Cryptographic Primitives Based on Hard Learning Problems

Avrim Blum*[1] Merrick Furst**[1] Michael Kearns[2] Richard J. Lipton[3]

[1] Carnegie Mellon University, Pittsburgh PA 15213
[2] AT&T Bell Labs, Murray Hill NJ 07974
[3] Princeton University, Princeton, NJ 08544

1 Introduction and Motivation

Modern cryptography has had considerable impact on the development of computational learning theory. Virtually every intractability result in Valiant's model [13] (which is *representation-independent* in the sense that it does not rely on an artificial syntactic restriction on the learning algorithm's hypotheses) has at its heart a cryptographic construction [4, 9, 1, 10]. In this paper, we give results in the reverse direction by showing how to construct several cryptographic primitives based on certain assumptions on the difficulty of learning. In doing so, we develop further a line of thought introduced by Impagliazzo and Levin [6].

As we describe, standard definitions in learning theory and cryptography do not appear to correspond perfectly in their original forms. However, we show that natural modifications to standard learning definitions can yield the desired connections. The particular cryptographic primitives we consider are pseudo-random bit generators, one-way functions, and private-key cryptosystems. We give transformations of hard learning problems into these cryptographic primitives with the desirable property that the complexity of the resulting primitive is not much greater than that of the hard-to-learn functions and distributions. In particular, our constructions are especially adept at preserving the degree of parallelism inherent in the hard functions and distributions.

Note that while it is well-known that some of the primitives above imply the existence of others (for instance, the equivalence of bit generators and one-way functions) [14, 7], we are interested in the separate results because the equivalences between primitives often do not preserve complexity measures such as circuit depth (parallelism). For instance, it is not known how to construct a bit generator in $\mathcal{NC}$ given a one-way function in $\mathcal{NC}$. One of the main potential benefits of this line of research is that as "simple" function classes (for instance, DNF formulae) continue to elude efficient learning, our belief in the intractability of learning such classes increases, and we can exploit this intractability to obtain simpler cryptographic primitives.

In addition to generic transformations, we describe a very simple pseudorandom bit generator based on the assumption that the class of parity functions

* Supported in part by an NSF Postdoctoral Fellowship.
** Supported in part by NSF grant CCR-9119319.

is hard to learn in the presense of random noise (an assumption similar to the intractability of decoding random linear codes). This generator is quite similar to a proposed one-way function due to Goldreich, Krawczyk and Luby [3], who then obtain a generator by running the one-way function through a generic transformation. We show that the output of this one-way function is already pseudorandom. This stronger assertion is apparently already known to some researchers in the cryptography community as a "folk theorem".

1.1 The Apparent Necessity of Average-Case Assumptions

In most learning theory models, a learning algorithm is required for *every* value of n to learn *all* functions over $\{0,1\}^n$ that meet some (usually strong) constraints known to the algorithm. For instance, we might ask that the learning algorithm be able to learn any DNF formula over $\{0,1\}^n$ with at most n^2 terms. Under such a definition, a failed DNF learning algorithm might still always be able to learn any n^2-term DNF provided n was even, and might learn all but a small scattered set of n^2-term DNF formulae when n was odd. A "hard" learning problem with such an algorithm seems not especially useful for cryptography.

We thus introduce an average-case model of learning that reduces these discrepancies between hardness for learning and the conditions required by cryptography, but otherwise preserves both the spirit and technical aspects of many existing models in learning theory. We then proceed to demonstrate that this worst-case/average-case discrepancy was essentially the only barrier to a natural correspondence between hard learning problems and many common cryptographic primitives.

2 Preliminaries

2.1 Learning Models

The learning models we consider are models of learning boolean functions from *labeled examples*.

Throughout the paper, $\mathcal{F}_n$ will denote a class of boolean functions over $\{0,1\}^n$, so each $f \in \mathcal{F}_n$ is a mapping $f : \{0,1\}^n \to \{0,1\}$. We assume that each function in $\mathcal{F}_n$ is represented using some fixed and reasonable *representation scheme*. A representation scheme for $\mathcal{F}_n$ is a pair $(\mathcal{R}_n, \mathcal{E}_n)$, where $\mathcal{R}_n \subseteq \{0,1\}^{r(n)}$ for some fixed polynomial $r(n)$, and $\mathcal{E}_n : \mathcal{R}_n \to \mathcal{F}_n$ is an onto mapping.

We interpret a string $\sigma \in \mathcal{R}_n$ as a *representation* of the function $\mathcal{E}_n(\sigma) \in \mathcal{F}_n$. Note that a function $f \in \mathcal{F}_n$ may have many representations in $\mathcal{R}_n$. Also, since $\mathcal{R}_n$ contains only $r(n)$-bit strings, we are insisting that each function in $\mathcal{F}_n$ have a "short" representation. We shall see that the computational details of *evaluating* the representations are relevant to our study. We define $\mathcal{F} = \{\mathcal{F}_n\}$ and $(\mathcal{R}, \mathcal{E}) = \{(\mathcal{R}_n, \mathcal{E}_n)\}$.

In our average-case learning models, the unknown target function will be generated according to some fixed distribution $\mathcal{P}_n$ over the function class $\mathcal{F}_n$ from the distribution *ensemble* $\mathcal{P} = \{\mathcal{P}_n\}$ over $\mathcal{F}$. When we have fixed a representation

scheme $(\mathcal{R}, \mathcal{E})$ for $\mathcal{F}$, sometimes we instead prefer to think of $\mathcal{P}_n$ as a distribution over representations $\mathcal{R}_n$, which implicitly defines a distribution over $\mathcal{F}_n$.

Once a target function $f \in \mathcal{F}_n$ is generated according to $\mathcal{P}_n$ (which we shall denote $f \in \mathcal{P}_n$), a learning algorithm will receive access to labeled examples of f selected according to some fixed distribution $\mathcal{D}_n$ over the input space $\{0,1\}^n$ from the distribution ensemble $\mathcal{D} = \{\mathcal{D}_n\}$. Each example is a pair $\langle x, f(x)\rangle$ where x is drawn randomly according to $\mathcal{D}_n$ (denoted $x \in \mathcal{D}_n$). If $S = x_1, \ldots, x_m$ is a sequence of inputs from $\{0,1\}^n$, we use $\langle S, f\rangle$ to denote the sequence $\langle x_1, f(x_1)\rangle, \ldots, \langle x_m, f(x_m)\rangle$ of labeled examples of f.

Definition 1. Let $\mathcal{F}$ be a class of boolean functions, $\mathcal{P}$ a distribution ensemble over $\mathcal{F}$, and $\mathcal{D}$ a distribution ensemble over $\{0,1\}^*$. For any $0 < \epsilon < 1/2$, we say that $\mathcal{F}$ is *ϵ-predictable on average with respect to $\mathcal{P}$ and $\mathcal{D}$* if there exists a polynomial time algorithm M (taking a labeled sample $\langle S, f\rangle$ and a test input $\tilde{x}$) and a polynomial $m(n)$ such that for infinitely many n,

$$\Pr_{f \in \mathcal{P}_n, S \in \mathcal{D}_n^{m(n)}, \tilde{x} \in \mathcal{D}_n}[M(\langle S, f\rangle, \tilde{x}) = f(\tilde{x})] \geq 1 - \epsilon.$$

We call M an *ϵ-prediction algorithm* (for $\mathcal{F}$ with respect to $\mathcal{P}$ and $\mathcal{D}$), and the function $m(n)$ is the *sample size* of M.

Note that in this definition, a smaller value of ϵ places a greater demand on the learning algorithm. We will want in particular to consider two extreme cases of predictability as follows:

Definition 2. Let $\mathcal{F}$ be a class of boolean functions, $\mathcal{P}$ a distribution ensemble over $\mathcal{F}$, and $\mathcal{D}$ a distribution ensemble over $\{0,1\}^*$. We say that $\mathcal{F}$ is *weakly predictable on average with respect to $\mathcal{P}$ and $\mathcal{D}$* if there exists some fixed polynomial $q(n)$ such that $\mathcal{F}$ is $(1/2 - 1/q(n))$-predictable on average with respect to $\mathcal{P}$ and $\mathcal{D}$. We say that $\mathcal{F}$ is *strongly predictable on average with respect to $\mathcal{P}$ and $\mathcal{D}$* if for any polynomial $q(n)$, $\mathcal{F}$ is $1/q(n)$-predictable on average with respect to $\mathcal{P}$ and $\mathcal{D}$.

We will also consider these same learning models when the learning algorithm is provided with *membership queries*. Here the definitions of prediction on average remain unchanged, but in addition to random labeled examples, the learning algorithm may receive the value $f(x)$ on inputs x of its own choosing; the test input $\tilde{x}$ is drawn after all queries are made to prevent the algorithm from cheating by querying $f(\tilde{x})$.

2.2 Measuring the Complexity of Representation Schemes and Distributions

In order to quantify the complexity of our proposed cryptographic primitives, we need to define complexity measures for the representation scheme $(\mathcal{R}, \mathcal{E})$ and the distribution ensembles $\mathcal{P}$ and $\mathcal{D}$. As we have mentioned, we use uniform circuit

families to allow the most precise statements and to emphasize the preservation of parallelism in our constructions. We begin with the straightforward case of the input distribution ensemble $\mathcal{D}$.

Definition 3. Let $\mathcal{D}$ be a distribution ensemble over $\{0,1\}^*$, and let $D = \{D_n\}$ be a uniform circuit sequence, where D_n takes $d(n)$ input bits for some polynomial $d(n)$ and outputs n bits. We say that $\mathcal{D}$ is *generated by* D if for every n, the output distribution of D_n is exactly $\mathcal{D}_n$: that is, if we choose w uniformly at random from $\{0,1\}^{d(n)}$ then $D_n(w) \in \{0,1\}^n$ is distributed according to $\mathcal{D}_n$.

We wish to formulate a similar notion for the generation of the distribution $\mathcal{P}$. For our cryptographic constructions, given a fixed representation scheme $(\mathcal{R}, \mathcal{E})$, it will be easier to think of the function distribution $\mathcal{P}_n$ as being over the set $\mathcal{R}_n$ (rather than over function class $\mathcal{F}_n$ itself), which then implicitly defines a distribution over $\mathcal{F}_n$ under $\mathcal{E}_n$. In this case we can define what it means for $\mathcal{P}$ to be generated by a circuit sequence.

Definition 4. Let $\mathcal{P}$ be a distribution ensemble over $\mathcal{R}_n \subseteq \{0,1\}^{r(n)}$, and let $P = \{P_n\}$ be a uniform circuit sequence, where P_n takes $p(n)$ input bits and outputs $r(n)$ bits for polynomials $p(n)$ and $r(n)$. We say that $\mathcal{P}$ is *generated by* P if for every n, the output distribution of P_n is exactly $\mathcal{P}_n$: that is, if we choose w uniformly at random from $\{0,1\}^{p(n)}$ then $P_n(w) \in \mathcal{R}_n$ is distributed according to $\mathcal{P}_n$.

A distribution $\mathcal{P}$ over $\mathcal{R}$ that allows very rapid generation of function representations will not be especially useful to us if the representation of f makes it expensive to compute $f(x)$. Thus, we make the following definition:

Definition 5. Let $(\mathcal{R}, \mathcal{E})$ be an encoding scheme for $\mathcal{F}$ such that $\mathcal{R}_n \subseteq \{0,1\}^{r(n)}$, and let $E = \{E_n\}$ be a uniform circuit sequence, where E_n takes $r(n)+n$ input bits and outputs a single bit. We say that $(\mathcal{R}, \mathcal{E})$ can be *evaluated by* E if for every n, on inputs $\sigma_f \in \mathcal{R}_n$ such that $f = \mathcal{E}_n(\sigma_f) \in \mathcal{F}_n$ and $x \in \{0,1\}^n$, we have $E_n(\sigma_f, x) = f(x)$.

Finally, we will need to define circuit sequences that are formed from other circuit sequences in certain ways. If $C = \{C_n\}$ is a circuit sequence, we define $C^m = \{C_n^m\}$ to be the sequence of m-fold replication of the circuits in C. More precisely, the circuit C_n^m takes $m \cdot n$ inputs, and consists of m disjoint "copies" of C_n: on inputs $x_1, \ldots, x_m \in \{0,1\}^n$, the output $C_n^m(x_1, \ldots, x_m)$ is the concatenation of $C_n(x_1), \ldots, C_n(x_m)$.

If C and D are sequences of circuits, we define the sequence $C \circ D = \{C_n \circ D_n\}$ as follows: the circuit $C_n \circ D_n$ has $q(n)$ inputs for some polynomial $q(n)$. Some or all of these inputs feed the fixed circuit D_n, whose outputs (along with some or all of the inputs) in turn feed the fixed circuit C_n. (Note that $C_n \circ D_n$ is technically a *set* of circuits since we have not specified exactly how the inputs are wired.) Similarly, if C,D and E are sequences of circuits, the sequence $C \circ (D, E)$ is that in which the circuit $C_n \circ (D_n, E_n)$ has some or all of its inputs fed to the

fixed circuits D_n and E_n in parallel, whose outputs (along with some or all of the inputs) in turn feed the fixed circuit C_n. So, replication increases the circuit width, and the composition operation $\circ$ increases circuit depth.

2.3 Discussion

Several widely studied function classes, such as DNF formulas, are believed to be hard to learn even when the distribution $\mathcal{D}$ is uniform. There has been less *formal* work studying what hard function distributions look like because distribution $\mathcal{P}$ is not used in the standard models. The following, however, is a distribution on DNF formulas that seems to defy all known methods of attack, and we believe that any method that could even weakly predict such functions over a uniform $\mathcal{D}$ would require profoundly new ideas.

Select at random two disjoint sets $A, B \subset \{1, \ldots, n\}$ each of size $\log n$. On input x, compute the parity of the bits indexed by A and the majority function of the bits indexed by B, and output the exclusive-or of the two results. This function can be represented by a polynomial size DNF formula (or decision tree) since the truth table has only $2^{2 \log n} = n^2$ entries. Because this $\mathcal{P}$ distribution seems hard even to weakly predict over uniform $\mathcal{D}$, it could be used for our bit generator in Section 3.1 (though learning becomes easy if membership queries are allowed, so the generator of Section 3.2 does not apply).

3 General Results

3.1 A Bit Generator Based on Hardness for Weak Prediction

We begin by showing that a function class hard to weakly predict on average can be used to create a CSPRBG whose circuit depth is comparable to that of the function class plus that of generating the hard distributions.

Definition 6. A *cryptographically strong pseudorandom bit generator (CSPRBG)* is a uniform circuit sequence $\mathcal{G} = \{\mathcal{G}_n\}$, where $\mathcal{G}_n$ takes n bits of input and produces $g(n) > n$ bits of output, with the following property: for any polynomial time algorithm T that produces a boolean output and any polynomial $q(n)$, there exists an n_0 such that for all $n \geq n_0$,

$$\left| \Pr_{y \in \{0,1\}^{g(n)}}[T(y) = 1] - \Pr_{x \in \{0,1\}^n}[T(\mathcal{G}_n(x)) = 1] \right| \leq 1/q(n).$$

We call the function $e(n) = g(n) - n$ the *expansion* of $\mathcal{G}$.

We first consider the special case in which the input distribution ensemble is uniform.

Theorem 7. *Let $\mathcal{F}$ be a class of boolean functions, $(\mathcal{R}, \mathcal{E})$ a representation scheme for $\mathcal{F}$, $\mathcal{P}$ a distribution ensemble over $\mathcal{R}$, and $\mathcal{U}$ the uniform distribution ensemble over $\{0,1\}^*$. Let $(\mathcal{R}, \mathcal{E})$ be evaluated by the circuit sequence E,*

and let $\mathcal{P}$ be generated by the uniform circuit sequence P. Then if $\mathcal{F}$ is not weakly predictable on average with respect to $\mathcal{P}$ and $\mathcal{U}$, there is a CSPRBG computed by the uniform circuit sequence $E^{m(n)} \circ P$ for some fixed polynomial $m(n)$.

Proof. (Sketch) Informally, for any n, the bit generator $\mathcal{G}_n$ behaves as follows: it takes as input a random bit string, and uses some of these bits to generate a function $f \in \mathcal{F}_n$ according to the distribution $\mathcal{P}_n$. The rest of the input bits are used directly as m inputs, $x_1, \ldots, x_m \in \{0,1\}^n$ for f. The output of the generator consists of $x_1, \ldots, x_m$ followed by the m bits $f(x_1), \ldots, f(x_m)$.

More formally, $\mathcal{G}_n$ takes as input a random string of $p(n) + m \cdot n$ bits. Here $p(n)$ is the number of random input bits required by the circuit P_n for generating a representation $\sigma_f \in \mathcal{R}_n$ of a function $f = \mathcal{E}_n(\sigma_f) \in \mathcal{F}_n$, where σ_f is distributed according to $\mathcal{P}$; m will be determined by the analysis. The generator feeds the first $p(n)$ input bits into P_n to obtain σ_f. The remaining $m \cdot n$ random input bits are regarded as m random vectors $x_1, \cdots, x_m \in \{0,1\}^n$. For each i the generator then feeds σ_f and x_i to a parallel copy of E_n to obtain $E_n(\sigma_f, x_i) = f(x_i)$. The output of the generator is then $x_1, \ldots, x_m, f(x_1), \ldots, f(x_m)$. It is easy to verify that $\mathcal{G}$ is computed by the circuit sequence $E^m \circ P$. Since our generator produces $g(n) = m \cdot n + m$ output bits, we obtain expansion provided that $m > p(n)$.

We now argue that $\mathcal{G}$ is in fact a CSPRBG. For contradiction, suppose that $\mathcal{G}$ is not, and let T be a polynomial time algorithm such that

$$\left| \Pr_{y \in \{0,1\}^{g(n)}}[T(y) = 1] - \Pr_{x \in \{0,1\}^n}[T(\mathcal{G}(x)) = 1] \right| \geq 1/q(n)$$

for some polynomial $q(n)$. For each $0 \leq i \leq m$, let t_i denote the probability that T outputs 1 when its first $i + mn$ input bits are the first $i + mn$ bits of $\mathcal{G}(x)$ on random x, and the remaining input bits of T are truly random. Note that by definition of $\mathcal{G}_n$, when $i = 0$ the input to T is truly random. So we have $|t_0 - t_m| \geq 1/q(n)$, and by a standard "probability walk" argument there must be an $0 \leq i < m$ such that $|t_i - t_{i+1}| \geq 1/(m \cdot q(n))$. Furthermore, we can find such an i (approximately) with high probability by performing repeated experiments with T using random draws from $\mathcal{P}_n$ and $\mathcal{D}_n$, and once such an i is found we can "center the bias" to produce an efficient algorithm T' such that $t'_i \geq 1/2 + 1/q'(n)$ and $t'_{i+1} \leq 1/2 - 1/q'(n)$ for some polynomial $q(n)$.

Algorithm T' thus has the following property: suppose we draw a function f randomly according to $\mathcal{P}_n$, and draw m random n-bit vectors $x_1, \ldots, x_m$ and we give T' the inputs $x_1, \ldots, x_m$ along with $f(x_1), \ldots, f(x_i)$, followed by an input bit b_{i+1} that is *either* $f(x_{i+1})$ or a truly random bit, followed by $g(n) - i - 1$ random bits. Then T' can determine with probability significantly better than random guessing whether its $i+1$st input bit b_{i+1} is $f(x_{i+1})$ or a truly random bit; we interpret an output of 1 as a guess that b_{i+1} is random, and an output of 0 as a guess that $b_{i+1} = f(x_{i+1})$.

Now suppose we have access to random examples according to $\mathcal{U}_n$ of a target function f drawn according to $\mathcal{P}_n$, and we also have a random test input $\tilde{x}$. Suppose we give to T' the random strings $x_1, \ldots, x_i, \tilde{x}, x_{i+2}, \ldots, x_m$ followed

by $f(x_1), \ldots, f(x_i)$, followed by an undetermined input bit b_{i+1}, followed by $g(n)-i-1$ random bits. Then it is easy to show by a simple averaging argument that the following strategy yields an algorithm for weakly predicting $\mathcal{F}$ with respect to $\mathcal{P}$ and $\mathcal{U}$: with the other inputs as specified, we run T' both with $b_{i+1} = 0$ and $b_{i+1} = 1$. If both inputs cause an output of 0, or both inputs cause an output of 1, we flip a coin to predict $f(\tilde{x})$. Otherwise, we predict that $f(\tilde{x})$ is the value of b_{i+1} that caused T' to output 0. □

The following simple lemma relates the hardness of learning a function class with respect to *some* input distribution ensemble to the hardness of learning a related function class with respect to the uniform distribution ensemble.

Lemma 8. *Let $\mathcal{F}$ be a class of boolean functions, $\mathcal{P}$ a distribution ensemble over $\mathcal{F}$, and $\mathcal{D}$ a distribution ensemble over $\{0,1\}^*$. Let the ensemble $\mathcal{D}$ be generated by the circuit sequence D, and let $\mathcal{F} \circ D$ denote the class of functions obtainable by composing a function in $\mathcal{F}_n$ with the circuit D_n. Then for any ϵ, if $\mathcal{F}$ is not ϵ-predictable on average with respect to $\mathcal{P}$ and $\mathcal{D}$, then $\mathcal{F} \circ D$ is not ϵ-predictable on average with respect to $\mathcal{P}$ and the uniform ensemble $\mathcal{U}$ over $\{0,1\}^*$.*

Proof. Immediate; we are simply letting the computation of the hard distribution ensemble $\mathcal{D}$ be part of the target function. □

From Theorem 7 and Lemma 8, we can now easily obtain a bit generator from a learning problem that is hard with respect to some input distribution ensemble.

Corollary 9. *Let $\mathcal{F}$ be a class of boolean functions, $(\mathcal{R}, \mathcal{E})$ a representation scheme for $\mathcal{F}$, $\mathcal{P}$ a distribution ensemble over $\mathcal{R}$, and $\mathcal{D}$ a distribution ensemble over $\{0,1\}^*$. Let $(\mathcal{R}, \mathcal{E})$ be evaluated by the circuit sequence E, let $\mathcal{P}$ be uniformly generated by the circuit sequence P, and let $\mathcal{D}$ be uniformly generated by the circuit sequence D. Then if $\mathcal{F}$ is not weakly predictable on average with respect to $\mathcal{P}$ and $\mathcal{D}$, there is a CSPRBG computed by the uniform circuit sequence $(E \circ D)^{m(n)} \circ P$ for some fixed polynomial $m(n)$.*

3.2 Improved Expansion via Nisan-Wigderson

The pseudorandom generator just described takes $p(n) + m \cdot n$ truly random input bits to $m \cdot n + m$ output bits, giving expansion $e(n) = m - p(n)$. While we can let $e(n)$ attain any desired value by choosing $m = m(n)$ as large as necessary, the *expansion ratio* (the number of output bits divided by the number of input bits) will not exceed $1 + 1/n$. There are standard methods which can be used to amplify the expansion of any generator. However, these methods iterate the generator and therefore significantly increase the resulting circuit depth.

By applying a result due to Nisan and Wigderson [12] we can improve our generator to obtain a much greater expansion ratio without a correspondingly large increase in circuit depth (or size). However, in order to prove security, we will need to assume that the class of functions is hard to weakly predict on average even when the learning algorithm is provided with membership queries.

We now describe how this stronger intractability assumption allows us to modify our generator to obtain an expansion ratio on the order of n^2 rather than just $O(1)$ as before.

The new CSPRBG $\mathcal{G}_n$ takes as input a random string of $p(n) + n^2$ bits. As before, $p(n)$ is the number of random input bits required by the circuit P_n. (For simplicity we assume that the hard distribution $\mathcal{D}_n$ over the inputs $\{0,1\}^n$ is the uniform distribution; similar improvements can be given for the general case as was done above.) Call the additional n^2 input bits $v = v_1, \ldots, v_{n^2}$. It is described by Nisan and Wigderson [12] how to uniformly construct a family of n^4 sets $S_1, \ldots, S_{n^4}$ such that: (1) $S_i \subset \{v_1, \ldots, v_{n^2}\}$; (2) $|S_i| = n$ for all i; and (3) $|S_i \bigcap S_j| \leq \log n$ for all $i \neq j$.

Our new generator will work as follows: as before, the first $p(n)$ bits are used to generate a function $f \in \mathcal{F}_n$ according to the distribution $\mathcal{P}_n$, and the remaining n^2 bits v are copied to the first n^2 output bits. Now, however, if we let f_i denote the function f applied to the subset of $v_1, \ldots, v_{n^2}$ indicated by the set S_i, the (n^2+i)th output bit is $f_i(v)$. Thus, $\mathcal{G}_n$ takes $p(n)+n^2$ input bits and gives $g(n) = n^2 + n^4$ output bits, for an expansion ratio of $\Omega(n^2)$. Note that the sets S_i are fixed as part of the generator description and thus are known to any potential adversary.

We now sketch the argument that if $\mathcal{F}$ is not weakly predictable on average even with membership queries, then $\mathcal{G}_n$ is a CSPRBG. To see this, suppose the contrary that $\mathcal{G}_n$ is not a CSPRBG. By standard arguments, there is an $1 \leq i \leq n^4$ and a polynomial time algorithm T such that

$$\Pr_{f \in \mathcal{P}_n, v \in \{0,1\}^{n^2}}[T(v, f_1(v), \ldots, f_i(v)) = f_{i+1}(v)]$$

exceeds $1/2 + 1/q(n)$ for some polynomial $q(n)$.

The function f_{i+1} only depends on the bits in S_{i+1} which without loss of generality we call $v_1, \ldots, v_n$. By an averaging argument we can find a *fixed* setting $z = z_{n+1}, \ldots, z_{n^2} \in \{0,1\}^{n^2-n}$ of the remaining bits $v_{n+1}, \ldots, v_{n^2}$ such that

$$\Pr_{f \in \mathcal{P}_n, v \in \{0,1\}^n}[T(vz, f_1(v,z), \ldots, f_i(v,z)) = f_{i+1}(v,z)]$$

exceeds $1/2 + 1/q(n)$ for some polynomial $q(n)$.

Now we describe how a learner who can make membership queries can weakly predict $\mathcal{F}$ on average. By the Nisan-Wigderson construction, each f_j other than f_{i+1} is actually a function of only $\log n$ bits in $v_1, \ldots, v_n$ (all other bits have been fixed) and we know which $\log n$ bits since the sets S_i are fixed as part of the generator description. Thus the entire truth table of each f_j is of size polynomial in n, and can be determined by making only n membership queries to f (the queries simply let the variables in $S_j \bigcap \{v_1, \ldots, v_n\}$ assume all n possible settings while the remaining variables have their values fixed according to z). To predict $f(v)$ on a challenge input v the learner looks up the values of $f_j(vz)$ for $1 \leq j \leq i$ and then outputs $T(vz, f_1(vz), \ldots, f_i(vz))$. This weakly predicts $f_{i+1}(vz) = f(v)$, and by contradiction proves our assertion that $\mathcal{G}_n$ is a CSPRBG.

3.3 A One-Way Function Based on Hardness for Strong Prediction

We now show that under the weaker assumption that a class of functions is hard to strongly predict on average, we can construct a one-way function whose circuit depth is comparable to that of the function class plus that of the hard distributions. A related result is given by Impagliazzo and Levin [6].

Definition 10. Let $F = \{F_n\}$ be a uniform sequence of circuits $F_n : \{0,1\}^n \to \{0,1\}^{s(n)}$ for some polynomial $s(n)$. We say that F is a *one-way function* if there exists a polynomial $q(n)$ such that for any polynomial time algorithm T, there exists an n_0 such that for all $n \geq n_0$,

$$\Pr_{x \in \{0,1\}^n}[F_n(T(F_n(x))) \neq F_n(x)] \geq 1/q(n).$$

Theorem 11. *Let $\mathcal{F}$ be a class of boolean functions, $(\mathcal{R}, \mathcal{E})$ a representation scheme for $\mathcal{F}$ ($\mathcal{R}_n \subseteq \{0,1\}^{r(n)}$), $\mathcal{P}$ a distribution ensemble over $\mathcal{R}$, and $\mathcal{D}$ a distribution ensemble over $\{0,1\}^*$. Let $(\mathcal{R}, \mathcal{E})$ be evaluated by the circuit sequence E, $\mathcal{P}$ be generated by the circuit sequence P, and $\mathcal{D}$ be generated by the circuit sequence D. Then if $\mathcal{F}$ is not strongly predictable on average with respect to $\mathcal{P}$ and $\mathcal{D}$, there is a one-way function F computed by the uniform circuit sequence $E^{m(n)} \circ (P, D^{m(n)})$ for some fixed polynomial $m(n)$.*

Proof. (Sketch) The construction is similar to that of the bit generator; the main differences are the polynomial $m(n)$ and the analysis. The input to the one-way function F_n will consist of $p(n) + m \cdot d(n)$ bits, where $p(n)$ is the number of inputs to P_n and $d(n)$ is the number of inputs to D_n. The first $p(n)$ bits are fed to P_n to produce a representation $\sigma_f \in \mathcal{R}_n$ of a function $f \in \mathcal{F}_n$. The remaining $m \cdot d(n)$ input bits are regarded as m blocks $w_1, \ldots, w_m \in \{0,1\}^{d(n)}$. Each w_i is fed to a parallel copy of D_n in order to produce an $x_i \in \{0,1\}^n$; if the w_i are selected randomly, then the x_i are distributed according to $\mathcal{D}_n$. Finally, each x_i is given along with σ_f to a parallel copy of E_n in order to obtain $E_n(\sigma_f, x_i) = f(x_i)$. The output of F_n is then $x_1, \ldots, x_m$ followed by $f(x_1), \ldots, f(x_m)$.

We begin the analysis by noting that if $v, v' \in \{0,1\}^{p(n)}$ and $w_1, \ldots, w_m$, $w'_1, \ldots, w'_m \in \{0,1\}^{d(n)}$ are such that

$$F_n(v, w_1, \ldots, w_m) = F_n(v', w'_1, \ldots, w'_m)$$

we must have $D_n(w_i) = D_n(w'_i)$ for all i. Let $x_i = D_n(w_i)$. Now although it is not necessarily true that $v = v'$, if we let $\sigma_f = P_n(v)$ and $\sigma_{f'} = P_n(v')$ be the representations in $\mathcal{R}_n$ of the functions $f, f' \in \mathcal{F}_n$, by construction of F_n it must be the case that $f(x_i) = f'(x_i)$ for all i.

Let $q(n)$ be such that $\mathcal{F}$ is not $3/q(n)$-predictable (with respect to $\mathcal{P}$ and $\mathcal{D}$). For any fixed $f \in \mathcal{F}_n$, the probability over m random examples from $\mathcal{D}_n$ that there exists a function $f' \in \mathcal{F}_n$ agreeing with f on those examples, but that has error greater than $1/q(n)$ with respect to f and $\mathcal{D}$, is at most $|\mathcal{F}_n|(1 - 1/q(n))^m$. This probability is smaller than $1/q(n)$ for $m = \Omega(q(n)[\log|\mathcal{F}_n| + \log q(n)])$,

(which is polynomial in n since $\log|\mathcal{F}_n| \leq r(n)$). We set m such that this is the case.

Thus, suppose that F is not a one-way function. Then there exists an algorithm T with probability at least $1-1/q(n)$ of finding an inverse for $F_n(v, w_1, \ldots, w_m)$ on random inputs. So given a test input $\tilde{x} \in \mathcal{D}_n$ and m examples $\langle x_i, f(x_i)\rangle$ of f drawn according to $\mathcal{D}_n$, we can simply give the string $x_1, \ldots, x_m, f(x_1), \ldots, f(x_m)$ to T. There is then probability at least $1-1/q(n)$ that T returns $v', w'_1, \ldots, w'_m$ such that $F_n(v', w'_1, \ldots, w'_m) = x_1, \ldots, x_m, f(x_1), \ldots, f(x_m)$. If this occurs, then by the argument above there is probability at least $1 - 1/q(n)$ that $P_n(v')$ represents a function f' with error at most $1/q(n)$, and we can simply compute $f'(\tilde{x})$. The probability that $f'(\tilde{x}) = f(\tilde{x})$ is at least $1 - 3/q(n)$, violating the assumption that $\mathcal{F}$ is not $3/q(n)$-predictable. □

3.4 A Private-Key Cryptosystem Based on Hardness for Weak Prediction

In section we describe a simple and natural mapping from hard learning problems to private-key cryptosystems. This mapping has the property that if a simple function representation (for instance, DNF formulas) is hard to learn over a simple distribution (for instance, uniform) then encrypting and decrypting are both easy. Define the function $E_{n,\sigma}(x) = E_n(\sigma, x)$. For example, if $(\mathcal{R}, \mathcal{E})$ is a representation for DNF formulas, then E_n is a circuit that takes a string representing a DNF formula f and some x, and produces $f(x)$ as output. $E_{n,\sigma}$, however, is just a depth-2 circuit.

A *private-key cryptosystem* is a tuple $(\mathbf{G}, \mathbf{E}, \mathbf{D})$ of three probabilistic polynomial time algorithms. [4] The key generator $\mathbf{G}$ takes as input 1^n, and outputs a key k of length n. The encryption algorithm $\mathbf{E}$ takes as input a message m and a key k, and produces ciphertext as output. The decryption algorithm $\mathbf{D}$ takes as input ciphertext and a key and produces a message. We require that $\mathbf{D}(\mathbf{E}(m, k), k) = m$.

The private-key cryptosystems we will discuss are probabilistic schemes that encrypt one bit at a time, encrypting each bit independently from the previous ones, as in [5]. We use standard notions of chosen plaintext and/or ciphertext attack. Since we encrypt bits independently, we may say a system is secure if for any polynomial time algorithm T, polynomial $q(n)$, and sufficiently large n, the probability that after performing its allowed type of attack T correctly guesses the decryption of an encrypted random bit is less than $1/2 + 1/q(n)$.

Suppose $\mathcal{F}$ is a class of boolean functions, $(\mathcal{R}, \mathcal{E})$ a representation scheme for $\mathcal{F}$, $\mathcal{P}$ a distribution ensemble over $\mathcal{R}$, and $\mathcal{D}$ a distribution ensemble over $\{0,1\}^*$. In addition, suppose $(\mathcal{R}, \mathcal{E})$ is evaluated by the uniform circuit sequence E, and $\mathcal{P}$ and $\mathcal{D}$ are generated by the uniform circuit sequences P and D respectively. Assume that the probability a random example from $\mathcal{D}_n$ is positive for a random

[4] Here we depart from our policy of describing primitives as uniform circuit families since we intend to describe the private-key system informally.

function f from $\mathcal{P}_n$ is in the range $[\frac{1}{2} - \frac{1}{n}, \frac{1}{2} + \frac{1}{n}]$ (if this is not the case then $\mathcal{F}$ is weakly predictable).

The cryptosystem based on $\mathcal{F}$, $\mathcal{P}$, and $\mathcal{D}$ is as follows. The key generator $\mathbf{G}$ uses input 1^n to generate P_n. $\mathbf{G}$ then feeds $p(n)$ random bits into P_n to produce a string $\sigma \in \mathcal{R}_n$. String σ is the key given to the encryption and decryption algorithms (so, technically, the security parameter is $r(n)$). Let f be the function represented by σ.

The encryption algorithm $\mathbf{E}$ begins by generating circuits D_n and E_n, and evaluating E_n on the private key to create $E_{n,\sigma}$. It encrypts a 1 by sending a random (according to $\mathcal{D}$) positive example of f and encrypts a 0 by sending a random (according to $\mathcal{D}$) negative example of f. This requires running D_n on $d(n)$ random bits to create an example x, and then computing $E_{n,\sigma}(x)$ to see if the example is of the appropriate type, repeating the procedure if it is not. Notice that the expected number of calls to D_n and $E_{n,\sigma}$ is just $2 + o(1)$ by our assumption on $\mathcal{D}$. (This could be improved by encrypting many bits at a time).

Decryption is even simpler than encryption. The decryption algorithm $\mathbf{D}$ begins by generating $E_{n,\sigma}$, and then decrypts strings x by computing $E_{n,\sigma}(x)$.

Theorem 12. *If $\mathcal{F}$ is not weakly predictable on average with respect to $\mathcal{P}$ and $\mathcal{D}$, then the cryptosystem $(\mathbf{G}, \mathbf{E}, \mathbf{D})$ described above is secure against chosen plaintext attack. If furthermore $\mathcal{F}$ is not weakly predictable with membership queries with respect to $\mathcal{P}$ and $\mathcal{D}$, then $(\mathbf{G}, \mathbf{E}, \mathbf{D})$ is secure against chosen plaintext and ciphertext attack.*

Proof. (Sketch) Suppose algorithm M can break $(\mathbf{G}, \mathbf{E}, \mathbf{D})$ with chosen plaintext attack for infinitely many n, and asks for the encryption of $m(n)$ plaintext bits. The learning algorithm simply requests $3m(n)$ labeled examples (which with high probability will result in at least $m(n)$ positive examples and at least $m(n)$ negative examples) and uses them to simulate $\mathbf{E}$ for M's plaintext queries. It then feeds the test input $\tilde{x}$ to M and uses M's response as the prediction. Since $r(n)$ is polynomial in n, and M responds correctly with probability at least $1/2 + 1/poly(r(n))$, this will be a weak prediction algorithm (for $\mathcal{F}$ with respect to $\mathcal{P}$ and $\mathcal{D}$).

If M makes chosen ciphertext queries, the learning algorithm, if it is allowed membership queries, can answer these in the obvious way as membership and ciphertext queries are equivalent here. Thus, if $(\mathbf{G}, \mathbf{E}, \mathbf{D})$ is vulnerable to chosen plaintext and ciphertext attack, then $\mathcal{F}$ is weakly predictable with membership queries with respect to $\mathcal{P}$ and $\mathcal{D}$. □

4 A Bit Generator Based on Parity Functions with Noise

Let $\mathcal{S}_n$ denote the set of all parity functions over $\{0,1\}^n$; specifically, for each of the 2^n subsets $S = \{x_{i_1}, \ldots, x_{i_k}\} \subseteq \{x_1, \ldots, x_n\}$ there is a function $f_S \in \mathcal{S}_n$ defined by

$$f_S(x_1, \ldots, x_n) = x_{i_1} \oplus \cdots \oplus x_{i_k}.$$

In this section we describe a simple bit generator whose security is based on the assumption that parity functions are hard to learn *in the presence of classification noise*. This means that we add a parameter $0 < \eta < 1/2$ to our learning model called the *noise rate*, and now a learning algorithm, rather than always receiving a labeled example $\langle x, f(x)\rangle$ of the target function f, will instead receive a *noisy* labeled example $\langle x, \ell\rangle$. Here $\ell = f(x)$ with probability $1 - \eta$ and $\ell = \neg f(x)$ with probability η where this choice is made independently for each requested example. Our specific hardness assumption is as follows.

The Parity Assumption: For some fixed constant $0 < \eta < 1/2$, there is *no* algorithm taking δ and n as input that runs in time polynomial in $1/\delta$ and n, and for infinitely many n does the following: Given access to noisy examples from the uniform distribution $\mathcal{U}_n$ on $\{0,1\}^n$ classified by an arbitrary $f_S \in \mathcal{S}_n$ with noise rate η, it produces the set S with probability at least $1 - \delta$.

Note that in comparison with our definitions in earlier sections, we have decreased the strength of our assumption in a number of ways. First, on those values of n for which a learning algorithm "succeeds", it must succeed for *every* function in $\mathcal{S}_n$. We have thus eliminated the distribution $\mathcal{P}_n$. Second, the learning algorithm must now actually find the target concept with arbitrarily high confidence. Third, the learning algorithm must learn with a possibly large rate of noise in the labels.

The parity assumption is closely related to the problem of efficiently decoding random linear codes, which is a long-standing open problem. It is known that the problem of finding the parity function that minimizes the number of disagreements with an input set of labeled examples is $\mathcal{NP}$-hard [2] (and easy to show it is MAX-SNP hard), and this optimization problem was used by McEliece [11] as the core of a proposed public-key cryptosystem with informal security arguments in which the matrix of examples must be carefully chosen. Recent results [8] provide some evidence in favor of the parity assumption by proving that parity functions cannot be learned using a certain class of statistical algorithms that include all known noise-tolerant learning algorithms in the Valiant model.

Based on the (unproven) parity assumption, we propose a simple pseudorandom bit generator. This generator is quite similar to a proposed one-way function due to Goldreich, Krawczyk and Luby [3], who then obtain a generator by running the one-way function through a generic transformation. Essentially, our contribution is to prove that the output of this one-way function is already pseudorandom. This stronger assertion is apparently already known to some researchers in the cryptography community as a "folk theorem".

The input seed to the generator $\mathcal{G}$ consists of $s(n) = n + m \cdot n + \mathcal{H}(\eta) \cdot m$ random bits, regarded as a block s_f of n bits, followed by m blocks $x_1, \ldots, x_m$ of n bits each, followed by a block s_r of $\mathcal{H}(\eta) \cdot m$ bits. Here m will be determined by the analysis, and $\mathcal{H}(\eta)$ is the binary entropy of the noise rate η in the parity assumption.

The block s_f encodes a parity function $f_S \in \mathcal{S}_n$; each 1 in s_f indicates a variable included in the subset S. As in our previous generator, $x_1, \ldots, x_m$ are regarded as inputs to f_S. The block s_r encodes a *longer* m-bit noise vector s'_r,

where s'_r has exactly $\lfloor \eta \cdot m \rfloor$ 1's. Such an encoding can be shown to require s_r to be of length only $\mathcal{H}(\eta) \cdot m$, and this encoding can be done in a number of standard ways. Thus, if we randomly choose s_r among all $\mathcal{H}(\eta) \cdot m$-bit vectors, then we randomly choose s'_r from among all m-bit vectors with exactly $\lfloor \eta \cdot m \rfloor$ 1's.

$\mathcal{G}$ works as follows: it first uses s_f to obtain the represented parity function f_S. It then expands s_r to obtain s'_r. It next computes $f(x_1), \ldots, f(x_m)$, and for each i lets $\ell_i = f(x_i)$ if the ith bit of s'_r is 0, and lets $\ell_i = \neg f(x_i)$ if the ith bit of s'_r is 1. The output of $\mathcal{G}$ is $x_1, \ldots, x_m$, followed by $\ell_1, \ldots, \ell_m$. Note that the output of $\mathcal{G}$ is a noisy sample of f_S, but where the number of noisy labels is *exactly* $\lfloor \eta \cdot m \rfloor$, rather than determined by m flips of a coin of bias η as in the parity assumption; this discrepancy will be dealt with in the proof of the coming theorem.

$\mathcal{G}$ takes $s(n) = n + m \cdot n + \mathcal{H}(\eta) \cdot m$ random bits as input, and outputs $t(n) = m \cdot n + m$ bits, for expansion $t(n) - s(n) = m(1 - \mathcal{H}(\eta)) - n$. Thus we get expansion provided we choose $m > (1/(1 - \mathcal{H}(\eta)) \cdot n$, which can be done so long as $\eta < 1/2$.

Theorem 13. *Under the parity assumption, $\mathcal{G}$ is a pseudorandom bit generator.*

Proof. (Sketch) The overall strategy is to show that the parity assumption in fact implies the stronger assumption (the *strong parity assumption*) that parity functions are not even weakly predictable on average (in the presence of noise rate η) with respect to the uniform distribution $\mathcal{P}_n$ over $\mathcal{S}_n$ and the uniform distribution $\mathcal{U}_n$ over $\{0,1\}^n$. The security of $\mathcal{G}$ can then be shown from arguments similar to that for the general bit generator outlined earlier. We must additionally show that the difficulty of learning with noise rate η implies the difficulty of learning when the m examples requested by the learning algorithm contain *exactly* $\lfloor \eta \cdot m \rfloor$ errors.

To see that the parity assumption implies the strong parity assumption, consider an algorithm A that contradicts the latter. To break the parity assumption, we first randomize S by XORing the labels given to examples with the classification given by some *random* parity function S'. Given such examples, A will perform noticeably better than guessing on a test example $\tilde{x}$. However, instead of giving to A these examples, we give to A examples in which we have replaced the ith bit x_i by a random bit in each. If x_i was irrelevant to the classification (not in $S \Delta S'$) the resulting distribution is indistinguishable from the original one, and A will still do better than random guessing. However, if $x_i \in S \Delta S'$, then there is now no correlation between the examples and the labels (the noise does not affect this) and so A cannot do better than random guessing. Thus, by repeating this process we can determine which variables are relevant to the classification and recover S, breaking the parity assumption.

Finally, we must address the fact that our generator is always injecting a fixed fraction of label errors rather than using a coin of bias η. Suppose that the learning problem became easy provided that the noise model always injected *exactly* $\lfloor \eta \cdot m \rfloor$ errors into any sample of size m requested by the learning algorithm.

We could then run this "fixed-fraction" algorithm many times on its requested sample size m, where the m examples come from a source with probability η of noise independently on each example. The probability that exactly $\eta \cdot m$ labels are noisy is certainly $\Omega(1/m)$ regardless of the value of η. So one of the runs of the fixed-fraction algorithm must succeed with high probability, and we can determine which run by hypothesis testing. □

Acknowledgements

We are grateful to Russell Impagliazzo and Steven Rudich for many insightful comments and suggestions on this research.

References

1. Dana Angluin and Michael Kharitonov. When won't membership queries help? In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*, pages 444–454, May 1991.
2. E. Berlekamp, R. McEliece, and H. van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24, 1978.
3. O. Goldreich, H. Krawczyk, and M. Luby. On the existence of pseudorandom generators. In *29th Annual Symposium on Foundations of Computer Science*, pages 12–21, October 1988.
4. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.
5. Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *JCSS*, 28(2):270–299, April 1984.
6. R. Impagliazzo and L. Levin. No better ways to generate hard NP instances than picking uniformly at random. In *31st Annual Symposium on Foundations of Computer Science*, October 1990.
7. R. Impagliazzo, L. Levin, and M. Luby. Pseudorandom generation from one-way functions. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, May 1989.
8. Michael Kearns. Efficient noise-tolerant learning from statistical queries. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing*, May 1993.
9. Michael Kearns and Leslie G. Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 433–444, May 1989. To appear, *Journal of the Association for Computing Machinery*.
10. M. Kharitonov. Cryptographic hardness of distribution-specific learning. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing*, May 1993.
11. R. J. McEliece. *A Public-Key System Based on Algebraic Coding Theory*, pages 114–116. Jet Propulsion Lab, 1978. DSN Progress Report 44.
12. N. Nisan and A. Wigderson. Hardness vs. randomness. In *29th Annual Symposium on Foundations of Computer Science*, pages 2–12, October 1988.
13. L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.
14. Andrew C. Yao. Theory and applications of trapdoor functions. In *23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, 1982.

This article was processed using the LaTeX macro package with LLNCS style