hand, a pseudo-random generator g is a function that "appears" to violate this fundamental law in the sense that $g(\mathcal{D})$ "appears" to have more Shannon entropy than $\mathcal{D}$. To make this more precise, we say two probability distributions $\mathcal{D}$ and $\mathcal{E}$ are *computationally indistinguishable* if no polynomial time algorithm can distinguish between x randomly chosen from $\mathcal{D}$ and y randomly chosen from $\mathcal{E}$. If g is a pseudo-random generator then $g(\mathcal{D})$ and $\mathcal{E}$ are computationally indistinguishable, where $\mathcal{D}$ is the distribution on the inputs to g and where $\mathcal{E}$ is the uniform distribution on strings the same length as the output of g. In this case,

[1] Previous to [22, Levin], attention focused only on one-way permutations, in which case the task of inverting $f(x)$ is to find x uniquely. The natural extension of the notion of inverting $f(X)$ in the case when f is not a permutation is to find any X' such that $f(X') = f(X)$.

we say that the *computational entropy* of g is the Shannon entropy of $\mathcal{E}$. Because the output of g is longer than its input, the computational entropy of g is greater than the Shannon entropy of its input, and in this sense g amplifies entropy. The notion of computational entropy provides one of the main conceptual tools in our paper.

Just as Shannon entropy quantifies the amount of randomness in a distribution, computational entropy quantifies the amount of "apparent" randomness (to a polynomial time algorithm) of a distribution. For example, we can relax the notion of a generator g being pseudo-random by allowing $g(\mathcal{U})$ to be computationally indistinguishable from a probability distribution $\mathcal{D}$ that is not necessarily the uniform distribution, where $\mathcal{D}$ has more Shannon entropy than $\mathcal{U}$. In this case, we call g a *pseudo-entropy generator* because the computational entropy of g is greater than the Shannon entropy of its input.

The notion of computational entropy is also useful in the case when the Shannon entropy of $\mathcal{D}$ is not necessarily greater than that of $\mathcal{U}$. We say that g has *false* entropy if the computational entropy of $g(\mathcal{U})$ exceeds the Shannon entropy of $g(\mathcal{U})$ (but not necessarily the Shannon entropy of $\mathcal{U}$).

We use computational entropy in constructions of pseudo-random generators as follows. 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.[2] The first step uses in a fundamental way the work of [10, Goldreich Levin].

In [10, Goldreich Levin], it turns out that the easily computable bit that is hidden is a random inner product of the input bits. This random inner product can be viewed as a hash function from many bits to one bit. One of our main technical lemmas (Lemma 13 on page 15), which roughly speaking can be viewed as follows, involves hashing. Let $\mathcal{H}_{n,l}$ be a family of universal-two hash functions [6, Carter Wegman] mapping n bit strings to l bit strings. Suppose $\mathcal{D}$ is the uniform distribution on an arbitrary subset of 2^m strings of length n, where m is slightly larger than l. Then, the distribution $H \circ H(Z)$ is roughly the uniform distribution on strings of length $|H| + l$, where H is uniformly distributed in $\mathcal{H}_{n,l}$ and Z is independently distributed according to $\mathcal{D}$. One use of this hashing lemma is to compact a distribution that is uniform on 2^m arbitrary strings of length n to a distribution that is uniform on l bit strings while maintaining the entropy of the randomly selected hash function used to do the compaction. Another use is to extract entropy out of the input bits that has been lost through the application of the one-way function as follows. Suppose f is a one-way function such that each string in the range of f has 2^l inverses. Then, this hashing lemma can be used to prove that the function $f'(X \circ H) = f(X) \circ H \circ H(X)$ is a one-way function that is close to a one-to-one function, i.e. the entropy of the output of f' is close to $n + |H|$.

The current paper is a combination of the results announced in the conference papers [16, Impagliazzo Levin Luby] and [14, Håstad].

1.1 Notation

If x and y are bit strings then $|x|$ is the length of x, $x \circ y$ is the concatenation of x and y, $x_{(i)}$ is the i^{th} bit of x, $x_{\leftarrow i}$ is the first i bits of x, $x_{i\rightarrow}$ is all but the first i bits of x. If x and y are two equal length bit strings then $x \odot y$ is the inner product mod 2 of x and y and $x \oplus y$ is the vector

[2] The presentation of these results in the body of the paper is in reverse order.

sum mod 2 (i.e. bitwise parity) of x and y. If a is a number, then $|a|$ is the absolute value of 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 S is a set then $|S|$ is the number of elements in S.

2 Adversaries

In this section, we formally introduce notions related to parameterizing the security of a primitive. In this paper, 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. In both cases, there is a notion of the *success rate* of the adversary. In the case of one-way functions, the success rate is the probability that the adversary finds an inverse of the function, and in the case of pseudo-random generators the success rate is the distinguishing probability of the adversary. A standard definition of successful adversary is one that allows polynomial resources, i.e. the running time is restricted to polynomial in the size of the input and the success rate is at least inverse polynomial in the input size. We choose instead to let the resources of a successful adversary be defined in terms of an arbitrary resource class, where the standard definition of a successful adversary corresponds to the resource class that is the set of all polynomial functions. Allowing general resource classes is important because in some situations it makes sense that a successful adversary must run in polynomial time and have a inverse polynomial success rate, whereas in other situations it makes sense to allow a successful adversary considerably more latitude to be considered successful, e.g. running time $2^{O(\log^c(n))}$ and success rate $1/2^{O(\log^c(n))}$ for some constant $c > 1$.

DEFINITION 2.1 (resources) *A* resource class R *is class of functions from* $\mathcal{N}$ *to* $\mathcal{N}$ *that includes the identity function* $r(n) = n$. *The resource class is closed under the following upward operation: If* $r \in R$ *then* $r'(n) = r(n^2)$ *is in* R. *The resource class is closed downwards also, i.e. if* $r \in R$ *and* r' *is any function with the property that* $r'(n) \leq r(n)$ *then* $r' \in R$.

We do not consider resource classes that contain a function that is exponential in n, i.e. $r(n) = 2^n$ is not in R. This is because if such a large r is in R then it is easy to construct an adversary that performs an exhaustive search that is always successful. Note that polynomial resources is the minimal resource class.

We consider both uniform and non-uniform adversaries. The difference between the two types of adversaries is that, a uniform adversary is a probabilistic Turing machine with a time restriction, whereas a non-uniform adversary is an infinite sequences of Boolean circuits, one circuit for each input length, with a size restriction on the circuit. At first glance, a non-uniform adversary seems too strong of an attack to allow for a pseudo-random generator, because it seems natural that the adversary can be thought of as a probabilistic Turing machine. However, a pseudo-random generator that is only secure against uniform adversaries when used to build a private key cryptosystem or when used to generate the random bits used in a Monte Carlo simulation may not be sufficiently secure. In the case of a private key cryptosystem, the time allowed for computation by the adversary before the cryptosystem is used may be much greater than the allowable time during the use of the cryptosystem. The result of the preprocessing can then be used by the adversary to break the cryptosystem within the allowable time. In the case of a Monte Carlo simulation algorithm, there is usually a problem input in addition to the random bit input. Since there is no restriction on how problem instances are generated, the safest restriction to assume

is no restriction at all, i.e. it is safest to assume that the problem instances are generated in a possibly non-uniform manner. This is exactly the reason that a pseudo-random generator with respect to non-uniform adversaries is used to prove that $BPP \subset DTime(2^{n^\epsilon})$ for every $\epsilon > 0$ [35, Yao]. A non-uniform adversary is equivalent to a uniform adversary that has extra input bits generated non-uniformly (see [19, Karp Lipton]).

DEFINITION 2.2 (uniform adversary) *A* uniform adversary *is a probabilistic Turing machine. The* time bound $T(n)$ *for a uniform adversary is the maximum running time on inputs of length* n.

DEFINITION 2.3 (non-uniform adversary) *A* non-uniform adversary *is a pair of Turing machines* A *and* M. *A is a preprocessing algorithm that on input n produces a string $A(n)$. The running time of A is not limited. Algorithm M accepts as input $x \in \{0,1\}^n$ and $A(n)$. The* time bound $T(n)$ *for a non-uniform adversary is the maximum running time of M over all inputs $x \in \{0,1\}^n$ and $A(n)$.*

DEFINITION 2.4 (unbounded adversary) *An* unbounded adversary A *is an adversary (uniform or non-uniform, it doesn't matter) with* unbounded time and space resources.

From the results of [19, Karp Lipton], a non-uniform adversary with time bound function in resource class R is equivalent to a (recursive) family of circuits with the size of the circuit for inputs of length n equal to $r(n)$ where r is some function in resource class R.

In all definitions and theorems, there are really two statements being made simultaneously, one with respect to uniform adversaries and the other with respect to non-uniform adversaries.

DEFINITION 2.5 (feasible adversary) *A function is* feasible *if it is in the resource class R. An adversary is* feasible *if the time bound function is feasible.*

DEFINITION 2.6 (negligible) *A function $p : \mathcal{N} \rightarrow \mathcal{N}$ is* negligible *with respect to resource class R if for all $r \in R$, for almost all n, $p(n) \leq 1/r(n)$.*

For the remainder of the paper, unless stated otherwise, the resource class R is fixed but arbitrary, and explicit mention of the resource class is suppressed when possible.

DEFINITION 2.7 (successful adversary) *A* successful adversary *is a feasible adversary that has non-negligible success rate.*

DEFINITION 2.8 (oracle Turing machine) *A probabilistic Turing machine M is an* oracle *machine if it makes calls to an adversary that is specified in advance. We let M^A denote M making calls to adversary A.*

In this paper, we prove statements of the form "There is a generic construction that takes any particular instance f of primitive A and produces a particular instance g of primitive B", e.g. our main theorem can be stated as "There is a generic construction that takes any one-way function f and produces a pseudo-random generator g". The construction in all cases is *uniform* in the following sense:

DEFINITION 2.9 (uniform reduction) *A reduction from f to g is* uniform *if there is an oracle Turing machine M such that if A is a successful adversary for g then M^A is a successful adversary for f.*

The reduction has the property that if A is a non-uniform adversary then so is M^A, whereas if A is a uniform adversary then so is M^A. The important point is that irrespective of the uniformity of the adversaries, the conversion machine M is always *uniform*.

To see the importance of this property, consider the following example. Suppose that f is the one-way function based on the difficulty of factoring a composite number that is the product of a pair of large randomly chosen primes, and g is the pseudo-random generator resulting from the construction. Suppose that someone managed to find a probabilistic polynomial time Turing machine A for distinguishing the output of the pseudo-random generator from truly random bits. Then, M^A is a probabilistic polynomial time Turing machine that factors a non-negligible fraction of the composite numbers.

Now instead suppose someone managed to find in exponential time a polynomial size circuit family A that distinguishes the output of the pseudo-random generator from truly random bits. Then, M^A is a polynomial size circuit that factors a non-negligible fraction of the composite numbers. However, M^A is constructed in *polynomial time* from A. Note that here the onus is on the adversary; the adversary may spend exponential time finding A if it exists. If the adversary is unsuccessful then we can use g securely, and on the other hand if the adversary finds an A then we can in only polynomial time (as opposed to exponential time) find the small circuit M^A to solve the factoring problem.

Often in our proofs we argue that the probability that a particular event occurs is very small, i.e. exponentially small in a size parameter n. These probabilities are smaller than inversely proportional to any function in the resource class, and thus they can be safely disregarded.

DEFINITION 2.10 (exponentially small in n) *If there is a positive constant c such that for all sufficiently large $n \in \mathcal{N}$, $a(n) \leq 2^{-cn}$, then for notational convenience we use* $\exp(-n)$ *in place of $a(n)$.*

3 One-way function

We first introduce some notation which is used to define one-way functions and also used throughout the remainder of the paper.

DEFINITION 3.1 (functions) *A* length (function) *$l(n)$ is a polynomial in n time computable monotone increasing function from $\mathcal{N}$ to $\mathcal{N}$ such that $l(n)$ is polynomial in n. A* function *f with input length $m(n)$ and output length $l(n)$ specifies for each $n \in \mathcal{N}$ a function $f_n : \{0,1\}^{m(n)} \rightarrow \{0,1\}^{l(n)}$. For simplicity, we write $f(x)$ in place of $f_n(x)$. We say that f is* polynomial time computable *if there is a polynomial time Turing machine that on input $x \in \{0,1\}^{l(n)}$ computes $f(x)$.*

DEFINITION 3.2 (siblings and preimages) *For all $x \in \{0,1\}^n$ and $x' \in \{0,1\}^n$, we say that x and x' are* siblings *if $f(x) = f(x')$. We say that x is a preimage of y if $f(x) = y$.*[3] *We let $f^{-1}(y)$ be*

[3]For notational simplicity, we make the convention that a preimage is always a bit string of length n.

the set of preimages of y, i.e.

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

Then, $|f^{-1}(y)|$ is the number of n-bit preimages of y.

DEFINITION 3.3 (probability notation) *A* probability ensemble $\mathcal{D}$ *with length $m(n)$ assigns to each positive integer n a probability distribution $\mathcal{D}_n$ on bit strings of length $m(n)$. We use capital letters and Greek letters to denote random variables and random events, and we use uncapitalized letters to indicate particular values for variables. We use* $\mathrm{Exp}[X]$ *to denote the expected value of X, and* $\Pr[E]$ *denotes the probability that event E occurs. We use the notation $X \in_{\mathcal{D}_n} \{0,1\}^{m(n)}$ to mean that random variable X is randomly distributed in $\{0,1\}^{m(n)}$ according to $\mathcal{D}_n$, and we use the notation $x \in_{\mathcal{D}_n} \{0,1\}^{m(n)}$ to mean that string x is randomly chosen in $\{0,1\}^{m(n)}$ according to $\mathcal{D}_n$. For $S \subseteq \{0,1\}^{m(n)}$, $\mathcal{D}_n[S]$ is the sum over all $x \in S$ of the probability of x with respect to $\mathcal{D}_n$. When S is a set then the notation $X \in_{\mathcal{U}} S$ means that X is randomly and uniformly distributed in S. The* uniform ensemble $\mathcal{U}$ *assigns to each positive integer n the uniform probability distribution $\mathcal{U}_n$ on strings of length n.*

DEFINITION 3.4 (polynomial samplable) *We say that $\mathcal{D}$ is* polynomial samplable *if there is a polynomial time Turing machine M with input length $k(n)$ and output length $m(n)$ such that, for each $n \in N$, $M(X) \in_{\mathcal{D}_n} \{0,1\}^{m(n)}$ when $X \in_{\mathcal{U}} \{0,1\}^{k(n)}$.*

Define $f(\mathcal{D})$ to be the probability ensemble with length $l(n)$, where $f(\mathcal{D}_n)$ is the probability distribution defined by the random variable $f(X)$ when $X \in_{\mathcal{D}_n} \{0,1\}^{m(n)}$.

Hereafter, unless stated otherwise, we use the following conventions. Both f and g are polynomial time computable functions. If no length functions are specified for a function then the input length is n and the output length is $l(n)$. Both $\mathcal{D}$ and $\mathcal{E}$ are probability ensembles. If no length function is specified for a probability ensemble then the length is n.

Intuitively, a function f is *one-way* if it is easy to compute but hard to invert, i.e. given x the value of $f(x)$ can be computed in polynomial time but every feasible adversary that receives as input $f(X)$ when $X \in_{\mathcal{U}} \{0,1\}^n$ can output X' such that $f(X') = f(X)$ with only negligible probability. It has not yet been proven that one-way functions exist (if $P{=}NP$ then they certainly do not exist, but even if $P \neq NP$ it is not clear if they exist), but 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 discrete log modulo a large randomly chosen prime [5, Blum Micali], quadratic residuosity [4, Blum Blum Shub], factoring a composite number N that is the product of large randomly chosen primes ([33, Vazirani Vazirani] based on the generator suggested in [4, Blum Blum Shub] and proved by adapting the techniques of [1, Alexi Chor Goldreich Schnorr]), problems from coding theory [9, Goldreich Krawczyk Luby] and the subset sum problem for appropriately chosen parameters [17, Impagliazzo Naor].

We give two definitions of functions that are one-way in some sense. For one-way functions, a successful adversary is required to output some preimage of $f(X)$ with non-negligible probability. This is the primary definition used in this paper. For somewhat one-way functions, a successful adversary is required to output some preimage of $f(X)$ almost all the time. In other words an adversary is unsuccessful if for an a priori given constant c the probability that it fails to output some preimage of $f(X)$ is at least n^{-c}. The reason for considering somewhat one-way functions

is that they seem to be more likely to occur naturally than one-way functions, but on the other hand, as Proposition 2 (page 8) shows, a one-way function can be constructed from any somewhat one-way function.

DEFINITION 3.5 (somewhat one-way function) *We say that f is* somewhat one-way on $\mathcal{D}$ *if, for some constant $c > 0$, for every feasible adversary A, the inverting probability* $\Pr[f(X) = f(A(f(X)))]$ *is at most $1 - 1/n^c$ when $X \in_{\mathcal{D}_n} \{0,1\}^n$.*

DEFINITION 3.6 (one-way function) *We say that f is* one-way on $\mathcal{D}$ *if, for every feasible adversary A, the inverting probability* $\Pr[f(X) = f(A(f(X)))]$ *is negligible when $X \in_{\mathcal{D}_n} \{0,1\}^n$. We say that f is* one-way *if f is one-way on the uniform distribution $\mathcal{U}$.*

If a function is one-way then it is somewhat one-way.

Proposition 1 *If there is a polynomial-samplable $\mathcal{D}$ such that f is one-way on $\mathcal{D}$ then there is a one-way function g.*

The function g is simply the composition of f and the polynomial time computable sampling function for $\mathcal{D}$. This proposition allows us to state most of our results in terms of one-way functions on the uniform ensemble as opposed to other probability ensembles.

DEFINITION 3.7 (copies of functions and ensembles) *Let $k(n)$ be a length function. Let $\mathcal{D}^k$ be the ensemble with length $nk(n)$ such that $\mathcal{D}^k_n$ is the distribution obtained by independently sampling $k(n)$ times from $\mathcal{D}_n$ and concatenating the results. Similarly, let f^k be the function with input and output lengths $nk(n)$ and $l(n)k(n)$, respectively, given by:*

$$f^k(x_1 \circ \ldots \circ x_{k(n)}) = f(x_1) \circ \ldots \circ f(x_{k(n)}),$$

where $x_1, \ldots, x_{k(n)} \in \{0,1\}^n$.

The following proposition is implicitly used in [35, Yao]. The importance of Proposition 2 is that it allows us to transform any function that is just a little bit hard to invert into one that is hard to invert almost always for a random input. For the remainder of the paper, we only consider one-way functions as opposed to somewhat one-way functions.

Proposition 2 *If f is somewhat one-way on $\mathcal{D}$ with associated constant $c > 0$, then f^k is one-way on $\mathcal{D}^k$, where $k(n) = n^{c+1}$.*

4 Pseudo-random generator

Informally, a polynomial time computable function f is *pseudo-random* if $f(X)$ is strictly longer than X and if every feasible adversary can distinguish $f(X)$ from a truly random string of the same length with only negligible probability. Intuitively, $f(X)$ "looks" just like a random string to any feasible adversary, even though it is generated from a string X that is strictly shorter. This intuition is captured in the definition of a pseudo-random generator [5, Blum Micali], [35, Yao]. Before giving the definition of a pseudo-random generator, we state some related useful definitions.

DEFINITION 4.1 (statistically indistinguishable and quasi-random) *$\mathcal{D}_n$ and $\mathcal{E}_n$ are* statistically indistinguishable within *δ if $\sum_{x\in\{0,1\}^n} |\mathcal{D}_n[\{x\}] - \mathcal{E}_n[\{x\}]| < \delta$. $\mathcal{D}_n$ is* quasi-random within *δ if $\mathcal{D}_n$ is statistically indistinguishable within δ from the uniform distribution.*

Proposition 3 *Let $\mathcal{D}_n$ and $\mathcal{E}_n$ be probability distributions on $\{0,1\}^n$ which are statistically indistinguishable within δ. Let f be any function on inputs of length n. Then, $f(\mathcal{D}_n)$ and $f(\mathcal{E}_n)$ are statistically indistinguishable within δ.*

The following definition is the computational analog of statistical indistinguishable. The definition of computationally indistinguishable appears in [11, Goldwasser Micali], [35, Yao] and [12, Goldwasser Micali Rackoff].

DEFINITION 4.2 (computationally indistinguishable) *The* distinguishing probability function *$p(n)$ of adversary A for $\mathcal{D}$ and $\mathcal{E}$ is $|\Pr[A(X) = 1] - \Pr[A(X') = 1]|$ when $X \in_{\mathcal{D}_n} \{0,1\}^n$ and $X' \in_{\mathcal{E}_n} \{0,1\}^n$. $\mathcal{D}$ is* computationally indistinguishable *from $\mathcal{E}$ if every feasible adversary has negligible distinguishing probability.*

The following proposition is the computational analog of Proposition 3.

Proposition 4 *Let $\mathcal{D}_n$ and $\mathcal{E}_n$ be probability distributions on $\{0,1\}^n$ which are computationally indistinguishable and let f be any function on inputs of length n that can be computed in feasible time. Then, $f(\mathcal{D}_n)$ and $f(\mathcal{E}_n)$ are computationally indistinguishable.*

[35, Yao] makes the following definition of pseudo-random.

DEFINITION 4.3 (pseudo-random generator) *Let g be a function such that the output length of g is strictly longer than its input length, i.e. $l(n) > n$ for all $n \in N$. We say that g is a* pseudo-random generator *if $g(\mathcal{U}_n)$ is computationally indistinguishable from $\mathcal{U}_{l(n)}$.*

The usual definition of a pseudo-random generator insists that the generator can stretch the input by any polynomial amount. The following shows the definition above is equivalent, and follows from [8, Goldreich Goldwasser Micali].

Proposition 5 *Suppose that g is a pseudo-random generator that stretches an n bit string to an $n+1$ bit string. Let $g_0(x) = g(x)$ and define inductively, for all $i \geq 1$, $g_i(x) = g(g_{i-1}(x)_{\leftarrow n}) \circ g_{i-1}(x)_{n\rightarrow}$. Then, for every length function $l(n)$, $g_{l(n)}$ is a pseudo-random generator.*

Some of our constructions have the property that for each value of n we construct a polynomial size family of generators and the guarantee is that at least one of them is pseudo-random, but we don't know which one. The following lemma provides a way to construct a pseudo-random generator from such a family by taking the "exclusive-or" of the outputs of the generators.

Lemma 6 *Let $l(n)$ and $k(n)$ be length functions such that $l(n) > nk(n)$. Let $t(n)$ be a function that is polynomial in n. Let*

$$\{g_{i,n} : \{0,1\}^n \rightarrow \{0,1\}^{l(n)} : i = 1, \ldots, k(n); n \in \mathcal{N}\}$$

be a family of functions such that for each $n \in \mathcal{N}$, for each $i = 1, \ldots, k(n)$ and for all $x \in \{0,1\}^n$, $g_{i,n}(x)$ is computable in time $t(n)$. Suppose that for each value of n there is a value $s(n) \in \{1, \ldots, k(n)\}$ such that the subfamily of functions $\{g_{s(n),n} : n \in \mathcal{N}\}$ is pseudo-random. Define generator

$$g_n(X_1 \circ \cdots \circ X_{k(n)}) = \oplus_{i=1}^{k(n)} g_{i,n}(X_i),$$

where $X_1 \in_{\mathcal{U}} \{0,1\}^n, \ldots, X_{k(n)} \in_{\mathcal{U}} \{0,1\}^n$. Then, the family $\{g_n : n \in \mathcal{N}\}$ is a pseudo-random generator.

PROOF: It is not hard to verify that g on inputs of length $nk(n)$ is computable in time $O(t(n)k(n))$. Since the length of the output of g on inputs of length $nk(n)$ is $l(n) > nk(n)$ by assumption, it remains to be shown that the output of g is indistinguishable from a truly random string of the same length.

Let $R \in_{\mathcal{U}} \{0,1\}^{l(n)}$. Suppose there is an adversary A that distinguishes $S = g_n(X_1 \circ \cdots \circ X_{k(n)})$ from R with non-negligible probability $p(n)$, i.e. A accepts S with probability p_S, A accepts R with probability p_R and $|p_S - p_R| \geq p(n)$. We show this implies that for all $i = 1, \ldots, k(n)$ there is an adversary M_i^A that distinguishes $g_{i,n}(X_i)$ from R with probability at least $p(n)$, and this contradicts the assumption that the family $\{g_{s(n),n} : n \in \mathcal{N}\}$ is pseudo-random.

For each $i = 1, \ldots, k(n)$ let M_i^A be the oracle Turing Machine that works as follows. On input Y, M_i^A generates $X_1 \in_{\mathcal{U}} \{0,1\}^n, \ldots, X_{i-1} \in_{\mathcal{U}} \{0,1\}^n$ and $X_{i+1} \in_{\mathcal{U}} \{0,1\}^n, \ldots, X_{k(n)} \in_{\mathcal{U}} \{0,1\}^n$ and computes $S' = \oplus_{j \neq i} g_{j,n}(X_j) \oplus Y$. Then M_i^A runs A on input S' and accepts if A accepts. By the nature of $\oplus$, if $Y = R$ then M_i^A accepts with probability p_R, whereas if $Y = g_{i,n}(X_i)$ then M_i^A accepts with probability p_S. Thus, for each value of i, M_i^A distinguishes $g_{i,n}(X_i)$ from R with probability $|p_S - p_R| \geq p(n)$. ■

If the original family in Lemma 6 does not satisfy the property that $l(n) > nk(n)$, then we can use Proposition 5 to stretch the output of each generator in the family and then apply Lemma 6.

In the remainder of the paper, we show how to construct a pseudo-random generator from any one-way function. We provide several constructions, depending on the structural properties of the one-way function, starting with the simpler constructions for one-way functions with a lot of structure and finishing with the most complicated construction for one-way functions with no required structural properties.

5 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 log problem [5, Blum Micali] and has been central to all such constructions since that time.

DEFINITION 5.1 (hidden and meaningful bit) *Let f' be a function of the form $f'(x) = f(x) \circ b(x)$ where $x \in \{0,1\}^n$ and both f and b are polynomial time computable and $b(x)$ is a single bit. The* prediction probability *$p(n)$ of adversary A for b given f on $\mathcal{D}$ is* $|\Pr[A(f(X)) = b(X)] -$

$\Pr[A(f(X) \neq b(X)]|$ *when* $X \in_{\mathcal{D}_n} \{0,1\}^n$. *Bit* b *is* hidden *for* f' *on* $\mathcal{D}$ *if every feasible adversary has negligible prediction probability for* b *given* f *on* $\mathcal{D}$. *Bit* b *is* hidden *for* f' *if it is hidden for* f' *on* $\mathcal{U}$. *Bit* b *is* $p(n)$-meaningful *for* f' *on* $\mathcal{D}$ *if there is an unbounded adversary* A *with prediction probability at least* $p(n)$. *Bit* b *is* meaningful *for* f' *on* $\mathcal{D}$ *if there is a constant* $c > 0$ *such that* b *is* $\frac{1}{n^c}$ *meaningful for* f' *on* $\mathcal{D}$. *Bit* b *is* meaningful *for* f' *if it is meaningful for* f' *on* $\mathcal{U}$.

The following proposition shows that we need only functions with the uniform distribution on inputs in the definition of a function with a hidden and meaningful bit.

Proposition 7 *Let* $f'(X) = f(X) \circ b(X)$ *where* b *is a hidden and meaningful bit of* f' *when* $X \in_{\mathcal{D}_n} \{0,1\}^n$, *where* $\mathcal{D}$ *is polynomially samplable. Let* M *be the polynomial time sampling algorithm for* $\mathcal{D}$ *and suppose without loss of generality that* M *takes* n *uniformly distributed bits to sample from* $\mathcal{D}_n$. *Let* $Y \in_{\mathcal{U}} \{0,1\}^n$ *and let* $c(Y) = b(M(Y))$, *let* $g(Y) = f(M(Y))$ *and let* $g'(Y) = g(Y) \circ c(Y)$. *Then* c *is a hidden and and meaningful bit for* g'.

How do we go about constructing a function that hides a meaningful bit from a one-way function? A key component of this construction comes from [10, Goldreich Levin].

DEFINITION 5.2 (inner product bit) *Let* $X \in_{\mathcal{D}_n} \{0,1\}^n$ *and* $R \in_{\mathcal{U}} \{0,1\}^n$. *Let* $\hat{f}(X \circ R) = f(X) \circ R$ *and let* $\mathcal{D}'_n = \mathcal{D}_n \circ \mathcal{U}_n$. *It is easy to see that if* f *is one-way on* $\mathcal{D}$ *then* $\hat{f}$ *is one-way on* $\mathcal{D}'$. *The* inner product bit *is* $b(X \circ R) = X \odot R$.

The following important proposition is from [10, Goldreich Levin].

Proposition 8 *Assume that* f *is one-way on* $\mathcal{D}$ *and let* $X \in_{\mathcal{D}_n} \{0,1\}^n$ *and let* $R \in_{\mathcal{U}} \{0,1\}^n$. *Let* $\hat{f}(X \circ R) = f(X) \circ R$. *Let* $\mathcal{D}'_n = \mathcal{D}_n \circ \mathcal{U}_n$ *and let* $b(X \circ R) = X \odot R$ *be the inner product bit. Let* $g(X \circ R) = \hat{f}(X \circ R) \circ b(X \circ R)$. *Then,* b *is hidden for* g *on* $\mathcal{D}'$.

In fact, what [10, Goldreich Levin] proves is something much stronger, which is stated in the following proposition.

Proposition 9 *Let* A *be a Turing machine that outputs a single bit on inputs of the form* $y \circ r$, *where* $r \in \{0,1\}^n$. *Let* $R \in_{\mathcal{U}} \{0,1\}^n$. *There is an oracle Turing Machine* M^A *which on input* y *and error parameter* $\delta > 0$ *outputs a list* L *of* n*-bit strings such that with probability* $1 - \exp(-n)$ *the following statement is true for all* $x \in \{0,1\}^n$: *If* $|\Pr[A(y \circ R) = x \odot R] - \Pr[A(y \circ R) \neq x \odot R]| \geq \delta$ *then* $x \in L$. *The running time of* M^A *is polynomial in* $\frac{1}{\delta}$ *and the running time of* A.

PROOF: (of Proposition 8) Suppose there is a feasible adversary A with non-negligible prediction probability $p(n)$ for $X \odot R$ given $f(X) \circ R$. When $x \in_{\mathcal{D}_n} \{0,1\}^n$ is fixed and y is set to $f(x)$ then with probability at least $\frac{p(n)}{2}$ the prediction probability of A for $x \odot R$ on input $y \circ R$ is at least $\frac{p(n)}{2}$. Consider the oracle Turing machine M^A described in Proposition 9. With probability at least $\frac{p(n)}{2}$, M^A on input $Y = f(X)$ and $\frac{p(n)}{2}$ outputs a list L of n-bit strings that with probability $1 - \exp(-n)$ contains X. Let N^{M^A} be the oracle Turing machine that on input y, runs M^A on input y to produce a list L, and then for each $x' \in L$ outputs x' if $f(x') = y$. With probability at least $\frac{p(n)}{2}$ an inverse of $f(X)$ is produced by N^{M^A}. ■

Proposition 8 presents an elegant, simple and general method of obtaining a hidden bit from a one-way function. We need the stronger Proposition 9 in some of our proofs.

Proposition 10 *Let f' be of the form $f'(X) = f(X) \circ b(X)$ where b is a bit function and $X \in_{\mathcal{U}} \{0,1\}^n$. Let probability ensembles $\mathcal{D}$ and $\mathcal{E}$ be defined as follows. Let $\mathcal{D}_n = f(X) \circ b(X)$ and let $\mathcal{E}_n = f(X) \circ \beta$ where $\beta \in_{\mathcal{U}} \{0,1\}$ is independent of X. If b is a hidden bit of f' then $\mathcal{D}$ and $\mathcal{E}$ are computationally indistinguishable.*

PROOF: Since both f and b are polynomial time computable functions, both $\mathcal{D}$ and $\mathcal{E}$ are polynomially samplable. We show below that there is an oracle Turing machine M such that if A is a feasible adversary that has non-negligible distinguishing probability $p(n)$ for $\mathcal{D}$ and $\mathcal{E}$, with $p(n) = \Pr[A(f(X) \circ b(X)) = 1] - \Pr[A(f(X) \circ \beta) = 1]$, then M^A is a feasible adversary with prediction probability $2p(n)$ for $b(X)$ given $f(X)$. This contradicts our assumption that b is a hidden bit of f' and thus $\mathcal{D}$ and $\mathcal{E}$ are computationally indistinguishable.

M^A follows a fairly standard outline common in the literature.

Description of $M^A(f(X))$

$B_0 \leftarrow A(f(X) \circ 0)$
$B_1 \leftarrow A(f(X) \circ 1)$
If $B_0 = B_1$ then output $\alpha \in_{\mathcal{U}} \{0,1\}$
Elseif $B_0 = 1$ then output 0
Elseif $B_1 = 1$ then output 1

It can be easily verified that $\Pr[M^A(f(X)) = b(X)] - 1/2 = p(n)$ and thus the prediction probability $\Pr[M^A(f(X)) = b(X)] - \Pr[M^A(f(X)) \neq b(X)]$ is equal to $2p(n)$. ■

Corollary 11 *Let probability ensembles $\mathcal{D}$ and $\mathcal{E}$ be defined as $\mathcal{D}_n = f(X) \circ R \circ (X \odot R)$ and $\mathcal{E}_n = f(X) \circ R \circ \beta$ where $X \in_{\mathcal{U}} \{0,1\}^n$, $R \in_{\mathcal{U}} \{0,1\}^n$ and $\beta \in_{\mathcal{U}} \{0,1\}$ are independently distributed. If f is a one-way function then $\mathcal{D}$ and $\mathcal{E}$ are computationally indistinguishable.*

PROOF: By Proposition 10 there is an oracle machine M such that if A is a feasible adversary that has non-negligible distinguishing probability for $\mathcal{D}$ and $\mathcal{E}$ then M^A is a feasible adversary that has non-negligible prediction probability for $X \odot R$ given $f(X) \circ R$. By Proposition 8, there is an oracle machine N such that if M^A is a feasible adversary that has non-negligible prediction probability for $X \odot R$ given $f(X) \circ R$ then N^{M^A} is a feasible adversary that has non-negligible inverting probability for f. ■

5.1 One-way permutation → pseudo-random generator

In this subsection, we describe a way to construct a pseudo-random generator from any one-way permutation which is substantially simpler than the original construction of [35, Yao]. The construction and proof described here is due to [10, Goldreich Levin].

Theorem 1 *If f is a one-way permutation then $g(X \circ R) = f(X) \circ R \circ (X \odot R)$ is a pseudo-random generator, where $X \in_{\mathcal{U}} \{0,1\}^n$ and $R \in_{\mathcal{U}} \{0,1\}^n$.*

PROOF: The generator g is the same as the g described in Proposition 8 (page 11). Let $\mathcal{D}$ and $\mathcal{E}$ be defined as in Corollary 11 (page 12). Because f is a permutation, $\mathcal{E}_n = f(X) \circ R \circ \beta$ is the uniform

distribution on strings of length $2n+1$. Since $\mathcal{D}$ and $\mathcal{E}$ are computationally indistinguishable, g is a pseudo-random generator. ■

The reason for the simplicity in the construction when f is a permutation (versus the case when f is not necessarily a permutation) is twofold: (1) $f(X)$ is uniformly distributed in $\{0,1\}^n$; (2) $f(X)$ uniquely determines X, and thus from $f(X)$ and R it is informationally possible to compute $R \odot X$, and consequently $R \odot X$ is a meaningful bit of $g(X \circ R)$. The first property ensures that the first $2n$ output bits of the generator are uniform and random, and the second property ensures that $R \odot X$ is a meaningful bit of $g(X \circ R)$ (Proposition 8 ensures that it is hidden). For a general one-way function, neither of these properties necessarily holds. In the sequel, we first develop the construction of a pseudo-random generator from any function that hides a meaningful bit, and then later we show how to construct a function that hides a meaningful bit from any one-way function.

6 Manipulating Entropy

Our constructions of pseudo-random generators from one-way functions can be viewed as manipulations of entropy. In this section, we provide the necessary definitions and technical tools for these manipulations.

The following definition of entropy is from [30, Shannon].

DEFINITION 6.1 (Shannon entropy) *For each $x \in \{0,1\}^n$, define the entropy of x with respect to $\mathcal{D}_n$ to be* $\mathrm{Ent}(x) = -\log(\mathcal{D}_n[\{x\}])$. *The* (Shannon) entropy, $\mathrm{Ent}(\mathcal{D}_n)$, *of $\mathcal{D}_n$ is* $\mathrm{Exp}[\mathrm{Ent}(X)]$ *when $X \in_{\mathcal{D}_n} \{0,1\}^n$. The* entropy function $\mathrm{Ent}(\mathcal{D})$ *assigns to each $n \in N$ the value* $\mathrm{Ent}(\mathcal{D}_n)$. *For a function f, we call* $\mathrm{Ent}(f(\mathcal{D}))$ *the* (Shannon) entropy of f on $\mathcal{D}$ *and* $\mathrm{Ent}(\mathcal{D}) - \mathrm{Ent}(f(\mathcal{D}))$ *the* degeneracy *of f on $\mathcal{D}$.*

DEFINITION 6.2 (computational entropy) *Let $s(n)$ be a function from $\mathcal{N}$ to the positive reals. We say f has* computational entropy at least $s(n)$ *if there is a polynomially samplable ensemble $\mathcal{E}$ such that $\mathcal{E}$ is computationally indistinguishable from $f(\mathcal{U})$ and* $\mathrm{Ent}(\mathcal{E}_n) \geq s(n)$.

The following proposition appears in [12, Goldwasser Micali Rackoff].

Proposition 12 *Let $k(n)$ be a length function. If $\mathcal{D}$ and $\mathcal{E}$ are polynomially samplable probability ensembles that are computationally indistinguishable then $\mathcal{D}^k$ and $\mathcal{E}^k$ are computationally indistinguishable. More precisely, there is a probabilistic oracle Turing machine M such that if A is a feasible adversary with non-negligible distinguishing probability for $\mathcal{D}^k$ and $\mathcal{E}^k$ then M^A is a feasible adversary with non-negligible distinguishing probability for $\mathcal{D}$ and $\mathcal{E}$.*

In Proposition 12 it is crucial that both probability ensembles are polynomially samplable because the sampling algorithms are incorporated into M. Because we want to be able to *uniformly* convert a successful adversary for distinguishing $\mathcal{D}^k$ from $\mathcal{E}^k$ into a successful adversary for distinguishing $\mathcal{D}$ from $\mathcal{E}$, we required $\mathcal{E}$ to be polynomially samplable in the definition of computational entropy. There is a weaker form of Proposition 12 where neither $\mathcal{D}$ nor $\mathcal{E}$ are required to be polynomially samplable, but then the much weaker conclusion of the proposition is that there is a non-uniform way of converting a feasible adversary for distinguishing $\mathcal{D}^k$ and $\mathcal{E}^k$ into a feasible adversary for distinguishing $\mathcal{D}$ and $\mathcal{E}$.

6.1 Smoothing Distributions with Hashing

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], [31, Santha Vazirani], [32, Vazirani], [34, Vazirani Vazirani], [7, Chor Goldreich] [24, McInnes].)

Lemma 13, which we introduce and prove in this subsection, is very useful in many of our constructions of various kinds of one-way functions and pseudo-random generators. However, it is probably best thought of as a result in the theory of slightly-random sources, instead of pseudo-random generators. Intuitively, it can be thought of as a method for extracting "uniform" random bits from a slightly-random source using real random bits as a "catalyst".

Before stating and proving Lemma 13, we start with some preliminaries. We need to use a variant definition of entropy used in [7, Chor Goldreich].

DEFINITION 6.3 (min-entropy) *The* min-entropy *of* $\mathcal{D}_n$ *is* $\min_{x \in \{0,1\}^n}\{-\log(\mathcal{D}_n[\{x\}])\}$.

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 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 6.4 (hash functions) *Let* $\mathcal{H}_{n,m}$ *be a family of functions from* n *bit strings to* m *bit strings. We say* $\mathcal{H}_{n,m}$ *is a family of* pairwise independent universal hash functions *if, for all* $x, y \in \{0,1\}^n$, $x \neq y$, $H(x) \circ H(y) \in_{\mathcal{U}} \{0,1\}^{2m}$ *when* $H \in_{\mathcal{U}} \mathcal{H}_{n,m}$. *A* system of hash functions *consists of one such family for all pairs* n *and* m.

We require that the number of hash functions in $\mathcal{H}_{n,m}$ be a power of two, say $2^{l(n,m)}$. Furthermore, we require that there is an easily computable one-to-one mapping from bit strings of length $l(n,m)$ into $\mathcal{H}_{n,m}$. Slightly abusing notation, when we write the name of the hash function this denotes the bit string describing the hash function, and when we write the name of the hash function applied to a parameter this denote the bit string that is the value of the hash function applied to the parameter, e.g. h denotes the description of a hash function and $h(x)$ denotes the value of the hash function h applied to x. Thus, the description of $H \in_{\mathcal{U}} \mathcal{H}_{n,m}$ can be thought of as $H \in_{\mathcal{U}} \{0,1\}^{l(n,m)}$.

The following system of pairwise independent universal hash functions has several nice properties. Let $\mathcal{H}_{n,m}$ be the set of all m by $n+1$ matrices over the field with two elements. A description of a hash function from this system is $h = (M, b)$, where M is an m by n bit matrix and b is a bit vector of length m. Then, $h(x) = (M \cdot x) \oplus b$. The description of $H \in_{\mathcal{U}} \mathcal{H}_{n,m}$ can be thought of as $H \in_{\mathcal{U}} \{0,1\}^{(n+1)m}$.

Another interesting set of pairwise independent hash functions can be obtained as follows. Suppose we are given a representation of $GF[2^n]$ and two numbers a and b in this field. Then if

we interpret $x \in \{0,1\}^n$ as an element of this field and let $h(x) = ax + b$ we obtain a hash function from n-bit strings to n-bit strings. If $m \leq n$ we can obtain a hash function to m-bit strings by dropping part of the output while if $m > n$ we need several independent hash functions. The advantage of this scheme over the first is that it uses fewer bits, but on the other hand we need a representation of the finite field in question.

Hereafter, whenever we refer to a family or system of hash functions, we mean one of the families defined here. However, any system of hash functions that satisfy the required properties may be used.

Lemma 13 can be interpreted as follows. Suppose we have a distribution $\mathcal{D}_n$ on strings of length n with min-entropy greater than m. A fair coin is used to generate a random hash function H mapping n bits to $m - 2e$ bits, where e is a small integer that controls the tradeoff between the "uniformity" of the output bits and the amount of entropy lost in the smoothing process. We then sample from $\mathcal{D}_n$ and apply H to the result. Lemma 13 states that the resulting bits are essentially uniformly and randomly distributed and almost uncorrelated with the bits used to generate H. Thus, we have managed to convert almost all the min-entropy of $\mathcal{D}_n$ into uniform random bits while maintaining our original supply of uniform random bits. Previously, [24, McInnes] proved a related lemma, and independently, [2, Bennett Brassard Robert] proved a similar lemma.

The following definition of entropy, due to [28, Renyi], is intermediate between Shannon entropy and min-entropy.

DEFINITION 6.5 (Renyi entropy) *The* Renyi entropy *of distribution $\mathcal{D}_n$ is equal to*

$$-\log\left(\sum_{x\in\{0,1\}^n} \mathcal{D}_n[\{x\}]^2\right).$$

It can be easily verified that the Renyi entropy of a distribution is at most the Shannon entropy and at least the min-entropy. In the following lemma, the distribution is required to have a certain amount of Renyi entropy. In the remainder of the paper, this lemma is applied to distributions which have min-entropy at least the amount of Renyi entropy required by the lemma, and because the Renyi entropy of a distribution is at least the min-entropy the lemma can be directly applied.

Some generalizations of Lemma 13 are possible, including weaker restrictions on the hash functions used. These generalizations can be used to make our constructions somewhat more efficient.

Lemma 13 *Let $\mathcal{D}_n$ be a distribution that has Renyi entropy at least m and let $l = m - 2e$. Then the distribution $H \circ H(X)$, where $H \in_{\mathcal{U}} \mathcal{H}_{n,l}$ and $X \in_{\mathcal{D}_n} \{0,1\}^n$, is quasi-random within 2^{-e}.*

PROOF: For $s \in \{0,1\}^l$ and $h \in \mathcal{H}_{n,l}$, let $W_s(h)$ be the probability that $h(X) = s$. By the properties of hash functions (see the remarks following the definition of hash functions on page 14), for each $h \in \mathcal{H}_{n,l}$, $\Pr[H = h] = 2^{-|h|}$. The quantity we need to bound is

$$\sum_{s\in\{0,1\}^l} \sum_{h\in\mathcal{H}_{n,l}} 2^{-|h|} |W_s(h) - 2^{-l}|. \tag{1}$$

By the properties of hash functions, for each fixed $s \in \{0,1\}^l$, $\mathrm{Exp}[W_s(H)] = 2^{-l}$. We can rewrite equation (1) as

$$\sum_{s \in \{0,1\}^l} \mathrm{Exp}[|W_s(H) - 2^{-l}|]. \tag{2}$$

For any random variable Y, $\mathrm{Exp}[|Y|] \leq \mathrm{Exp}[Y^2]^{1/2}$, and thus equation (2) is upper bounded by

$$\sum_{s \in \{0,1\}^l} \mathrm{Exp}[(W_s(H) - 2^{-l})^2]^{1/2}. \tag{3}$$

Below we show that $\mathrm{Exp}[(W_s(H) - 2^{-l})^2] \leq 2^{-2l-2e}$. From this we conclude that equation (3) is upper bounded by $2^l \cdot 2^{-l-e} \leq 2^{-e}$.

We now show that $\mathrm{Exp}[(W_s(H) - 2^{-l})^2] \leq 2^{-2l-2e}$. For each $h \in \mathcal{H}_{n,l}$ we can rewrite

$$W_s(h) = \sum_{x \in \{0,1\}^n} \mathcal{D}_n[\{x\}]\chi(h(x) = s),$$

where $\chi(h(x) = s)$ takes the value 1 if $h(x) = s$ and is 0 otherwise. We can rewrite $\mathrm{Exp}[(W_s(H) - 2^{-l})^2]$ as

$$\mathrm{Exp}\left[\sum_{x,y} \mathcal{D}_n[\{x\}]\mathcal{D}_n[\{y\}]\left(\chi(H(x) = s) - 2^{-l}\right)\left(\chi(H(y) = s) - 2^{-l}\right)\right]. \tag{4}$$

Since the events $H(x) = s$ and $H(y) = s$ are independent for fixed x and y, $x \neq y$, all terms except the diagonal terms are zero when we move the expected value inside the sum. Using this and the fact that $\Pr[\chi(H(x) = s) = 1] = 2^{-l}$, equation (4) can be rewritten as

$$\sum_{x \in \{0,1\}^n} {\mathcal{D}_n}^2[\{x\}]\left(2^{-l}(1 - 2^{-l})^2 + (1 - 2^{-l})(2^{-l})^2\right) \leq 2^{-l} \sum_{x \in \{0,1\}^n} {\mathcal{D}_n}^2[\{x\}]. \tag{5}$$

Because the Renyi entropy of $\mathcal{D}_n$ is at least m, $\sum_{x \in \{0,1\}^n} {\mathcal{D}_n}^2[\{x\}] \leq 2^{-m}$. Thus, equation (5) is upper bounded by $2^{-l-m} = 2^{-2l-2e}$. ■

6.2 Converting arbitrary entropy into uniform entropy

Lemma 13 is very 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 13 says that the min-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 min-entropy in Lemma 13 by the weaker and more natural 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 problem 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.

In a little more detail, we first prove that for sufficiently large $k(n)$ there is a distribution $\mathcal{E}_n$ on strings of length $nk(n)$ that is statistically indistinguishable within $\exp(-n)$ from $\mathcal{D}_n^{k(n)}$

(the concatenation of $k(n)$ independent samples of $\mathcal{D}_n$) such that the min-entropy of $\mathcal{E}_n$ is approximately equal to $\mathrm{Ent}(\mathcal{D}_n^{k(n)})$. Lemma 13 shows that if H is a randomly chosen hash function from $nk(n)$ bits to slightly less than the min-entropy of $\mathcal{E}_n$ bits and $X \in_{\mathcal{E}_n} \{0,1\}^{nk(n)}$, then $H \circ H(X)$ is statistically indistinguishable within $\exp(-n)$ from the uniform distribution. Since $\mathcal{E}_n$ and $\mathcal{D}_n^{k(n)}$ are statistically indistinguishable within $\exp(-n)$, we conclude using Proposition 3 (page 9) that if $X_1, \ldots, X_{k(n)}$ are independently chosen according to $\mathcal{D}_n$ then $H \circ H(X_1 \circ \cdots \circ X_{k(n)})$ is statistically indistinguishable within $\exp(-n)$ from the uniform distribution on approximately $|H| + k(n)\mathrm{Ent}(\mathcal{D}_n)$ bits.

Lemma 14 *Let $k(n)$ be a length function. For every probability ensemble $\mathcal{D}$ there is a probability ensemble $\mathcal{E}$ with length function $nk(n)$ satisfying:*

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

PROOF: Let $S \subseteq \{0,1\}^n$ be the set of elements with probability at least 2^{-2n} with respect to $\mathcal{D}_n$, and let $S' = \{0,1\}^n - S$. Let $\mathcal{D'}_n$ be the distribution on $\{0,1\}^n$ described as: (1) for all $x \in S$, $\mathcal{D'}_n[\{x\}] = \mathcal{D}_n[\{x\}]/\mathcal{D}_n[S]$; (2) for all $x \in S'$, $\mathcal{D'}_n[\{x\}] = 0$. It is easy to show that $\mathcal{D}_n[S'] \leq 2^{-n}$, and from this it follows that $\mathrm{Ent}(\mathcal{D'}_n) \geq \mathrm{Ent}(\mathcal{D}_n) - 2^{-n}$ and that $\mathcal{D'}_n$ and $\mathcal{D}_n$ are statistically indistinguishable within 2^{-n}.

For each $x \in \{0,1\}^n$, let $\mathrm{Ent}(x) = -\log(\mathcal{D'}_n[\{x\}])$. When $X_i \in_{\mathcal{D'}_n} \{0,1\}^n$ independently for $i = 1, \ldots, k(n)$, $Z = \sum_{i=1,\ldots,k(n)} \mathrm{Ent}(X_i)$ is the sum of independent random variables on the interval $[0, 2n]$ with expected value $\mathrm{Ent}(\mathcal{D'}_n)$. Hence, by an elementary extension of Chernoff bounds, with probability $1 - \exp(-k(n))$, Z has value within an additive factor of $nk(n)^{2/3}$ of its expectation. Thus, with probability $1 - \exp(-k(n))$, $Z > k(n)\mathrm{Ent}(\mathcal{D'}_n) - nk(n)^{2/3}$. This means that only with probability $\exp(-k(n))$, the sequence $x_1, \ldots, x_{k(n)}$ has probability greater than $2^{-k(n)\mathrm{Ent}(\mathcal{D}'_n)+nk(n)^{2/3}}$ when x_i is chosen independently according to X_i for all i, Restricting $\mathcal{D'}^{k(n)}$ to the complement of this exponentially small in probability set of sequences, and renormalizing the distribution as before, we obtain $\mathcal{E}_n$. ■

Corollary 15 *Let $k(n) = n^c$ for any constant $c > 0$. Let $H \in_{\mathcal{U}} \mathcal{H}_{nk(n),k(n)\mathrm{Ent}(\mathcal{D}_n)-2nk(n)^{2/3}}$, and let $X_i \in_{\mathcal{D}_n} \{0,1\}^n$ independently for $i = 1, \ldots, k(n)$. Then the distribution $H \circ H(X_1 \circ \ldots \circ X_{k(n)})$ is quasi-random within $\exp(-n)$.*

PROOF: Use Lemma 14 (page 17), Lemma 13 (page 15) and Proposition 3 (page 9). ■

7 Pseudo-entropy generator

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 7.1 (pseudo-entropy generator) *We say that f is a* pseudo-entropy generator *if f has computational entropy at least $s(n)$ where $s(n) \geq n + 1/n^c$ for some constant c, where n is the input length function for f.*

Note that a pseudo-random generator is a pseudo-entropy generator because the uniform distribution is polynomially samplable and because the output length $l(n)$ of a pseudo-random generator is greater than its input length n, and consequently $\mathrm{Ent}(\mathcal{U}_{l(n)}) = l(n) \geq n + 1$.

7.1 Pseudo-entropy generator → pseudo-random generator

Proposition 12 (page 13), together with Lemma 13 (page 15) via Corollary 15 (page 17), 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. This is exactly what we do in the following theorem.

Theorem 2 *Let f be a pseudo-entropy generator with computational entropy at least $s(n) = n + 1/n^c$, let $k = n^{3c+4}$ and let $j = ks(n) - 2nk^{2/3}$. Then, $g(H \circ X_1 \circ \ldots \circ X_k) = H \circ H(f(X_1) \circ \ldots \circ f(X_k))$ is a pseudo-random generator, where $H \in_{\mathcal{U}} \mathcal{H}_{kl(n),j}$ and, for all $i = 1, \ldots, k$, $X_i \in_{\mathcal{U}} \{0,1\}^n$ are independent.*

PROOF: Let $\mathcal{D}$ be the polynomial samplable probability ensemble with $\mathrm{Ent}(\mathcal{D}_n) \geq s(n)$ that is computationally indistinguishable from $f(X)$, where $X \in_{\mathcal{U}} \{0,1\}^n$. By Proposition 12 (page 13), $f^k(X_1 \circ \ldots \circ X_k) = f(X_1) \circ \ldots \circ f(X_k$ is computationally indistinguishable from $\mathcal{D}^k$. Note that $\mathrm{Ent}(\mathcal{D}^k_n) = k\mathrm{Ent}(\mathcal{D}_n) \geq ks(n)$. Let $\mathcal{D}'_n$ be the probability distribution defined by $H \circ H(Y_1 \circ \ldots \circ Y_k)$, where, for all $i = 1, \ldots, k$, $Y_i \in_{\mathcal{D}_n} \{0,1\}^{l(n)}$ are independent. By Corollary 15 (page 17), $\mathcal{D}'_n$ is quasi-random within $\exp(-n)$. Let $\mathcal{E}'_n$ be the probability distribution defined by the output of g. Then, since f^k is computationally indistinguishable from $\mathcal{D}^k$, it follows from Proposition 4 (page 9) that $\mathcal{E}'$ is computationally indistinguishable from $\mathcal{D}'$. This is true in the sense that there is an oracle Turing machine M such that if A is a feasible adversary that distinguishes $\mathcal{D}'$ and $\mathcal{E}'$ with non-negligible probability then M^A is a feasible adversary that distinguishes $\mathcal{D}$ from $\mathcal{E}$ with non-negligible probability. The uniform reduction is possible because both $\mathcal{D}$ and $\mathcal{E}$ are polynomially samplable and the sampling algorithms are incorporated into M. Because $\mathcal{D}'$ is quasi-random within $\exp(-n)$, and because by choice of k the output of g is longer than the input, g is a pseudo-random generator. ■

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

In this subsection we describe a construction of a pseudo-entropy generator from any one-way one-to-one function. This construction, together with Theorem 2 (page 18), yields a pseudo-random generator from any one-way one-to-one function. The overall construction is quite different in spirit than the original construction of [9, Goldreich Krawczyk Luby] that yields the same result. We present this construction because it illustrates how to construct a pseudo-entropy generator in a particularly simple way using [10, Goldreich Levin].

The construction is the same as in Proposition 8 (page 11). If we use this same construction applied to a one-way one-to-one function, then it is not possible to argue that the result is a

pseudo-random generator.[4] On the other hand, all is not lost, because it is still the case that $f(X)$ is uniformly distributed on some set of 2^n bit strings, and it is still the case that $R \odot X$ is both hidden and meaningful. We show that this construction yields a pseudo-entropy generator.

Lemma 16 *If f is a one-way one-to-one function then g is a pseudo-entropy generator, where $g(X \circ R) = f(X) \circ R \circ (X \odot R)$.*

PROOF: The generator g is the same as the g described in Proposition 8 (page 11). Consider the probability ensembles $\mathcal{D}$ and $\mathcal{E}$ defined in Corollary 11 (page 12). Corollary 11 shows that $\mathcal{E}$ and $\mathcal{D}$ are computationally indistinguishable. On the other hand, because β is an independent random bit whereas $R \odot X$ is completely determined by $f(X) \circ R$, $\mathcal{E}_n$ has one more bit of entropy than $\mathcal{D}_n$. Furthermore, since f is a one-to-one function, $\mathrm{Ent}(\mathcal{D}_n) = 2n$, which is the entropy of the input to g. ■

7.3 One-way regular function → pseudo-entropy generator

In this subsection we show how to construct a pseudo-entropy generator from a one-way regular function. This result was previously obtained by [9, Goldreich Krawczyk Luby] using a different construction and proof techniques.

DEFINITION 7.2 (regular function) *A function f is* $\mathrm{reg}(n)$-regular *if, for each $n \in \mathcal{N}$, for each $x \in \{0,1\}^n$, the number of siblings of x is* $\mathrm{reg}(n) - 1$. *Thus, for each $x \in \{0,1\}^n$, $|f^{-1}(f(x))| = \mathrm{reg}(n)$.*

The main result of this subsection shows how to construct a pseudo-entropy generator from any function f which is one-way and for which the function $|f^{-1}|$ is computable in polynomial time. 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. In general it is not the case that $|f^{-1}|$ can be computed in polynomial time, and considerably more effort is needed to construct a pseudo-entropy generator from one-way functions without this property. For regular functions, $|f^{-1}(f(x))|$ is the same for all x of a given length and, as we show, this makes it easy to overcome the problem that $|f^{-1}|$ may not be computable in polynomial time.

To see where we get into trouble with the construction given in Proposition 8 (page 11), suppose f is a one-way $\mathrm{reg}(n)$-regular function. Then, the degeneracy of f is $\log(\mathrm{reg}(n))$. The problem with applying directly the construction given in Proposition 8 is twofold. Suppose that $\mathrm{reg}(n)$ is fairly large, e.g. $\mathrm{reg}(n) = 2^{n/4}$, and consider the probability ensembles $\mathcal{D}$ and $\mathcal{E}$ defined in Corollary 11 (page 12). These two ensembles are still computationally indistinguishable. However, $\mathrm{Ent}(\mathcal{D}_n) = 2n - \log(\mathrm{reg}(n))$, and thus we have lost $n/4$ bits of the input entropy. Furthermore, although $r \odot x$ is a hidden bit of g, it is no longer a meaningful bit. In fact, $\mathrm{Ent}(\mathcal{E}_n) \approx \mathrm{Ent}(\mathcal{D}_n)$, and even stronger, $\mathcal{D}_n$ and $\mathcal{E}_n$ are statistically indistinguishable within $\exp(-n)$.

The idea to overcome these problems is to create a new function which is the original one-way function concatenated with bits extracted out of the input using hashing to regain the lost entropy. Then, Proposition 8 can be applied to the new function to obtain a pseudo-entropy generator. This

[4] In fact, it is easy to concoct an example where the construction yields a generator g that is definitely not pseudo-random, even if f is a one-way one-to-one function. Consider the function $f(X) = f'(X) \circ 0$ where f' is a one-way permutation. Then, f is a one-way one-to-one function and yet the resulting g is not a pseudo-random generator, because the 0 bit at the end can always be predicted.

technique of extracting entropy out of the input to f equal to the degeneracy of f using techniques similar to that of Lemma 13 (page 15) has many applications in [15, Impagliazzo Luby]. We now give a highly intuitive presentation of this technique. Let $\mathrm{rank}(x) = |\{y < x : f(y) = f(x)\}|$, i.e. $\mathrm{rank}(x)$ is the rank of x among all of its siblings. For now, we make the highly unreasonable assumption that $\mathrm{rank}(x)$ is computable in polynomial time given x. Consider the function $\bar{f}(x) = f(x) \circ \mathrm{rank}(x)$. $\bar{f}(x)$ is one-to-one and so $\mathrm{Ent}(\bar{f}(\mathcal{U}_n)) = n$, i.e. $\bar{f}$ has degeneracy zero. Furthermore, the task of inverting $\bar{f}$ is at least as hard as that of inverting f. On input $f(x) \circ r$, an adversary doesn't just have to find *some* preimage of $f(x)$, it has to be able to find the r^{th} smallest preimage. Thus, if $\mathrm{rank}(x)$ is computable in polynomial time, then, using the same ideas as used in Theorem 1 (page 12), it is easy to construct a pseudo-random generator based on $\bar{f}$.

Unfortunately, the value of $\mathrm{rank}(x)$ is not in general computable in polynomial time. Suppose for the moment that the function d defined below is computable in time polynomial in n given $f(x)$. We show how to construct a pseudo-random generator based on this (still not justifiable) assumption.

DEFINITION 7.3 (the function d) *For $x \in \{0,1\}^n$, define $d(f(x)) = \mathrm{ilog}(|f^{-1}(f(x))|)$,*

The idea is to let the easily computable quantity $H \circ H(x)$, where $H \in \mathcal{H}_{n,d(f(x))+\mathrm{ilog}(2n)}$, serve the same purpose of the possibly hard to compute quantity $\mathrm{rank}(x)$. Redefine $\bar{f}(X \circ H) = f(X) \circ H \circ H(X)_{\leftarrow d(f(X))+\mathrm{ilog}(2n)}$.[5] Their are two claims: (1) $\bar{f}(X \circ H)$ is "almost" a one-to-one function, this follows from the following simple lemma; (2) $\bar{f}(X \circ H)$ is a one-way function if $f(X)$ is a one-way function, this follows using a proof based on Lemma 13 (page 15). From these two claims, using the same basic outline as given in Theorem 1 (page 12), it follows that a pseudo-random generator can be constructed from $\bar{f}$.

Lemma 17 *Let c be any positive integer, let $x \in \{0,1\}^n$ and let $H \in_U \mathcal{H}_{n,n+c}$. Then, $f(x)$, H and $H(x)_{\leftarrow d(f(x))+c}$ together uniquely determine x with probability at least $1 - 1/2^c$.*

PROOF: For any $x' \neq x$ and for any $i \leq n+c$, when $H \in_U \mathcal{H}_{n,n+c}$, $\Pr[H(x)_{\leftarrow i} = H(x')_{\leftarrow i}] = 2^{-i}$. There is ambiguity in determining x from the given information if and only if there is some $x' \in f^{-1}(f(x))$ such that $H(x') = H(x)$. The lemma follows because $\Pr[\exists x' \in f^{-1}(f(x)) - \{x\} : H(x')_{\leftarrow d(f(x))+c} = H(x)_{\leftarrow d(f(x))+c}] \leq \frac{|f^{-1}(f(x))|-1}{2^{d(f(x))+c}} \leq \frac{1}{2^c}$. ■

We now formalize the above intuition, first introducing a technical lemma that is used in the proofs of Lemma 18 and later on in Lemma 25, then stating and proving the informal claims made above under the assumption that d is polynomial time computable, and finally stating formally the result under this assumption in Theorem 3.

DEFINITION 7.4 (statistical coverage) *Let $\mathcal{D}$ and $\mathcal{E}$ be probability distributions on $\{0,1\}^n$, We say $\mathcal{E}$ θ-*statistically covers *$\mathcal{D}$ if, for all $x \in \{0,1\}^n$, $\mathcal{E}[\{x\}] - \theta\mathcal{D}[\{x\}] \geq 0$.*

[5] To avoid variable length outputs of functions on inputs of the same length, we adopt the convention that whenever a hash function is applied to an argument and only a prefix of the entire hash value is specified as output, then the output of the hash applied to the argument is padded out to a fixed maximum length with a string of zeroes. For example, $H(X)_{\leftarrow d(f(X))+\mathrm{ilog}(2n)}$ is padded with a string of zeroes so that the entire length is the maximum possible length $n + \mathrm{ilog}(2n)$.

The intuitive motivation for statistical coverage is the following. Suppose an adversary A inverts f with probability p when the input distribution to A is $\mathcal{D}$, and furthermore suppose that $\mathcal{E}$ θ-dominates $\mathcal{D}$. Then, A inverts f with probability at least p/θ when the input distribution to A is $\mathcal{E}$.

Lemma 18 *Let f be a one-way function. Suppose that $d(f(x))$ is computable in time polynomial in n given $f(x)$. Define $\bar{f}(X \circ H) = f(X) \circ H \circ H(X)_{\leftarrow d(f(X))+\mathrm{ilog}(2n)}$, where $X \in_{\mathcal{U}} \{0,1\}^n$ and $H \in_{\mathcal{U}} \mathcal{H}_{n,n+\mathrm{ilog}(2n)}$. Then,*

(1) $\bar{f}(X \circ H)$ is a one-way function.

(2) $\mathrm{Ent}(\bar{f}(X \circ H)) \geq n + |H| - \frac{1}{2}$.

PROOF:

Proof of (1): Suppose adversary A inverts $\bar{f}(X \circ H)$ with probability $p(n)$ which is non-negligible. We prove that the following oracle Turing machine M^A on input $Y = f(X)$ finds $X' \in f^{-1}(Y)$ with probability at least $\frac{p(n)^3}{16n}$.

Description of $M^A(Y)$

Compute $d(Y)$.

$H \in_{\mathcal{U}} \mathcal{H}_{n,d(Y)+\mathrm{ilog}(2n)}$

$\alpha \in_{\mathcal{U}} \{0,1\}^{d(y)+\mathrm{ilog}(2n)}$.

Run A on input $Y \circ H \circ \alpha$.

If A outputs $X' \circ H$ with $f(X') = Y$ then output X'.

Let $\mathcal{D}_n$ be the distribution defined by $f(X) \circ H \circ H(X)$, let $\mathcal{D}'_n$ be defined by $f(X) \circ H \circ \alpha_{\leftarrow d(f(X))-2\mathrm{ilog}(2/p(n))} \circ H(X)_{d(f(X))-2\mathrm{ilog}(2/p(n))\rightarrow}$ and let $\mathcal{E}_n$ be defined by $f(X) \circ H \circ \alpha$. A successfully finds an inverse of f with probability at least $p(n)$ when the input distribution is $\mathcal{D}_n$. By Lemma 13, for each fixed value of $Y = y$, when $X' \in_{\mathcal{U}} f^{-1}(y)$ the distribution defined by $y \circ H \circ H(X')_{\leftarrow d(y)-2\mathrm{ilog}(2/p(n))}$ is statistically indistinguishable within $\frac{p(n)}{2}$ from the distribution defined by $y \circ H \circ \alpha_{\leftarrow d(y)-2\mathrm{ilog}(2/p(n))}$. From this it follows that $\mathcal{D}_n$ and $\mathcal{D}'_n$ are statistically indistinguishable within $\frac{p(n)}{2}$. From this and Proposition 3 (page 9) it follows that A finds an inverse of f with probability at least $p(n) - \frac{p(n)}{2} = \frac{p(n)}{2}$ when the input distribution is $\mathcal{D}'_n$. It is easy to see that $\mathcal{E}_n$ $\frac{8n}{p(n)^2}$-statistically covers $\mathcal{D}'_n$, and thus A finds an inverse of f with probability at least $\frac{p(n)^3}{16n}$ when the input distribution is $\mathcal{E}_n$.

Proof of (2): Fix y in the range of f and let $x \in f^{-1}(y)$. From Lemma 17, $y \circ H \circ H(x)_{\leftarrow d(y)+\mathrm{ilog}(2n)}$ uniquely determines x with probability at least $1 - \frac{1}{2n}$. Thus, $\mathrm{Ent}(\bar{f}(\mathcal{U}_{n+|H|})) \geq |H| + n(1 - \frac{1}{2n}) \geq |H| + n - \frac{1}{2}$. ■

Corollary 19 *Let f be a one-way function. Suppose that $d(f(x))$ is computable in time polynomial in n given $f(x)$. Define $\bar{f}(X \circ H) = f(X) \circ H \circ H(X)_{\leftarrow d(f(X))+\mathrm{ilog}(2n)}$, define $\bar{b}(X \circ H \circ R) = (X \circ H) \odot R$ and define*

$$g(X \circ H \circ R) = \bar{f}(X \circ H) \circ R \circ \bar{b}(X \circ H \circ R),$$

where $X \in_{\mathcal{U}} \{0,1\}^n$, $H \in_{\mathcal{U}} \mathcal{H}_{n,n+\mathrm{ilog}(2n)}$ and $R \in_{\mathcal{U}} \{0,1\}^{n+|H|}$. Then, g is a pseudo-entropy generator.

PROOF: Let probability ensembles $\bar{\mathcal{D}}$ and $\bar{\mathcal{E}}$ be defined (analogously to $\mathcal{D}$ and $\mathcal{E}$ in Corollary 11 (page 12)) as $\bar{\mathcal{D}}_n = \bar{f}(X \circ H) \circ R \circ \bar{b}(X \circ H \circ R)$ and $\bar{\mathcal{E}}_n = \bar{f}(X \circ H) \circ R \circ \beta$, where $\beta \in_{\mathcal{U}} \{0,1\}$. From Lemma 18, part (1) and Corollary 11 it follows that $\bar{\mathcal{D}}$ and $\bar{\mathcal{E}}$ are computationally indistinguishable. From Lemma 18, part (2) it follows that $\mathrm{Ent}(\bar{\mathcal{E}}_n) \geq 2(n+|H|)+1/2$. On the other hand, the input entropy to g is $2(n+|H|)$, and thus it follows that g is a pseudo-entropy generator. ■

Theorem 3 *Let f be a one-way function such that $d(f(x))$ is computable in time polynomial in n given $f(x)$. Then, a pseudo-random generator can be constructed from f.*

PROOF: Combine Corollary 19 with Theorem 2 (page 18). ■

We use Corollary 19 to prove that a pseudo-random generator can be constructed from any one-way regular function (Theorem 4). Theorem 4 was originally proved by [9, Goldreich Krawczyk Luby] using a different construction and proof. The obvious idea to prove this theorem from the above is to apply Corollary 19 to obtain a pseudo-entropy generator and then the proof follows by Theorem 2 (page 18). The only difficulty with directly applying Corollary 19 is that Corollary 19 assumes that $d(f(x)) = \mathrm{ilog}(\mathrm{reg}(n))$ can be computed in polynomial time, and this might not be the case for an arbitrary one-way regular function. When $\mathrm{ilog}(\mathrm{reg}(n))$ cannot be computed in polynomial time then the following theorem provides a way to circumvent this problem using Lemma 6 (page 9).

Theorem 4 *A pseudo-random generator can be constructed from any one-way regular function.*

PROOF:

For inputs of length n, and for all $i = 0, \ldots, n$, let $g_{i,n}$ be the generator obtained by assuming that $\mathrm{ilog}(\mathrm{reg}(n)) = i$. For each value of n, for one value of i this assumption is right. Thus, from Corollary 19, for each n there is a value $s(n) \in \{0, \ldots, n\}$ such that family $\{g_{s(n),n} : n \in \mathcal{N}\}$ has pseudo-entropy. Then, from Theorem 2 (page 18) and from Proposition 5 (page 9) the family $\{g_{i,n} : i = 0, \ldots, n, n \in \mathcal{N}\}$ can be converted into a family of generators $\{G_{i,n} : i = 0, \ldots, n, n \in \mathcal{N}\}$, where $G_{i,n}$ maps n bits to more than $n(n+1)$ bits, such that the subfamily $\{G_{s(n),n} : n \in \mathcal{N}\}$ is pseudo-random. Then, from Lemma 6 (page 9) it follows that the polynomial time computable family of generators $\{G_n : n \in \mathcal{N}\}$ is pseudo-random, where $G_n \equiv \oplus_{i=0}^{n} G_{i,n}$.

■

8 False-entropy generator

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 8.1 (false-entropy generator) *We say that f is a* false-entropy generator *if f has computational entropy at least $s(n)$ where $s(n) \geq \mathrm{Ent}(f(\mathcal{U}_n)) + 1/n^c$ for some constant c. The* false entropy *of f is $s(n) - \mathrm{Ent}(f(\mathcal{U}_n))$.*

As the following lemma shows, it is easy to see that a function that hides a meaningful bit is also a false-entropy generator.

Lemma 20 *Let $X \in_{\mathcal{U}} \{0,1\}^n$ and let $f'(X) = f(X) \circ b(X)$ where $b(X)$ is a hidden and meaningful bit of $f'(X)$ (see Definition 5.1 on page 10). Then f' is a false-entropy generator.*

PROOF: Let c be a constant such that an unbounded adversary has prediction probability $p(n) > 1/n^c$ for $b(X)$ given $f(X)$. Let $\mathcal{D}$ be the probability ensemble defined by $\mathcal{D}_n = f'(X)$ and let $\mathcal{E}$ be the probability ensemble defined by $\mathcal{E}_n = f(X) \circ \beta$ where $\beta \in_{\mathcal{U}} \{0,1\}$. Since $b(X)$ is hidden given $f(X)$, by Corollary 11 (page 12), $\mathcal{D}$ is computationally indistinguishable from $\mathcal{E}$. Also, both $\mathcal{D}$ and $\mathcal{E}$ are polynomially samplable. When $b(X)$ is $p(n)$-meaningful for $f'(X)$, it can be shown that the entropy $b(X)$ adds to the preceeding bits is at most $1 - q(n)$ and thus $\mathrm{Ent}(\mathcal{E}_n) \geq \mathrm{Ent}(\mathcal{D}_n) + q(n)$, where $q(n) \approx p(n)^2$. ■

8.1 False-entropy generator → pseudo-entropy generator

Our present goal is to transform a function f with false entropy into a pseudo-entropy generator g. The major obstacle is that f could be many-to-one. In this case, even though the output of f seemingly has more entropy than it really has, the Shannon entropy of the output of f may be much less than the length of the input; intuitively the application of f to the input may cause more of a loss in Shannon entropy than the corresponding gain in false entropy.

We introduce a general method for recovering this loss in Shannon entropy without affecting the false entropy. This construction is related to the construction given in Subsection 7.3 of a pseudo-entropy generator from a one-way function $f(X)$ under the condition that $d(f(X))$ is polynomial time computable (see page 20 for the definition of d).

DEFINITION 8.2 *Let $k = k(n)$ and $j = j(n)$ be length functions, let $X, X_1, \ldots, X_k \in_{\mathcal{U}} \{0,1\}^n$ independently, let $X' = X_1 \circ \cdots X_k$ and let $f' = f^k$. Let $H \in_{\mathcal{U}} \mathcal{H}_{nk,j}$ and define $g(X' \circ H) = f'(X') \circ H \circ H(X')$.*

It turns out that $g(X' \circ H)$ is a pseudo-entropy generator for appropriate choices of k and j. The intuition is that the false entropy of $f'(X')$ is k times that of $f(X)$ and that with high probability $d(f'(X'))$ is close to the degeneracy of $f'(X')$. (On the other hand, $d(f(X))$ is on average within one of the degeneracy of $f(X)$, but the variation of $d(f(X))$ can be quite high.) Note that the degeneracy of $f(X)$ is $n - \mathrm{Ent}(f(X))$ and thus the degeneracy of $f'(X')$ is k times this quantity. The construction of g from f' is similar to that given in Subsection 7.3. We set j roughly equal to the degeneracy of $f'(X')$, and the intuition is that H extracts entropy equal to the degeneracy of $f'(X')$ from X' without compromising the false entropy of $f'(X')$. As before, Lemma 13 (page 15) plays an instrumental in proving that $g(X' \circ H)$ is a pseudo-entropy generator.

The following two lemmas shows that how to set the parameters when f satisfies a technical condition: $\mathrm{Ent}(f(X))$ can be approximated fairly well in time polynomial in n. We can then use Lemma 6 (page 9) to construct a pseudo-entropy generator without this technical condition.

Lemma 21 *Fix $c > 0$, $k = n^c$ and $j = k(n - \mathrm{Ent}(f(X))) - 2nk^{2/3}$. Let $\mathcal{E}_n$ be the distribution defined by $g(X' \circ H)$ except that $H(X')$ is replaced by $R \in_{\mathcal{U}} \{0,1\}^{|H|+j}$. Then, $g(X' \circ H)$ is statistically indistinguishable from $\mathcal{E}_n$ within $\exp(-n)$.*

PROOF: The proof follows from the claim that with probability $1 - \exp(-n)$ with respect to fixed $x_1, \ldots, x_k$ chosen according to $X_1, \ldots, X_k$, respectively, the following distribution is quasi-random within $\exp(-n)$. For all $i = 1, \ldots, k$ let $X_i'(x_i) \in_{\mathcal{U}} f^{-1}(f(x_i))$. The distribution is $H \circ H(X_1'(x_1) \circ \ldots \circ X_k'(x_k))$. From Lemma 13 (page 15), this distribution is quasi-random within $\exp(-n)$ if the min-entropy of $X_1'(x_1) \circ \ldots \circ X_k'(x_k)$ is at least $j + nk^{2/3}$. The min-entropy of $X_1'(x_1) \circ \ldots \circ X_k'(x_k)$ is simply $\sum_{i=1,\ldots,k} \log(|f^{-1}(f(x_i))|)$. Now we again think of $x_1, \ldots, x_k$ as being random instead of fixed. Let $Z_i = \log(|f^{-1}(f(X_i))|)$. The range of Z_i is $[0, n]$, $\mathrm{Exp}[Z_i] = n - \mathrm{Ent}(f(\mathcal{U}_n))$ and the sum can be written as $\sum_{i=1,\ldots,k} Z_i$. Using Chernoff bounds, this sum is at least $k\mathrm{Exp}[Z_i] - nk^{2/3} = j + nk^{2/3}$ with probability $1 - \exp(-n)$. ■

Lemma 22 *If f has at least n^{-c} bits of false entropy and an approximation $a = a(n)$ of $\mathrm{Ent}(f(X))$ that is correct to within an additive error of $n^{-(c+1)}$ can be computed in polynomial time then, with $k = n^{3c+4}$ and $j = k(n - a) - 2nk^{2/3}$, $g(X' \circ H)$ is a pseudo-entropy generator.*

PROOF: We claim that $g(X' \circ H)$ has computational entropy at least $nk + |H| + 1$. Let $\mathcal{D}_n$ be computationally indistinguishable from $f(X)$ such that $\mathrm{Ent}(\mathcal{D}_n) \geq \mathrm{Ent}(f(X)) + n^{-c} \geq a + n^{-c} - n^{-(c+1)}$. (The last term is the round-off error involved in approximating $\mathrm{Ent}(f(X))$ by a.) From Lemma 21, $g(X' \circ H)$ is statistically indistinguishable from $\mathcal{E}_n$ within $\exp(-n)$. Since $\mathcal{D}_n$ is computationally indistinguishable from $f(X)$, $\mathcal{E}_n$ is computationally indistinguishable from $\mathcal{D}_n^k \circ R$ by Proposition 12 (page 13). $\mathcal{D}_n^k \circ R$ has Shannon entropy $k\mathrm{Ent}(\mathcal{D}_n) + |R| = k\mathrm{Ent}(\mathcal{D}_n) + |H| + j$. Since $\mathrm{Ent}(\mathcal{D}_n) \geq a + n^{-c} - n^{-(c+1)}$, this quantity is at least $nk + |H| + 1$ for our choice of k. ■

We get rid of the technical assumption that we can approximate the entropy of the false entropy generator by appealing to Lemma 6 (page 9).

Theorem 5 *A pseudo-random generator can be constructed from any function f that has false entropy at least n^{-c} for some constant $c \geq 0$.*

PROOF: We can try n^{c+2} values

$$\frac{1}{n^{c+1}}, \frac{1}{n^{c+1}}, \ldots, \frac{n^{c+2}-1}{n^{c+1}}, \frac{n^{c+2}}{n^{c+1}} = n$$

for a and be guaranteed that at least one of these values is within an additive factor $n^{-(c+1)}$ of $\mathrm{Ent}(f(X))$. Each of these n^{c+2} values, combining the constructions given in Lemma 22, Theorem 2 (page 18) and Proposition 5 (page 9), yields a candidate for a pseudo-random generator stretching a bit string of length n into one of length n^{c+4}. We then apply Lemma 6 (page 9) to obtain a pseudo-random generator that stretches a string of length n^{c+3} to one of length n^{c+4}. ■

8.2 One-way function → false-entropy generator

In the previous section we reduced the problem of constructing a pseudo-random generator to the problem of finding a function with false entropy. In this section, we present the final step of the

main result; we show how to construct a function g with false entropy from *any* one-way function f.

To provide some intuition for the final construction, we first give a simpler construction which can be used to construct a false-entropy generator from f which has the following (weaker than we want) non-uniform property: If there is a successful adversary A for the false-entropy generator then we can construct (in possibly exponential time) a *non-uniform* successful adversary A' for inverting f. After presenting the simpler construction, we show how to construct a false-entropy generator g from f which has the following (what we want) uniform property: There is an oracle Turing machine M such that if A is a successful adversary for g then M^A is a successful adversary for inverting f.

DEFINITION 8.3 *Let* $\bar{f}(X \circ H \circ I \circ R) = f(X) \circ H \circ H(x)_{\leftarrow I+\mathrm{ilog}(2n)} \circ I \circ R$, *where* $I \in_{\mathcal{U}} \{0, \ldots, n\}$, $X \in_{\mathcal{U}} \{0,1\}^n$, $R \in_{\mathcal{U}} \{0,1\}^n$ *and* $H \in_{\mathcal{U}} \mathcal{H}_{n,n+\mathrm{ilog}(2n)}$.

We claim that in a non-uniform sense $\bar{f}(X \circ H \circ I \circ R) \circ (R \odot X)$ is a false-entropy generator. Let distribution $\mathcal{E}_n$ be defined by $\bar{f}(X \circ H \circ I \circ R) \circ (R \odot X)$ and let $\mathcal{D}_n$ be the same as $\mathcal{E}_n$ with the exception that if $I = d(f(X))$ (recall the definition of d from page 20) then $R \odot X$ is replaced by a random bit $\beta \in_{\mathcal{U}} \{0,1\}$. Using a proof similar to that of Lemma 18 (page 21) part (1), one can easily prove that a feasible adversary that has non-negligible distinguishing probability for $\mathcal{E}_n$ and $\mathcal{D}_n$ can be uniformly converted into a feasible adversary that inverts f with non-negligible probability. Similarly, because $I = d(f(X))$ with probability $\frac{1}{n+1}$, using Lemma 18 part (2) one can easily prove that the entropy of $\mathcal{D}_n$ is greater than that of $\mathcal{E}_n$ by an additive factor of at least $\frac{1}{2(n+1)}$. Thus, we have in a sense shown that $\bar{f}$ is a generator with false entropy at least $\frac{1}{2(n+1)}$. But the problem is that, although we can sample $\mathcal{E}_n$ in polynomial time, we cannot sample $\mathcal{D}_n$ in polynomial time unless we can tell if $I = d(f(X))$ in polynomial time, and this is the source of the non-uniform nature of the reduction. Nevertheless, this yields a false-entropy generator in the non-uniform sense and putting together the constructions given in this paper for constructing a pseudo-random generator from a false-entropy generator yields a pseudo-random generator in the non-uniform sense described above. The interested reader is refered to [16, Impagliazzo Levin Luby] for the remaining details.

The construction of a false-entropy generator in the uniform sense essentially uses the same ideas, but the proof is more involved.

DEFINITION 8.4 *Define random variable* $Y = X \circ H \circ I \circ R$. *Define* $\mathrm{ipb}(Y) = X \odot R$. *Let* $p = \Pr[I < d(f(X))]$ *and let* $p' = \Pr[I \leq d(f(X))] = p + \frac{1}{n+1}$. *Let* $k = 6(n+1)^3$. *For* $j = 1, \ldots, k$, *let* $Y_1, \ldots, Y_k$ *be independent random variables distributed identically to* Y. *Let* $l = kp' - 2n^2$ *and let* $H' \in_{\mathcal{U}} \mathcal{H}_{k,l}$. *Define*

$$g(Y_1 \circ \cdots \circ Y_k \circ H') = \bar{f}(Y_1) \circ \cdots \circ \bar{f}(Y_k) \circ H' \circ H'(\mathrm{ipb}(Y_1) \circ \cdots \circ \mathrm{ipb}(Y_k)).$$

Our main claim is that g is a false-entropy generator. We assume for now that we know the exact value of p, and later we remove this assumption. We first introduce some more notation and then give some intuition for why this is the case before presenting the proof.

DEFINITION 8.5 *Let* $\mathcal{E}_n$ *be the distribution defined by the output of* g. *Define independent random variable* $\beta \in_{\mathcal{U}} \{0,1\}^l$ *and define* $g'(Y_1 \circ \cdots \circ Y_k \circ H' \circ \beta) = \bar{f}(Y_1) \circ \cdots \circ \bar{f}(Y_k) \circ H' \circ \beta$. *Let* $\mathcal{E}'_n$ *be*

the distribution defined by the output of g'. We call the initial part of the two distributions $\mathcal{E}_n$ and $\mathcal{E}'_n$ where they are the same the identical part *and the remaining part (the last l bits) the* different part. *The interesting part of the two distributions is the different part. Finally, let $\mathcal{D}_n$ be the joint distribution defined by*

$$\mathcal{D}_n = g(Y_1 \circ \cdots \circ Y_k \circ H') \circ g'(Y_1 \circ \ldots \circ Y_k \circ H' \circ \beta).$$

Thus, the first half of a string randomly chosen according to $\mathcal{D}_n$ is distributed according to $\mathcal{E}_n$, the second half is distributed according to $\mathcal{E}'_n$ and the distribution on the two halves is correlated, i.e. the two halves are exactly the same in the identical parts.

The intuition is that $\mathcal{E}_n$ and $\mathcal{E}'_n$ are computationally indistinguishable, and thus the false-entropy of g is the difference between the entropy of $\mathcal{E}'_n$ and the entropy of $\mathcal{E}_n$. Lemma 23 shows that $\mathcal{E}'_n$ has significantly more entropy than $\mathcal{E}_n$ and Lemma 25 shows that these two distributions are computationally indistinguishable. An important feature of this proof is that $\mathcal{D}_n$ is a distribution which is polynomial time samplable. This is one reason why we are able to prove that g is a false-entropy generator in the uniform sense.

Lemma 23 $\mathrm{Ent}(\mathcal{E}'_n) \geq \mathrm{Ent}(\mathcal{E}_n) + 1$.

PROOF: The entropy in $\mathcal{E}'_n$ and $\mathcal{E}_n$ in the identical part is exactly the same. The additional entropy in the different part of $\mathcal{E}'_n$ is equal to $l = kp' - 2n^2 \geq kp + 4n^2$.

For each j where $I_j < d(f(X_j))$, the amount of entropy added to the different part of $\mathcal{E}_n$ by $\mathrm{ipb}(Y_j)$ is at most 1. On the other hand, as in the proof of part (2) of the proof of Lemma 18 (see page 21), under the condition that $I_j \geq d(f(X_j))$, $\mathrm{ipb}(Y_j)$ is determined by $\bar{f}(Y_j)$ with probability at least $1 - \frac{1}{2n}$. From this it follows that the extra entropy added to the different part of $\mathcal{E}_n$ by $\mathrm{ipb}(Y_j)$ is at most $p + \frac{1}{2n}$, and therefore the total additional entropy in the different part of $\mathcal{E}_n$ is at most $k(p + \frac{1}{2n}) \leq kp + 4n^2 - 1$. Thus, $\mathrm{Ent}(\mathcal{E}'_n) - \mathrm{Ent}(\mathcal{E}_n) \geq 1$. ■

Our next goal is to prove that $\mathcal{E}'_n$ and $\mathcal{E}_n$ are computationally indistinguishable. Our argument is slightly nonstandard in that it is not a straightforward hybrid argument. We assume for contradiction that there is a feasible adversary A that can distinguish the two distributions, and show that this implies there is a feasible adversary A' for inverting f.

DEFINITION 8.6 *Let A be a feasible adversary such that $\delta = \Pr[A(\mathcal{E}_n) = 1] - \Pr[A(\mathcal{E}'_n) = 1]$ is positive and non-negligible.*

DEFINITION 8.7 *(the hybrid distributions) For $j = 0, \ldots, k$ we define hybrid distribution $\bar{\mathcal{D}}_j$ inductively as follows. $\bar{\mathcal{D}}_0 = \mathcal{D}_n$. For $j = 1, \ldots, k$, $\bar{\mathcal{D}}_j$ is defined in terms of the following set of independent random variables: $R_1, \ldots, R_j \in_{\mathcal{U}} \{0,1\}^m$ where m is polynomial in $1/\delta$ and n; $\alpha_1, \ldots, \alpha_j \in_{\mathcal{U}} \{0,1\}$; $B_1, \ldots, B_j$, where $\Pr[B_i = 1] = p'$ for all $i = 1, \ldots, j$; $Y_{j+1}, \ldots, Y_k$, H' and β as in the above definition of $\mathcal{D}_n$. $\bar{\mathcal{D}}_j$ is of the form $\bar{\mathcal{E}}_j \circ \bar{\mathcal{E}'}_j$. Choosing a random instance according to $\bar{\mathcal{D}}_j$ can be thought of as follows: Randomly choose instances $r_1, \ldots, r_j$ of $R_1, \ldots, R_j$ and instances $b_1, \ldots, b_j$ of $B_1, \ldots, B_j$, and let $\vec{r} =< r_1, \ldots, r_j >$ and $\vec{b} =< b_1, \ldots, b_j >$. Use the algorithm described below to determine how $\vec{r}$ and $\vec{b}$ fix the values of $y_1, \ldots, y_j$. The conditional distribution with respect to $\vec{r}$ and $\vec{b}$ is $\bar{\mathcal{D}}_j(\vec{r}, \vec{b}) = \bar{\mathcal{E}}_j(\vec{r}, \vec{b}) \circ \bar{\mathcal{E}'}_j(\vec{r}, \vec{b})$, where $\bar{\mathcal{E}}_j(\vec{r}, \vec{b})$ is of the form*

$$\bar{f}(y_1) \circ \cdots \circ \bar{f}(y_j) \circ \bar{f}(Y_{j+1}) \circ \cdots \circ \bar{f}(Y_k) \circ$$

$$H' \circ H'(\gamma(b_1, y_1, \alpha_1) \circ \cdots \circ \gamma(b_j, y_j, \alpha_j) \circ \mathrm{ipb}(Y_{j+1}) \circ \cdots \circ \mathrm{ipb}(Y_k)).$$

The function $\gamma(b_i, y_i, \alpha_i)$ uses b_i to select as its output either $\mathrm{ipb}(y_i)$ or α_i: $b_i = 0$ implies $\gamma(b_i, y_i, \alpha_i) = \mathrm{ipb}(y_i)$ whereas $b_i = 1$ implies $\gamma(b_i, y_i, \alpha_i) = \alpha_i$. $\bar{\mathcal{E}'}_j(\vec{r}, \vec{b})$ is of the same form as $\bar{\mathcal{E}}_j(\vec{r}, \vec{b})$ except that the last l bits are replaced by random variable β.

Description of Distribution $\bar{\mathcal{D}}_j(\vec{r}, \vec{b})$

Let $N_0 = \frac{kn}{\delta}$.

Let $N_1 = N_0 \mathrm{ilog}(N_0)$.

Let $\vec{r}^* =< r_1, \ldots, r_{j-1} >$ and let $\vec{b}^* =< b_1, \ldots, b_{j-1} >$. The values of $\vec{r}^*$ and $\vec{b}^*$ determine the values of $y_1, \ldots, y_{j-1}$ and how the functions $\gamma(b_1, y_1, \alpha_1), \ldots, \gamma(b_{j-1}, y_{j-1}, \alpha_{j-1})$ are defined. These parameters determine the distribution $\bar{\mathcal{D}}_{j-1}(\vec{r}^*, \vec{b}^*)$. The bits of r_j are used to make all of the random choices described below, and b_j is used as a selector.

Use r_j to choose N_0 instances S_j of Y_j.

For each $y_j \in S_j$: use r_j to choose N_1 samples according to $\bar{\mathcal{E}}_{j-1}(\vec{r}^*, \vec{b}^*, \bar{f}(Y_j) = \bar{f}(y_j), \mathrm{ipb}(Y_j) = \gamma(b_j, y_j, \alpha_j))$ and let $p(y_j)$ be the fraction of these samples for which A outputs 1, and use r_j to choose N_1 samples according to $\bar{\mathcal{E}'}_{j-1}(\vec{r}^*, \vec{b}^*, \bar{f}(Y_j) = \bar{f}(y_j))$ and let $p'(y_j)$ be the fraction of these samples for which A outputs 1.

Let y_j be the string in S_j that maximizes $p(y_j) - p'(y_j)$.

Set $\bar{\mathcal{E}}_j(\vec{r}, \vec{b}) = \bar{\mathcal{E}}_{j-1}(\vec{r}^*, \vec{b}^*, \bar{f}(Y_j) = \bar{f}(y_j), \mathrm{ipb}(Y_j) = \gamma(b_j, y_j, \alpha_j))$

Set $\bar{\mathcal{E}'}_j(\vec{r}, \vec{b}) = \bar{\mathcal{E}'}_{j-1}(\vec{r}^*, \vec{b}^*, \bar{f}(Y_j) = \bar{f}(y_j))$.

DEFINITION 8.8 *For all values of $\vec{r}$ and $\vec{b}$, let*

$$\delta_j(\vec{r}, \vec{b}) = \Pr[A(\bar{\mathcal{E}}_j(\vec{r}, \vec{b})) = 1] - \Pr[A(\bar{\mathcal{E}'}_j(\vec{r}, \vec{b})) = 1],$$

and let $\delta_j = \delta_j(\vec{R}, \vec{B})$.

The intuition for this choice of probability distribution is that the event $B_j = 1$ occurs with exactly the same probability as the event $I_j \leq d(f(X_j))$ occurs. With respect to the distribution on Y_j conditional on $I_j \leq d(f(X_j))$ no feasible adversary can tell the difference between $\mathrm{ipb}(Y_j)$ and a random bit $\alpha_j \in_{\mathcal{U}} \{0, 1\}$. Although we cannot tell for a particular y_j whether $i_j \leq d(f(x_j))$, in the above algorithm it is the case that with high probability there are several $y_j \in S_j$ such that $i_j \leq d(f(x_j))$. By fixing y_j in $\bar{\mathcal{D}}_{j-1}(\vec{R}, \vec{b})$ to the string which maximizes $p(y_j) - p'(y_j)$, we can show that if there is no feasible adversary that can tell the difference between $\mathrm{ipb}(Y_j)$ under the condition $I_j \leq d(f(X_j))$ from a random bit α_j then $\delta_j(\vec{R}, \vec{b})$ is very close to $\delta_{j-1}(\vec{R}, \vec{b})$, even in the case when $b_j = 1$ and $\mathrm{ipb}(Y_j)$ is replaced with a random bit α_j in the distribution $\bar{\mathcal{E}}_j(\vec{R}, \vec{b})$.

Lemma 24 *$\bar{\mathcal{E}}_k$ and $\bar{\mathcal{E}'}_k$ are statistically indistinguishable within $\exp(-n)$.*

PROOF: For every fixed $\vec{r}$, the identical parts of the two distributions are exactly the same string, and the only portion where the two distributions differ is in the different part, i.e. the

last l bits. In $\bar{\mathcal{E}'}_k(\vec{r},\vec{B})$ these bits are chosen completely randomly while in $\bar{\mathcal{E}}_k(\vec{r},\vec{B})$ these bits are chosen as $H'(\gamma(B_1,y_1,\alpha_1)\circ\cdots\circ\gamma(B_k,y_k,\alpha_k))$, where $y_1,\ldots,y_k$ are fixed according to $\vec{r}$ and $\vec{B}$. Recall H' is a random hash function that maps to $l = kp' - 2n^2$ bits. By standard Chernoff bounds, when a random instance $\vec{b}$ of $\vec{B}$ is chosen the probability that fewer than $kp' - k^{2/3}/7 \geq kp' - n^2$ of the bits of $\vec{b}$ are set to 1 is $\exp(-n)$. In the positions where $b_i = 0$, $\gamma(b_i,y_i,\alpha_i)$ is fixed to $\mathrm{ipb}(y_i)$, but in the positions where $b_i = 1$, $\gamma(b_i,y_i,\alpha_i)$ is a uniformly distributed random variable α_i. From this and Lemma 13 (page 15), when $b_i = 1$ for at least $kp' - n^2$ values of i, $H'\circ H'(\gamma(b_1,y_1,\alpha_1)\circ\cdots\circ\gamma(b_k,y_k,\alpha_k))$ and $H'\circ\beta$ are statistically indistinguishable within $\exp(-n)$. Thus, $\bar{\mathcal{E}}_k(\vec{r},\vec{B})$ and $\bar{\mathcal{E}'}_k(\vec{r},\vec{B})$ are statistically indistinguishable within $\exp(-n)$ for every setting of $\vec{r}$. ■

Lemma 25 *Let $X \in_U \{0,1\}^n$. If there is a feasible adversary A that distinguishes $\mathcal{E}_n$ from $\mathcal{E}'_n$ with non-negligible probability at least δ then there is a feasible adversary A'^A that inverts $f(X)$ with probability at least $\frac{\delta}{16c}$.*

PROOF:

We define step j to be *uneventful* if $\delta_j \geq \delta - \frac{j\delta}{2k}$, and otherwise the step is *eventful*. Note that if j is the first eventful step then $\delta_{j-1} - \delta_j > \frac{\delta}{2k}$. By the definition of uneventful, if all k steps are uneventful then the distinguishing probability of adversary A for the two distributions $\bar{\mathcal{E}}_k$ and $\bar{\mathcal{E}'}_k$ is at least $\delta/2$. Lemma 24 shows that this distinguishing probability is at most $\exp(-n)$, and thus there is at least one eventful step, and from this we construct a feasible adversary A' for inverting f with non-negligible probability. Let j be the first eventful step.

Define

$$\delta_j^0 = \Pr[A(\bar{\mathcal{E}}_{j-1}(I_j > d(f(X_j)))) = 1] - \Pr[A(\bar{\mathcal{E}'}_{j-1}(I_j > d(f(X_j)))) = 1]$$

and

$$\delta_j^1 = \Pr[A(\bar{\mathcal{E}}_{j-1}(I_j \leq d(f(X_j)))) = 1] - \Pr[A(\bar{\mathcal{E}'}_{j-1}(I_j \leq d(f(X_j)))) = 1].$$

Since $I_j \leq d(f(X_j))$ with probability p', we have

$$\delta_{j-1} = p'\delta_j^1 + (1-p')\delta_j^0.$$

Define

$$\epsilon_j^0 = \Pr[A(\bar{\mathcal{E}}_j(B_j = 0)) = 1] - \Pr[A(\bar{\mathcal{E}'}_j(B_j = 0)) = 1]$$

and

$$\epsilon_j^1 = \Pr[A(\bar{\mathcal{E}}_j(B_j = 1)) = 1] - \Pr[A(\bar{\mathcal{E}'}_j(B_j = 1)) = 1].$$

Since $B_j = 0$ with probability p', we have

$$\delta_j = \Pr[A(\bar{\mathcal{E}}_j) = 1] - \Pr[A(\bar{\mathcal{E}'}_j) = 1] = p'\epsilon_j^1 + (1-p')\epsilon_j^0.$$

Since $\delta_{j-1} - \delta_j > \frac{\delta}{2k}$, one of the following must be true.

(1) $\delta_j^0 - \epsilon_j^0 > \frac{\delta}{2k}$.

(2) $\delta_j^1 - \epsilon_j^1 > \frac{\delta}{2k}$.

We first show that (1) can't be true, and then we show (2) implies that the adversary A'^A_j described below inverts f with non-negligible probability.

To prove (1) can't be true, note that $\delta^0_j(\vec{R}^*, \vec{b}^*)$ is the difference between the accepting probability of A for $\bar{\mathcal{E}}_{j-1}(\vec{R}^*, \vec{b}^*, I_j > d(f(X_j)))$ and $\bar{\mathcal{E}}'_{j-1}(\vec{R}^*, \vec{b}^*, I_j > d(f(X_j)))$. On the other hand, by the way $\bar{\mathcal{D}}_j(\vec{R}, \vec{b})$ is constructed when $b_j = 0$, $\epsilon^0_j(\vec{R}, \vec{b})$ is determined by the value $y_j \in S_j$ that corresponds to the largest estimate, based on N_1 samples, of the difference between the accepting probability of A under the distribution $\bar{\mathcal{E}}_{j-1}(\vec{R}^*, \vec{b}^*, \bar{f}(Y_j) = \bar{f}(y_j), \mathrm{ipb}(Y_j) = \mathrm{ipb}(y_j))$ and $\bar{\mathcal{E}}'_{j-1}(\vec{R}^*, \vec{b}^*, \bar{f}(Y_j) = \bar{f}(y_j))$. We first observe that $1 - p' = \Pr[I_j > d(f(X_j))] \geq \frac{1}{n+1}$: If this is not true then there is an $x \in \{0,1\}^n$ such that $\mathrm{ilog}(|f^{-1}(f(x))|) \geq n$, which implies that at least half of the $x' \in \{0,1\}^n$ are preimages of $f(x)$, from which it easily follows that f is not at all one-way. From this and the choice of N_0, with probability $1 - \exp(-n)$ there is at least one $y_j \in S_j$ for which $\Pr[A(\bar{\mathcal{E}}_{j-1}(\vec{R}^*, \vec{b}^*, \bar{f}(Y_j) = \bar{f}(y_j), \mathrm{ipb}(Y_j) = \mathrm{ipb}(y_j))) = 1] - \Pr[A(\bar{\mathcal{E}}'_{j-1}(\vec{R}^*, \vec{b}^*, \bar{f}(Y_j) = \bar{f}(y_j))) = 1] \geq \delta^0_j(\vec{R}^*, \vec{b}^*) - \frac{\delta}{16c}$. (The probability is with respect to R_j.) Furthermore, N_1 is large enough so that for all $y_j \in S_j$, with probability $1 - \exp(-n)$, $p(y_j) - p'(y_j)$ is within $\frac{\delta}{16c}$ of $\Pr[A(\bar{\mathcal{E}}_{j-1}(\vec{R}^*, \vec{b}^*, \bar{f}(Y_j) = \bar{f}(y_j), \mathrm{ipb}(Y_j) = \mathrm{ipb}(y_j))) = 1] - \Pr[A(\bar{\mathcal{E}}'_{j-1}(\vec{R}^*, \vec{b}^*, \bar{f}(Y_j) = \bar{f}(y_j))) = 1]$. (Once again, this probability is with respect to R_j.) Thus, it follows that with probability $1 - \exp(-n)$, for the $y_j \in S_j$ corresponding to the largest estimate it is the case that $\Pr[A(\bar{\mathcal{E}}_{j-1}(\vec{R}^*, \vec{b}^*, \bar{f}(Y_j) = \bar{f}(y_j), \mathrm{ipb}(Y_j) = \mathrm{ipb}(y_j))) = 1] - \Pr[A(\bar{\mathcal{E}}'_{j-1}(\vec{R}^*, \vec{b}^*, \bar{f}(Y_j) = \bar{f}(y_j))) = 1] \geq \delta^0_j(\vec{R}^*, \vec{b}^*) - 3\frac{\delta}{16c}$, and from this it follows that

$$\delta^0_j - \epsilon^0_j \leq \delta^0_j - \left(\delta^0_j - 3\frac{\delta}{16c}\right) + \exp(-n) \leq \frac{\delta}{2k},$$

which contradicts (1).

We now show that (2) implies that the adversary A'^A_j described below inverts $f(X)$ with probability at least $\frac{\delta}{16c}$. Define

$$\hat{\delta}_j = |\Pr[A(\bar{\mathcal{E}}_{j-1}(I_j \leq d(f(X_j)))) = 1] - \Pr[A(\bar{\mathcal{E}}_{j-1}(I_j \leq d(f(X_j)), \mathrm{ipb}(Y_j) = \alpha_j)) = 1]|.$$

We first show it can't be the case that $\hat{\delta}_j < \frac{\delta}{8c}$. In particular, we show that $\hat{\delta}_j < \frac{\delta}{8c}$ implies that $\delta^1_j - \epsilon^1_j \leq \frac{\delta}{2k}$, and this contradicts (2). By the triangle inequality,

$$\Pr[A(\bar{\mathcal{E}}_{j-1}(\vec{R}^*, \vec{b}^*, I_j \leq d(f(X_j)), \mathrm{ipb}(Y_j) = \alpha_j)) = 1]$$

$$- \Pr[A(\bar{\mathcal{E}}'_{j-1}(\vec{R}^*, \vec{b}^*, I_j \leq d(f(X_j)))) = 1] \geq \delta^1_j(\vec{R}^*, \vec{b}^*) - \hat{\delta}_j(\vec{R}^*, \vec{b}^*).$$

On the other hand, by the way $\bar{\mathcal{D}}_j(\vec{R}, \vec{b})$ is constructed when $b_j = 1$, $\epsilon^1_j(\vec{R}, \vec{b})$ is determined by the value $y_j \in S_j$ that corresponds to the largest estimate, based on N_1 samples, of the difference between the accepting probability of A under the distribution $\bar{\mathcal{E}}_{j-1}(\vec{R}^*, \vec{b}^*, \bar{f}(Y_j) = \bar{f}(y_j), \mathrm{ipb}(Y_j) = \alpha_j)$ and $\bar{\mathcal{E}}'_{j-1}(\vec{R}^*, \vec{b}^*, \bar{f}(Y_j) = \bar{f}(y_j))$. It is easy to see that $p' \geq \frac{1}{2(n+1)}$. From this and the choice of N_0, with probability $1 - \exp(-n)$ there is at least one $y_j \in S_j$ for which $\Pr[A(\bar{\mathcal{E}}_{j-1}(\vec{R}^*, \vec{b}^*, \bar{f}(Y_j) = \bar{f}(y_j), \mathrm{ipb}(Y_j) = \alpha_j)) = 1] - \Pr[A(\bar{\mathcal{E}}'_{j-1}(\vec{R}^*, \vec{b}^*, \bar{f}(Y_j) = \bar{f}(y_j)))) = 1] \geq \Pr[A(\bar{\mathcal{E}}^1_{j-1}(\vec{R}^*, \vec{b}^*, I_j \leq d(f(X_j)), \mathrm{ipb}(Y_j) = \alpha_j)) = 1] - \Pr[A(\bar{\mathcal{E}}'_{j-1}(\vec{R}^*, \vec{b}^*, I_j \leq d(f(X_j)))) = 1] - \frac{\delta}{16c}$. Furthermore, N_1 is large enough so that for all $y_j \in S_j$, with probability $1 - \exp(-n)$, $p(y_j) - p'(y_j)$ is within $\frac{\delta}{16c}$ of $\Pr[A(\bar{\mathcal{E}}_{j-1}(\vec{R}^*, \vec{b}^*, \bar{f}(Y_j) = \bar{f}(y_j), \mathrm{ipb}(Y_j) = \alpha_j)) = 1] - \Pr[A(\bar{\mathcal{E}}'_{j-1}(\vec{R}^*, \vec{b}^*, \bar{f}(Y_j) =$

$\bar{f}(y_j)))) = 1]$. Thus, it follows that with probability $1 - \exp(-n)$, for the $y_j \in S_j$ corresponding to the largest estimate it is the case that $\Pr[A(\bar{\mathcal{E}}_{j-1}(\vec{R}^*, \vec{b}^*, \bar{f}(Y_j) = \bar{f}(y_j), \mathrm{ipb}(Y_j) = \alpha_j)) = 1] - \Pr[A(\bar{\mathcal{E}'}_{j-1}(\vec{R}^*, \vec{b}^*, \bar{f}(Y_j) = \bar{f}(y_j)))) = 1] \geq \delta_j^1(\vec{R}^*, \vec{b}^*) - \hat{\delta}_j(\vec{R}^*, \vec{b}^*) - 3\frac{\delta}{16c}$, and from this it follows that

$$\delta_j^1 - \epsilon_j^1 \leq \delta_j^1 - \left(\delta_j^1 - \hat{\delta}_j - 3\frac{\delta}{16c}\right) + \exp(-n) \leq \frac{\delta}{2k},$$

which contradicts (2).

Let $X \in_{\mathcal{U}} \{0,1\}^n$. We now show $\hat{\delta}_j \geq \frac{\delta}{8c}$ implies that the following adversary A'^A_j inverts $f(X)$ with probability at least $\frac{\delta}{16c}$.

Description of $A'^A_j(f(X))$

$I \in_{\mathcal{U}} \{0, \ldots, n\}$

$H \in_{\mathcal{U}} \mathcal{H}_{n,n+\mathrm{ilog}(2n)}$

$V \in_{\mathcal{U}} \{0,1\}^{I+\mathrm{ilog}(2n)}$

$T \in_{\mathcal{U}} \{0,1\}^n$.

Let M^A be the oracle Turing machine described in Proposition 10 (page 12) that tries to predict $X \odot T$ on inputs chosen according to $\bar{\mathcal{E}}_{j-1}(\bar{f}(Y_j) = f(X) \circ H \circ V \circ I \circ T, \mathrm{ipb}(Y_j) = 0)$ when computing B_0 and inputs chosen according to $\bar{\mathcal{E}}_{j-1}(\bar{f}(Y_j) = f(X) \circ H \circ V \circ I \circ T, \mathrm{ipb}(Y_j) = 1)$ when computing B_1. Note that $X \odot T$ is not part of the input to M^A.

Let M'^{M^A} be the oracle Turing machine described in Proposition 9 (page 11). Run this oracle machine on inputs as just described for M^A except that T is no longer considered as part of the input. The output is a list L of n-bit strings which contains the set of all $x' \in \{0,1\}^n$ such that M^A has prediction probability at least $\frac{\delta}{8c}$ for $X \odot T$.

For all $x' \in L$, if $f(x') = f(X)$ then output x'.

The proof is similar to the proof of Lemma 18 (page 21). $\hat{\delta}_j \geq \frac{\delta}{8c}$ and Proposition 10 (page 12) shows that the prediction probability of M^A for $X \odot T$ is at least $\frac{\delta}{4c}$ when the input distribution is $\bar{\mathcal{E}}_{j-1}(\bar{f}(Y_j) = f(X) \circ H \circ V \circ I \circ T, I \leq d(f(X)))$, except that the j^{th} inner product bit is missing from the input. Thus, it happens with probability at least $\frac{\delta}{8c}$ for a randomly chosen instance x and i of X and I, respectively, conditional on $i \leq d(f(x))$, that the prediction probability of M^A for $x \circ T$ is at least $\frac{\delta}{8c}$. From this and Proposition 9 (page 11) it follows that M'^{M^A} outputs an x' with $f(x') = f(X)$ with probability at least $\frac{\delta}{8c}$ when the inputs to M'^{M^A} are chosen according to $\bar{\mathcal{E}}_{j-1}(\bar{f}(Y_j) = f(X) \circ H \circ V \circ I \circ T, I \leq d(f(X)))$. When $I \leq d(f(X))$ then by Lemma 13 (page 15) the input distribution is statistically indistinguishable within $\frac{\delta}{16c}$ from the same distribution with the first $I - 2\mathrm{ilog}(\frac{16c}{\delta})$ bits of $H(X)_{\leftarrow I+\mathrm{ilog}(2n)}$ replaced by random bits. On the other hand, the same distribution where all bits of $H(X)_{\leftarrow I+\mathrm{ilog}(2n)}$ are replaced by random bits and the condition $I \leq d(f(X))$ is removed $\frac{2n(n+1)(16c)^2}{\delta^2}$-statistically covers this distribution. Since this is the actual distribution used within A'^A_j, it follows that A'^A_j outputs an inverse of $f(X)$ with probability at least $\frac{\delta^3}{2n(n+1)(16c)^3}$.

Finally, we note that for all $j = 1, \ldots, k$ the adversary A'^A_j can be uniformly constructed, and that the running time for the entire construction is polynomial in $\frac{1}{\delta}$ and n as required. The adversary A'^A on input $f(X)$ consists of running A'^A_j for all possible values of $j = 1, \ldots, k$ to try to invert $f(X)$. ■

Theorem 6 *If f is a one-way function and p is polynomial time computable then g is a false-entropy generator.*

PROOF: This follows directly from Lemmas 23 and 25. ■

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

PROOF: That pseudo-random generators imply one-way functions is well known. The converse now follows from Theorem 6 and Theorem 5, except for the assumption about knowing the exact value of p. However we can deal with this problem exactly as we dealt with the unknown false entropy in Theorem 5 (page 24). We make one generator for each value of $p = \frac{0}{n}, \frac{1}{n}, \frac{2}{n}, \ldots, \frac{n}{n}$ using independently chosen inputs, use Proposition 5 (page 9) to increase the length of the output of each generator appropriately, and then use Lemma 6 (page 9) to show that the exclusive-or of all the outputs yields a pseudo-random generator. ■

9 Open Problems

The results presented in this paper unify different concepts in theoretical cryptography. When combined with other work ([8, Goldreich Goldwasser Micali], [23, Luby Rackoff], [13, Goldreich Micali Wigderson], [25, Naor]), they show that applications ranging from private key encryption to zero-knowledge proofs can be based on any one-way function. [15, Impagliazzo Luby] shows that most cryptographic applications that are impossible in a world where anything that is informationally possible is computationally possible must be implicitly based on a one-way function.

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 well-known that a secure protocol can be constructed from a pseudo-random generator, and thus by the results described above a secure protocol can be constructed from a one-way function. [15, Impagliazzo Luby] provides some 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. Thus, secure protocols for any of these applications is equivalent to the existence of one-way functions. Other results can be found in [26, Naor Yung], which gives a signature scheme that can be based on any one-way permutation, and in [29, Rompel], which improves this by basing such a scheme on any one-way function. Some applications seem unlikely to be shown possible based on any one-way function, e.g. [18, Impagliazzo Rudich] give strong evidence that exchanging secrets is an application of this kind.

A fundamental issue is that of efficiency, both in size and time; the 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. This is not good news for practical applications; it would be nice to have a much more parsimonious construction. We would like to develop general constructions of pseudo-random generators that are as time and size efficient as the one-way function on which they are based.

A practical problem is to build a very efficient pseudo-random generator based on the conjectured difficulty of some well-studied problem. [17, Impagliazzo Naor] present some progress in this direction, they construct a pseudo-random generator based on the subset sum problem which is as efficient as RSA. Pseudo-random generators are typically constructed based on the conjectured difficulty of one specific problem, e.g. factoring, quadratic residuosity, discrete log, etc. A related line of research is to develop an efficient generator that is pseudo-random as long as one of several problems is one-way.

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 and Hugo Krawczyk 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 14 and Oded and Hugo for introducing the fourth author to this question.

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, L., Blum, M., Shub, M., "A Simple Secure Unpredictable Pseudo-Random Number Generator", *SIAM J. on Computing*, Vol. 15, No. 2, 1986, pp. 364-383. A preliminary version appeared in *CRYPTO 1982*.

[5] 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.

[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] 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.

[9] Goldreich, O., Krawczyk, H. and Luby, M., "On the Existence of Pseudorandom Generators", 29^{th} *FOCS*, 1988, pp. 12-24.

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

[11] 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.

[12] 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.

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

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

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

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

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

[19] Karp, R., Lipton, R., "Turing Machines that Take Advice," *L'Enseignement Mathematique*, Vol. 28, pp. 191-209 (1982). A preliminary version appeared in 12^{th} *STOC*, 1980.

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

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

[22] 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.

[23] 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.

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

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

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

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

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

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

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

[31] 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,

[32] 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.

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

[34] 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*.

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