Pseudo-Random Generators for All Hardnesses

Christopher Umans
Microsoft Research
One Microsoft Way
Redmond, WA 98052
umans@microsoft.com

ABSTRACT

We construct the first pseudo-random generators with logarithmic seed length that convert s bits of hardness into $s^{\Omega(1)}$ bits of 2-sided pseudo-randomness *for any s*. This improves [8] and gives a direct proof of the optimal hardness vs. randomness tradeoff in [15]. A key element in our construction is an augmentation of the standard low-degree extension encoding that exploits the field structure of the underlying space in a new way.

1. INTRODUCTION

One of the most intriguing and active areas in complexity theory concerns the power of randomized algorithms. It is now known that BPP (randomized polynomial-time with two-sided error) and RP (randomized polynomial-time with one-sided error) coincide with P *under the plausible assumption* that some language in E requires exponential-size circuits. In other words, within the context of polynomial running time, randomness probably does not help. This remarkable conclusion is the result of a long line of research focused on "hardness vs. randomness" tradeoffs, culminating with [9] (with subsequent improvements and simplifications in [17, 7, 8, 15]). In general, given a hard language L in E, these tradeoffs give deterministic simulations of BPP whose running time depends inversely on the circuit complexity of L.

The rough outlines of the derandomization picture – namely, the hardness assumption under which BPP = P – have been known for several years now. Only recently have the finer details been filled in giving the "correct" quantitative tradeoff (up to a polynomial slowdown) between hardness and randomness, stated in the following theorem of [15]:

THEOREM 1 ([15]). *If there exists a function family $f = \{f_n\}$ in E with circuit complexity at least $s(n)$, then*

$$BPTIME(\ell) \subseteq DTIME(2^{O(s^{-1}(\ell^{O(1)}))}).$$

There are currently two potential routes to proving Theorem 1. The less direct route comprises two steps: first, one constructs a *hitting-set generator* (HSG) that converts s bits of hardness into $s^{\Omega(1)}$ bits of 1-*sided* pseudo-randomness. This was the route taken by [15]; we give one necessary definition and the result below:

DEFINITION 1. *(ϵ-HSG) An ϵ-*HSG for size s *is a function $H : \{0,1\}^t \to \{0,1\}^m$ such that for all circuits C of size at most s whose acceptance probability is at least ϵ:*

$$\Pr_z[C(H(z)) = 1] > 0. \quad (1)$$

THEOREM 2 ([15]). *Given a function $f : \{0,1\}^{\log n} \to \{0,1\}$ with circuit complexity at least s, one can construct an $1/m$-HSG $H : \{0,1\}^{O(\log n)} \to \{0,1\}^m$ for size m circuits with $m = s^{\Omega(1)}$; moreover, H can be computed in $n^{O(1)}$ time with oracle access to f.*

The second step utilizes a result of [1] (see [3] for a simpler proof) stating that HSGs suffice to derandomize BPP; Theorem 1 then follows.

An alternative, more direct route entails building a *pseudo-random generator* (PRG) that converts s bits of hardness into $s^{\Omega(1)}$ bits of *2-sided* pseudo-randomness. Motivated in part by the connection between extractors and PRGs discovered by Trevisan [19], considerable attention has recently been focused on attaining such a construction [17, 7, 8], so far without success. The best known construction prior to this paper is due to [8]; we give one necessary definition and the result below:

DEFINITION 2. *(ϵ-PRG) An ϵ-*PRG for size s *is a function $G : \{0,1\}^t \to \{0,1\}^m$ such that for all circuits C of size at most s:*

$$|\Pr_z[C(G(z)) = 1] - \Pr_x[C(x) = 1]| \leq \epsilon. \quad (2)$$

THEOREM 3 ([8]). *Given a function $f : \{0,1\}^{\log n} \to \{0,1\}$ with circuit complexity at least s, one can construct an $1/m$-PRG $G : \{0,1\}^{O(\log n)} \to \{0,1\}^m$ for size m circuits with $m = s^{\Omega(1/\log\log\log n)}$; moreover, G can be computed in $n^{O(1)}$ time with oracle access to f.*

Theorem 3 is *not* sufficient to obtain Theorem 1. Our main result is the construction of an "optimal" PRG that converts s bits of hardness into $s^{\Omega(1)}$ bits of *2-sided* pseudo-randomness, improving Theorem 3 and resolving an open question raised by [8] and [15].

STOC'02, May 19-21, 2002, Montreal, Quebec, Canada.

THEOREM 4 (THIS PAPER). *Given a function*

$$f : \{0,1\}^{\log n} \to \{0,1\}$$

with circuit complexity at least s, one can construct an $1/m$-PRG $G : \{0,1\}^{O(\log n)} \to \{0,1\}^m$ for size m circuits with $m = s^{\Omega(1)}$; moreover, G can be computed in $n^{O(1)}$ time with oracle access to f.

One nice consequence of this theorem is that it gives a direct, self-contained proof of Theorem 1, without appealing to [1]. Our PRG also avoids the explicit hardness amplification phase that was required in most prior (NW-style) PRGs, by converting worst-case hardness *directly* into 2-sided randomness. The only other such PRG construction which circumvents hardness amplification is due to [15], but it has a non-optimal seed length of $O(\log^2 n/\log s)$.

Theorem 4 shows in a precise sense the *equivalence* (up to a polynomial) between hardness and 2-sided pseudo-randomness, something that was not known previously. It is folklore (see also [7]) that s-bits of 2-sided pseudo-randomness (in the form a $\frac{1}{2}$-PRG for size s with seed-length t) gives rise to s bits of hardness (in the form of a function $f : \{0,1\}^{t+1} \to \{0,1\}$ with circuit complexity at least s). The other direction is considerably more difficult; our main theorem gives the first conversion in the other direction with only "reasonable" losses: from the aforementioned function, we construct a $\frac{1}{2}$-PRG for size $s^{\Omega(1)}$ with seed length $O(t)$. Thus we give the first *optimal* reduction from hard functions to PRGs, in the sense of [7] – any reduction that does significantly better produces a harder function than the one originally supplied to it.

From another persepective, PRGs, like extractors, have been used in (often complexity theoretic) applications to prove statements beyond their intended derandomization consequence (see, e.g., [11, 10, 12, 6]). From this viewpoint, PRGs are "tools" for converting between two fundamental resources – hardness and pseudo-randomness – and it is useful to have optimal PRGs at our disposal for use in complexity theoretic and other applications.

Finally, our PRG builds on the new algebraic PRG construction in [15], and it is possible that some of the algebraic structure that we exploit (which goes beyond that which is used in [15]) may be helpful in improving the extractor constructions in that paper, which use the same framework. Indeed, the PRG construction in this paper yields extractors via the connection between extractors and certain PRGs noticed by Trevisan [19]. It remains an open problem to construct optimal extractors, and the new techniques of [18] and [15] may be able to be pushed further with additional insight into the algebraic structure of these constructions; we hope some of the ideas in this paper can be useful to this end.

2. OVERVIEW OF OUR RESULTS

In addition to the HSG in Theorem 1, in [15] they also construct a PRG, but it only has the desired seed length when it is constructed from an exponentially hard function. Nevertheless, it is signficant in that it is the first PRG construction not based on [14] (with the exception of the Blum-Micali-Yao-style PRGs [2, 21], which are based on a qualitatively stronger hardness assumption). We are able to make the basic construction of [15] work for all hardnesses, using some new ideas which we outline below. Since the starting point for our construction is the PRG of [15], we outline their construction before explaining our improvements.

2.1 The PRG construction of [15]

As a first step we take the truth table of a hard function (which is just a n-bit string x), and encode it in a multivariate low-degree polynomial p_x so that given i, we can easily compute a location at which the evaluation of the polynomial gives us x_i. This is almost the usual "low-degree extension" encoding of x, however for technical reasons it is slightly non-standard in [15]. The polynomial p is defined over the vector space $\mathbb{F}_q^d$ (where $\mathbb{F}_q$ denotes the finite field of q elements), but we can also view $\mathbb{F}_q^d$ as the *extension field* $\mathbb{F}_{q^d}$. The multiplicative group of $\mathbb{F}_{q^d}$ is cyclic, so there is an $\alpha \in \mathbb{F}_{q^d}$ (a *primitive element*) that generates this group. Once we have α, we can write down the evaluations of p_x at every point in the order imposed by α – namely $p_x(\alpha^0), p_x(\alpha^1), p_x(\alpha^2), \ldots$.

A first attempt at building a PRG is to use the seed to choose a location in this list of evaluations of p_x, and output m successive evaluations in the list. It turns out that this is an extractor [15] but not yet a PRG[1], and to see why we need to examine the proof technique used by [15] and see where it breaks. The first step of the proof is standard: if this construction is *not* a PRG, then there is an efficient *next-element predictor* f that given the first $m-1$ output elements, outputs the m-th element with non-negligible probability. In our case, the next-element predictor predicts $p_x(\alpha^{j+m})$ from the $m-1$ previous evaluations $p_x(\alpha^{j+1}), p_x(\alpha^{j+2}), \ldots, p_x(\alpha^{j+m-1})$, for any j.

Now, we want to use f to build a small circuit that computes x_i given input i, contradicting the hardness of x. The idea (first used in [18]) is roughly this: we hardwire the first $m-1$ evaluations in the list into the circuit, and use f to predict the m-th evaluation; we then use f on the first $m-2$ evaluations in the list together with the just-predicted value to predict the $(m+1)$-st evaluation, and so on down the list. If f were *errorless*, this would give us a way to compute *every* evaluation, and since x_i is one of these evaluations, we would be able to compute x_i given i. In actuality, f makes many errors, but we can overcome that by exploiting the error-correcting properties of low-degree polynomials. Specifically, we use the predictor along a *random curve* at each step, and apply error correction along that curve. A good portion of [15] was devoted to making this general idea actually work, and we won't delve into the details here.

But, there is a more serious problem with our PRG attempt. The running time of the above procedure is proportional to the number of steps it takes. No matter how we "embed" x into p_x, there will be some i for which x_i is at least n steps away, and the size of our circuit will be at least n, which is far too large to contradict the hardness of x (which is *a priori* less than n). In fact it seems that any such sequential use of a single predictor will suffer from the same problem.

In [15], this problem is overcome by arranging to have *several* different predictors available, working at larger and larger "strides." By switching from one predictor to the next, they are able to "travel" from the locations of the hardwired evaluations to any given location in $\mathbb{F}_q^d$ in only

[1]In this discussion we ignore the distinction between outputting m field elements and outputting m bits.

poly(m) steps. The price for having several predictors is that they end up with several "candidate" PRGs, only one of which is guaranteed to be a PRG. The standard thing to do in this situation is to XOR the candidate PRGs with independent seeds, which produces a single PRG, but blows up the seed length. It is this blow-up that prevents [15] from achieving PRGs with logarithmic seed length for all except exponential hardnesses.

2.2 Our improvements

In this paper we suggest a method for making a *single* predictor work, by altering the *encoding* of x. The general idea is to force the evaluation of p_x at any given location in $\mathbb{F}_q^d$ to be *repeated* in a small number of carefully chosen different locations in $\mathbb{F}_q^d$. Then, after using the predictor for a small number of steps, we will have learned the evaluations of p_x at enough "faraway" locations to "jump" to a remote location and begin predicting there.

The challenge is to arrange the locations of repeated values so that this "jumping" allows us to get to any location in $\mathbb{F}_q^d$ in only poly(m) steps (and that the required sequence of predictor invocations can be easily computed). It turns out that the following scheme works: we ensure that $p_x(\alpha^i) = p_x(\alpha^{iq})$ for all i (here we use critically that $\alpha^{q^d} = \alpha$, so each location has at most $d-1$ associated locations with the same value). We illustrate a single "jump" in this scheme. Suppose for some j we have used the predictor to learn values $p_x(\alpha^j), p_x(\alpha^{j+1}), \dots, p_x(\alpha^{j+(m-1)q})$. Then we also know the values

$$p_x(\alpha^{jq^{d-1}}), p_x(\alpha^{jq^{d-1}+1}), \dots, p_x(\alpha^{jq^{d-1}+(m-1)})$$

since multiplying the exponent on α by q in each of these evaluations gives exponents $j, j+q, j+2q, \dots j+(m-1)q$. We can therefore "jump" to location $\alpha^{jq^{d-1}}$, and begin predicting there, since we know the requisite $m-1$ values needed to get the predictor started. Of course in this simplified presentation we are ignoring the fact that all of the techniques of [15] for handling error-correction along curves, etc... must be integrated, but with some effort this can be done.

In order to handle error-correction, we need p_x to be a low-degree polynomial, but then it is impossible to *simultaneously* enforce the above constraint (that $p_x(\alpha^i) = p_x(\alpha^{iq})$ for *all* i). We finesse this issue by allowing p_x to map into an expanded range $\mathbb{F}_{q^d}$ (instead of $\mathbb{F}_q$). The encoding is no longer a low-degree polynomial; however, viewing $\mathbb{F}_{q^d}$ as $\mathbb{F}_q^d$, *each coordinate* of p_x is a low-degree polynomial. This allows us to continue to use the error-correction properties of the original encoding while also benefiting from the ability to "jump" between locations using a single predictor.

The encoding we end up with turns out to be quite nice mathematically. Suppose we take our original row of evaluations of p_x and write below it the $d-1$ permutations of the same list obtained by applying each of the non-trivial field automorphisms $\sigma \in G$, where G is the Galois group of $\mathbb{F}_{q^d}$ over $\mathbb{F}_q$. Then the i-th element of the new encoding is the i-th column in this array. Just as [15] used critically that multiplication by α corresponds to a *linear* transform of $\mathbb{F}_q^d$, we use (in addition) that all $\sigma \in G$ are linear transforms of $\mathbb{F}_q^d$. Finally, we remark that the main new ingredient of our encoding – the repeated values at different locations – has a clean interpretation in terms of the Fourier Transform of the encoding, which we point out in Section 3.3.

3. THE ENCODING

In this section we describe a function $\widetilde{\mathrm{LDE}}_x$ that encodes a string $x \in \{0,1\}^k$. This function resembles a standard "low-degree extension" with an important additional "periodicity" property that we describe below. We begin with some preliminaries.

3.1 Preliminaries

We will be working with the field of q elements, $\mathbb{F}_q$, and the extension field $\mathbb{F}_{q^d}$, which is a vector space over $\mathbb{F}_q$. In computations, we assume that elements of $\mathbb{F}_{q^d}$ are represented as vectors in $\mathbb{F}_q^d$ with respect to some basis $\{\beta_0, \beta_1, \dots, \beta_{d-1}\}$, where the β_i are elements of $\mathbb{F}_{q^d}$. The *standard basis* is $\{1, z, z^2, \dots, z^{d-1}\}$, where z is an indeterminate, and we think of $\mathbb{F}_{q^d}$ as polynomials in z modulo an irreducible degree d polynomial $p(z)$.

A very useful alternative basis is a *normal basis* which is a basis of the form $\{\sigma\alpha | \sigma \in G\}$, where α is an element of the extension field and G is the Galois group of the extension. In our case the Galois group of $\mathbb{F}_{q^d}$ over $\mathbb{F}_q$ is cyclic of order d and generated by the *Frobenius automorphism*, $\alpha \mapsto \alpha^q$, for $\alpha \in \mathbb{F}_{q^d}$. Thus a normal basis for $\mathbb{F}_{q^d}$ over $\mathbb{F}_q$ has the form $\left\{\alpha, \alpha^q, \alpha^{q^2}, \dots, \alpha^{q^{d-1}}\right\}$, for some $\alpha \in \mathbb{F}_{q^d}$. The "normal basis theorem," found in most algebra texts, states that a normal basis exists for any finite Galois extension of fields, which certainly covers our situation. For our purposes it will be sufficient to observe that a normal basis can be found in time polynomial in the size of the extension field.

We will also need the following lemma which allows us to construct certain extension fields using an irreducible polynomial whose coefficients all lie in a subfield of the base field.

LEMMA 1. *Let h be a prime power and let a and b be relatively prime integers. Then there exists a degree b irreducible polynomial over $\mathbb{F}_{h^a}$ all of whose coefficients lie in $\mathbb{F}_h$.*

PROOF. The field $\mathbb{F}_{h^{ab}}$ contains both $\mathbb{F}_{h^a}$ and $\mathbb{F}_{h^b}$. The field $\mathbb{F}_{h^b}$ is obtained by adjoining the root α of a degree b polynomial $p(z)$ irreducible over $\mathbb{F}_h$. The field $\mathbb{F}_{h^a}(\alpha)$ contains both $\mathbb{F}_{h^a}$ and $\mathbb{F}_{h^b}$ and is contained in $\mathbb{F}_{h^{ab}}$. However, no proper subfield of $\mathbb{F}_{h^{ab}}$ can contain both $\mathbb{F}_{h^a}$ and $\mathbb{F}_{h^b}$ since $(a, b) = 1$. Therefore $\mathbb{F}_{h^a}(\alpha) = \mathbb{F}_{h^{ab}}$, and so $p(z)$ must be irreducible over $\mathbb{F}_{h^a}$. $\square$

A fact that we will use heavily is that the Frobenius map $\alpha \mapsto \alpha^q$ is $\mathbb{F}_q$*-linear*; that is, $(\alpha+\beta)^q = \alpha^q + \beta^q$ for all $\alpha, \beta \in \mathbb{F}_{q^d}$ and $(a\beta)^q = a\beta^q$ for all $a \in \mathbb{F}_q$ and $\beta \in \mathbb{F}_{q^d}$. The latter equality follows from the fact that $a^q = a$ iff $a \in \mathbb{F}_q$. One nice feature of a normal basis is that exponentiation by q is simply a cyclic shift of the coordinates.

The multiplicative group of a finite field is cyclic; a generator of this group is called a *primitive element* of the field.

3.2 The augmented low-degree extension

Pick parameters d, h, and q, and let $\left\{\beta, \beta^q, \beta^{q^2}, \dots, \beta^{q^{d-1}}\right\}$ be a normal basis for $\mathbb{F}_{q^d}$ over $\mathbb{F}_q$. From here on, we assume that elements of $\mathbb{F}_{q^d}$ are stored as vectors in $\mathbb{F}_q^d$ with respect to this basis; we also identify $\mathbb{F}_q^d$ with $\mathbb{F}_{q^d}$ via this basis. Let α be a primitive element of $\mathbb{F}_{q^d}$.

String $x \in \{0,1\}^k$ is the string we wish to encode. Our first step is to encode x using a low-degree extension. As in [15], this encoding is slightly non-standard. We desire a function $\mathrm{LDE}_x : \mathbb{F}_q^d \rightarrow \mathbb{F}_q$ with the following properties:

$(P1)$ LDE_x is a d-variate polynomial with total-degree at most dh, and

$(P2)$ there exists a polynomial-time computable function $\ell : [k] \rightarrow [q^d - 1]$ such that $\mathrm{LDE}_x(\alpha^{\ell(i)}) = x_i$ for all $i \in [k]$.

This is only a slightly more restrictive that the usual scenario in which we expect a function $\ell' : [k] \rightarrow \mathbb{F}_q^d$ for which $\mathrm{LDE}_x(\ell'(i)) = x_i$; here we require that the function have the form $\ell'(i) = \alpha^{\ell(i)}$. Such a low-degree extension can be constructed as long as some modest restrictions on h, d, and q are met[2]:

THEOREM 5. *There exists a function $LDE_x : \mathbb{F}_q^d \rightarrow \mathbb{F}_q$ satisfying $(P1)$ and $(P2)$ provided h is a prime power, $h^d > k$, q is a power of h, and $(\log_h q, d) = 1$. Given x, LDE_x can be constructed in poly(q^d) time.*

PROOF. Since q is a power of h we have $\mathbb{F}_h \subseteq \mathbb{F}_q$. By Lemma 1, $\mathbb{F}_{q^d}$ can be obtained by adjoining the root of a degree d polynomial $p(z)$ irreducible over $\mathbb{F}_q$ and with coefficients in $\mathbb{F}_h$. This implies that the set

$$T = \left\{ \sum_{i=0}^{d-1} a_i z^i | a_i \in \mathbb{F}_h \right\} \setminus \{0\}$$

is a subgroup of the multiplicative group of $\mathbb{F}_{q^d}$; it has order $h^d - 1$ and so setting $r = (q^d - 1)/(h^d - 1)$, we have $T = \{\alpha^{rj}\}_{j=1,2,\ldots,h^d-1}$.

Now, for $i = 0, 1, \ldots, k-1$, we compute vector representations for the elements of T, i.e., we compute a's in $\mathbb{F}_h$ so that $\alpha^{ri} = \sum_{j=0}^{d-1} a_j^{(i)} z^j$. The following standard low-degree extension $\mathrm{LDE}'_x : \mathbb{F}_q^d \rightarrow \mathbb{F}_q$ is not yet our final polynomial:

$$\mathrm{LDE}'_x(y_0, y_1, \ldots y_{d-1}) = \sum_{i=0}^{k-1} x_i \cdot \prod_{j=0}^{d-1} \frac{\prod_{b \in \mathbb{F}_h, b \neq a_j^{(i)}} (y_j - b)}{\prod_{b \in \mathbb{F}_h, b \neq a_j^{(i)}} (a_j^{(i)} - b)} \tag{3}$$

It is clear that LDE'_x has total degree at most hd. Now, let $L : \mathbb{F}_q^d \rightarrow \mathbb{F}_q^d$ be the linear transform corresponding to a change of basis from $\left\{\beta, \beta^q, \ldots, \beta^{q^{d-1}}\right\}$ to $\{1, z, \ldots, z^{d-1}\}$. Our final polynomial LDE_x is defined as

$$\mathrm{LDE}_x(y_0, \ldots, y_{d-1}) = \mathrm{LDE}'_x(L(y_0, \ldots, y_{d-1})). \tag{4}$$

Clearly LDE_x has total degree at most hd and therefore satisfies property $(P1)$. Defining $\ell(i) = ri$ we see that:

$$\begin{aligned} \mathrm{LDE}_x(\alpha^{\ell(i)}) &= \mathrm{LDE}_x(\alpha^{ri}) = \mathrm{LDE}'_x(L(\alpha^{ri})) \\ &= \mathrm{LDE}'_x(a_0^{(i)}, a_1^{(i)}, \ldots, a_{d-1}^{(i)}) = x_i, \end{aligned}$$

and ℓ is clearly easy to compute, so property $(P2)$ is satisfied. $\square$

Now, let $\sigma : \mathbb{F}_{q^d} \rightarrow \mathbb{F}_{q^d}$ be the Frobenius map and let $\pi_i : \mathbb{F}_q^d \rightarrow \mathbb{F}_q$ be the projection of the i-th coordinate, for $i = 0, 1, \ldots d-1$. We are seeking a function $\widetilde{\mathrm{LDE}}_x : \mathbb{F}_{q^d} \rightarrow \mathbb{F}_{q^d}$ with the following properties:

$(\tilde{P}1)$ for all i, $\pi_i \widetilde{\mathrm{LDE}}_x$ is a d-variate polynomial with total degree at most dh, and

$(\tilde{P}2)$ there exists a polynomial-time computable function $\ell : [k] \rightarrow [q^d - 1]$ such that $\pi_0(\widetilde{\mathrm{LDE}}_x(\alpha^{\ell(i)})) = x_i$ for all $i \in [k]$, and

$(\tilde{P}3)$ $\sigma \widetilde{\mathrm{LDE}}_x = \widetilde{\mathrm{LDE}}_x \sigma$.

It would be easy to achieve properties $(\tilde{P}1)$ and $(\tilde{P}2)$ by simply repeating LDE_x in each of the d coordinates; the interesting requirement is $(\tilde{P}3)$, and in Section 3.3 we show that is has a clean interpretation in terms of the Fourier Transform of the function $\widetilde{\mathrm{LDE}}_x$.

We can now define the *augmented low-degree extension* and prove that it satisfies the above three properties.

DEFINITION 3 (AUGMENTED LOW-DEGREE EXTENSION). *Given $x \in \{0,1\}^k$, define $\widetilde{LDE}_x : \mathbb{F}_{q^d} \rightarrow \mathbb{F}_{q^d}$ as follows:*

$$\widetilde{LDE}_x(y) = \sum_{j=0}^{d-1} LDE_x(\sigma^{-j} y) \cdot \beta^{q^j}. \tag{5}$$

THEOREM 6. *The augmented low-degree extension $\widetilde{LDE}_x$ satisfies properties $(\tilde{P}1)$, $(\tilde{P}2)$, and $(\tilde{P}3)$, provided h is a prime power, $h^d > k$, q is a power of h, and $(\log_h q, d) = 1$. Given x, $\widetilde{LDE}_x$ can be constructed in poly(q^d) time.*

PROOF. If follows from the fact that σ is an $\mathbb{F}_q$-linear map that $\mathrm{LDE}_x \sigma^{-j}$ is a d-variate total degree dh polynomial if LDE_x is. Hence $\widetilde{\mathrm{LDE}}_x$ satisfies property $(\tilde{P}1)$.

It is clear that $\pi_0 \widetilde{\mathrm{LDE}}_x = \mathrm{LDE}_x$, and therefore $\widetilde{\mathrm{LDE}}_x$ enjoys property $(\tilde{P}2)$ since LDE_x satisfies the corresponding property $(P2)$.

For property $(\tilde{P}3)$, we observe that for all i,

$$\begin{aligned} \sigma \widetilde{\mathrm{LDE}}_x(\alpha^i) &= \left(\sum_{j=0}^{d-1} \mathrm{LDE}_x(\sigma^{-j} \alpha^i) \beta^{q^j} \right)^q \\ &= \sum_{j=0}^{d-1} \mathrm{LDE}_x(\sigma^{-j} \alpha^i) \beta^{q^{j+1}} \\ &= \sum_{j=0}^{d-1} \mathrm{LDE}_x(\sigma^{-j} \sigma \alpha^i) \beta^{q^j} = \widetilde{\mathrm{LDE}}_x(\sigma \alpha^i), \end{aligned}$$

and so $\sigma \widetilde{\mathrm{LDE}}_x = \widetilde{\mathrm{LDE}}_x \sigma$ as required. $\square$

3.3 Fourier interpretation of property $(\tilde{P}3)$

In this section we observe that property $(\tilde{P}3)$, which is exactly what we need for our PRG construction, has a clean mathematical interpretation. The characterization in this section is not necessary for the remainder of the paper however, and the reader may safely skip to the construction of the PRG in Section 4 if they wish.

DEFINITION 4 (FINITE FIELD FOURIER TRANSFORM). *Let $\mathbb{F}$ be a field of n elements and α a primitive element of $\mathbb{F}$. Given a function $f : \mathbb{F} \rightarrow \mathbb{F}$, the Fourier transform of f is given by:*

$$\hat{f}(\alpha^j) = \sum_{i=0}^{n-1} f(\alpha^i) \alpha^{ij} \tag{6}$$

[2]The conditions stated in [15] are slightly inaccurate, but this does not affect their construction.

and the Inverse Fourier Transform is given by:

$$f(\alpha^i) = \frac{1}{n}\sum_{j=0}^{n-1}\hat{f}(\alpha^j)\alpha^{-ij}. \quad (7)$$

We reproduce the following theorem essentially from [20]. It is used there for a presentation of BCH codes, which can be seen as the Fourier transforms of functions satisfying property $(\tilde{P}3)$ and one additional property.

THEOREM 7. *A function $f : \mathbb{F}_{q^d} \to \mathbb{F}_{q^d}$ satisfies $\sigma f = f\sigma$ if and only if the image of $\hat{f}$ is contained in $\mathbb{F}_q$.*

PROOF. $(\Rightarrow)$ Suppose that $\sigma f = f\sigma$. Then we have for all j,

$$\begin{aligned}(\hat{f}(\alpha^j))^q &= \left(\sum_{i=0}^{n-1} f(\alpha^i)\alpha^{ij}\right)^q = \sum_{i=0}^{n-1}\sigma f(\alpha^i)\alpha^{ijq} \\ &= \sum_{i=0}^{n-1} f(\sigma\alpha^i)\alpha^{ijq} = \sum_{i=0}^{n-1} f(\alpha^{iq})\alpha^{(iq)j} = \hat{f}(\alpha^j),\end{aligned}$$

which implies that $\hat{f}(\alpha^j) \in \mathbb{F}_q$ (the last equality holds since $\{iq : i \in [n]\} = [n]$).

$(\Leftarrow)$ Suppose that $\hat{f}(\alpha^j) \in \mathbb{F}_q$ for all j. Then we have for all i:

$$\begin{aligned}\sigma f(\alpha^i) &= \left(\frac{1}{n}\sum_{j=0}^{n-1}\hat{f}(\alpha^j)\alpha^{-ij}\right)^q = \left(\frac{1}{n}\right)^q \sum_{j=0}^{n-1}\hat{f}(\alpha^j)^q\alpha^{-(iq)j} \\ &= \frac{1}{n}\sum_{j=0}^{n-1}\hat{f}(\alpha^j)\alpha^{-(iq)j} = f(\alpha^{iq}) = f(\sigma\alpha^i),\end{aligned}$$

and hence $\sigma f = f\sigma$. □

4. THE PRG CONSTRUCTION

We can now present our PRG construction. As in [15] we first build a PRG over a large (non-binary) alphabet, and then convert it to a PRG satisfying the usual definition (Definition 2). In place of equation (2) in that definition, it is sufficient to require that there be no efficient *next-bit predictor* – a function that can predict the i-th bit of output given the first $i-1$ bits of output significantly better than random guessing. This alternative requirement turns out to be the correct thing to generalize to define q-ary PRGs:

DEFINITION 5 (q-ARY PRG). *Let Σ be an alphabet of q elements. A ρ-q-ary* PRG for size *s is a function $G : \{0,1\}^t \to \Sigma^m$ such that for all $1 \le i \le m$ and all functions $f : \Sigma^{i-1} \to \Sigma^{(\rho^{-2})}$ with size s circuits,*

$$\Pr_y[\exists j \;\; f(G(y)_{1\ldots i-1})_j = G(y)_i] \le \rho. \quad (8)$$

Using an efficiently encodable and list-decodable binary code, we can easily convert q-ary PRGs into ordinary PRGs.

DEFINITION 6. *A binary code $C : \{0,1\}^{\bar{k}} \to \{0,1\}^{\bar{n}}$ is* (ρ,ℓ)-list-decodable *if for all $r \in \{0,1\}^{\bar{n}}$, the set $S_r = \{x : \Delta(C(x), r) \le (1/2-\rho)\bar{n}\}$ has size at most ℓ. The code is* efficiently encodable *if C is computable in time poly$(\bar{n})$, and* efficiently list-decodable *if S_r can be computed from r in time poly$(\bar{n},\ell)$.*

By the Johnson bound (see, e.g., [5]) any family of binary codes with relative distance at least $1/2-\rho^2$ is (ρ,ρ^{-2})-list-decodable. Guruswami and Sudan [4] give several efficiently encodable and (ρ,ρ^{-2})-list-decodable binary codes with blocklength $\bar{n} = \text{poly}(\bar{k},\rho^{-1})$, which is sufficient for our purposes. The following lemma (see, e.g., [15]) is standard – it essentially applies the Goldreich-Levin hard-core bit reduction to each output element of the q-ary PRG:

LEMMA 2. *Let Σ be an alphabet of q elements, and let $G : \{0,1\}^t \to \Sigma^m$ be a ρ-q-ary PRG for size s. Let $C : \{0,1\}^{\bar{k}=\log q} \to \{0,1\}^{\bar{n}}$ be an efficiently encodable and efficiently list-decodable (ρ,ρ^{-2})-list-decodable code. Then $G' : \{0,1\}^{t+\log\bar{n}} \to \{0,1\}^m$ defined by:*

$$G'(y,j) = C(G(y)_1)_j \circ C(G(y)_2)_j \circ \cdots \circ C(G(y)_m)_j$$

is a $2\rho m$-PRG for size s', provided $s \ge s'\bar{n} + m\bar{n}^{O(1)} + (\bar{n}/\rho)^{O(1)}$.

In other words, an $(\epsilon/2m)$-q-ary-PRG for size s with output length m can be converted into a ϵ-PRG for size not much smaller than s with no loss in the output length.

Let $S(x)$ denote the circuit complexity of string $x \in \{0,1\}^k$, viewed as the truth table of a Boolean function. Our main theorem is the following:

THEOREM 8 (MAIN). *For every k, d, prime power h, $q = h^c$, string $x \in \{0,1\}^k$ and primitive element α of $\mathbb{F}_{q^d}$ satisfying $h^d > k$ and $(c,d) = 1$, the function $G_x : \mathbb{F}_{q^d} \to \mathbb{F}_{q^d}^m$ defined as follows:*

$$G_x(y) = \widetilde{LDE}_x(\alpha y) \circ \widetilde{LDE}_x(\alpha^2 y) \circ \cdots \circ \widetilde{LDE}_x(\alpha^m y) \quad (9)$$

is a ρ-q^d-ary PRG for size s, provided $S(x) > s \cdot poly(m,q)$ and $q > \Omega(\rho^{-4}hd^4\log^2 q)$. Furthermore, G_x is computable in poly(q^d) time with oracle access to x.

Given an n-bit string x with $S(x) \ge \log^{O(1)} n$ we can prove Theorem 4 by plugging in the following parameters to Theorem 8:

- $m = S(x)^\delta$ for a sufficiently small constant $\delta < 1$
- $\rho = \frac{1}{2m^2}$
- $k = n$
- d prime between $\frac{\log k}{\log m}$ and $2 \cdot \frac{\log k}{\log m}$
- h prime between m and $2m$
- q smallest power of h larger than $\Omega(m^9 d^4 \log^2 q)$

One can verify that k, d, h and q satisfy the requirements of Theorem 8, which then produces a ρ-q^d-ary PRG for size $s = m \cdot \log^{O(1)} n$. Since $q = \text{poly}(m)$ we have $q^d = \text{poly}(n)$, and so applying Lemma 2 gives a $1/m$-PRG for size m as required. Note that Theorem 4 is trivial for $S(x) \le \log^{O(1)} n$, as then the identity function suffices.

5. PROOF OF THE MAIN THEOREM

In this section we give the proof of Theorem 8. Our proof follows the general outline of [17] and [15] and resembles the proof of [15] in many respects. We include it here mainly because there are some critical new details and the notation is quite different from previous accounts. Moreover, this section makes the results of this paper completely self-contained with the exception of the following lemma from [16] that allows us to list-decode low-degree univariate polynomials from very noisy data:

LEMMA 3 ([16]). *Given n distinct pairs (x_i, y_i) in field $\mathbb{F}_q$ and parameters* **agree** *and* **deg** *with* $\mathbf{agree} > \sqrt{2 \cdot \mathbf{deg} \cdot n}$, *there are at most $2n/\mathbf{agree}$ degree* **deg** *polynomials p such that $p(x_i) = y_i$ for at least* **agree** *pairs. Moreover, a list of all such polynomials can be computed in time $poly(n, \log q)$.*

We assume for the purpose of contradiction that we have a *next-element predictor* $f : \mathbb{F}_{q^d}^{i-1} \to \mathbb{F}_{q^d}^{(\rho^{-2})}$, computable by a size s circuit, and violating equation (8). For ease of notation we assume that $i = m$, WLOG since the predictor can simply ignore some prefix of its input. In the proof we describe an oracle function $R : \{0,1\}^t \times [k] \to \{0,1\}$ running in time $\mathrm{poly}(m, q)$, and we argue that there exists an *advice string* $A \in \{0,1\}^t$ for which $R^f(A, i) = x_i$ for all i. Replacing oracle calls with copies of the size s circuit for f, and hardwiring the "good" advice string A, we obtain a circuit of size $s \cdot \mathrm{poly}(m, q)$ computing x, contradicting the hardness of x.

The proof is somewhat lengthy, so we divide it into three parts. In the first part, we show the existence of two *reference curves* that satisfy certain properties. In the second part we show how to perform an *interleaved learning* step, which allows us to learn the evaluations of $\widetilde{\mathrm{LDE}}_x$ at some new locations given the evaluations of $\widetilde{\mathrm{LDE}}_x$ at some old locations. In the third part, we describe how to use a short sequence of interleaved learning steps to learn x_i, given i.

5.1 Part 1: reference curves

Set $r = O(d^2 \log q)$. We show by the probabilistic method that there exist "good" degree r *reference curves* $C_1 : \mathbb{F}_q \to \mathbb{F}_q^d$ and $C_2 : \mathbb{F}_q \to \mathbb{F}_q^d$. By degree r we mean that $\pi_j C_1$ and $\pi_j C_2$ are degree r univariate polynomials, for all $j \in [d]$. Using Property $(\tilde{P}1)$ and the fact that α and σ are $\mathbb{F}_q$-linear maps from $\mathbb{F}_q^d$ to itself, we see that

$$p(b) = \pi_j(G_x(\alpha^i \sigma^\ell C_1(b))_m) = \pi_j \widetilde{\mathrm{LDE}}_x(\alpha^{m+i} \sigma^\ell C_1(b))$$

is a degree $\le rdh$ univariate polynomial for all $i \in [q^d - 1]$, all $j \in [d]$ and all $\ell \in [d]$; the same holds for C_2.

We want C_1 and C_2 to satisfy the following three properties, for all $i \in [q^d - 1]$, all $j \in [d]$, and all $\ell \in [d]$:

1. the success rate of f along any "shift" of C_1 or C_2 is close to its overall success rate ρ:

$$\Pr_{b \in \mathbb{F}_q}[\exists j \;\; f(G_x(\alpha^i \sigma^\ell C_1(b))_{1\ldots m-1})_j = G_x(\alpha^i \sigma^\ell C_1(b))_m] \ge \rho/2,$$

   and the same holds for C_2.

2. If $T_{i,j,\ell}^{(1)}$ is a fixed set of at most $4\rho^{-3}$ degree $\le rdh$ univariate polynomials including

$$p(b) = \pi_j(G_x(\alpha^i \sigma^\ell C_1(b))_m)$$

   then for all $p' \in T_{i,j,\ell}^{(1)}$ such that $p' \not\equiv p$, there exists $b \in \{b | \sigma^\ell C_1(b) = \alpha^{-1} \sigma^\ell C_2(b)\}$ for which $p'(b) \neq p(b)$.

3. If $T_{i,j,\ell}^{(2)}$ is a fixed set of at most $4\rho^{-3}$ degree $\le rdh$ univariate polynomials including

$$p(b) = \pi_j(G_x(\alpha^i \sigma^\ell C_2(b))_m)$$

   then for all $p' \in T_{i,j,\ell}^{(2)}$ such that $p' \not\equiv p$, there exists $b \in \{b | C_1(b) = C_2(b)\}$ for which $p'(b) \neq p(b)$.

Picking C_1 and C_2 randomly as specified in [15], we get that for fixed i, j, and ℓ, the above three properties hold with probability at least $1 - \frac{1}{2d^2 q^d}$. Then by the union bound, C_1 and C_2 satisfy the above three properties for *all* i, j, and ℓ simultaneously with probability at least $1/2$. Therefore the desired C_1 and C_2 exist. The details of this argument follow [15] closely; for completeness we now briefly describe how to pick the curves.

Set $r' = r/(d+1)$ and pick r random elements of $\mathbb{F}_q$:

$$b_{01}, \ldots, b_{0r'}, b_{11}, \ldots, b_{1r'}, \ldots, b_{d1}, \ldots, b_{dr'}$$

and r random vectors in $\mathbb{F}_q^d$:

$$v_{01}, \ldots, v_{0r'}, v_{11}, \ldots, v_{1r'}, \ldots, v_{d1}, \ldots, v_{dr'}.$$

Curves C_1 and C_2 are degree $(r-1)$ curves for which $C_1(b_{ij}) = v_{ij}$ for all i, j and $C_2(b_{ij}) = v_{ij}$ for $i = 0$ and all j, and $C_2(b_{ij}) = \alpha^{q^i} v_{ij}$ for all $i > 0$ and all j.

It is clear that for all i and all ℓ, the set $\{\alpha^i \sigma^\ell C_1(b) | b \in \mathbb{F}_q\}$ is an r-wise independent set of points in $\mathbb{F}_q^d$, and the same holds with respect to C_2. This implies that property (1) holds with very high probability, using a higher-moment tail inequality. It is also clear that $\{b | C_1(b) = C_2(b)\} = \{b_{0j}\}_{j \in [r']}$ is a set of r' random points in $\mathbb{F}_q$, which implies that property (3) holds with very high probability. Similarly, $\{b | \sigma^\ell C_1(b) = \alpha^{-1} \sigma^\ell C_2(b)\} = \{b_{(d-\ell)j}\}_{j \in [r']}$ is a set of r' random points in $\mathbb{F}_q$, which implies that property (2) holds with very high probability.

5.2 Part 2: a single interleaved learning step

For all $i \in [q^d - 1]$ and all $\ell \in [d]$, we describe how to learn the values of

$$G_x(\alpha^i \sigma^\ell C_1(b))_m \text{ and } G_x(\alpha^i \sigma^\ell C_2(b))_m \;\; \forall b \in \mathbb{F}_q$$

given the values of

$$G_x(\alpha^i \sigma^\ell C_1(b))_{1\ldots m-1} \text{ and } G_x(\alpha^i \sigma^\ell C_2(b))_{1\ldots m-1} \;\; \forall b \in \mathbb{F}_q$$

in $\mathrm{poly}(m, q)$ time, with oracle access to the next-element predictor f. The algorithm follows:

- compute a set of "predicted" values for all $j \in [d]$ and $b \in \mathbb{F}_q$, using q calls to f:

$$S_{j,b}^{(1)} = \left\{\pi_j f(G_x(\alpha^i \sigma^\ell C_1(b))_{1\ldots m-1})_s\right\}_{s \in [\rho^{-2}]}.$$

- Notice that $p_j(b) = \pi_j(G_x(\alpha^i \sigma^\ell C_1(b))_m)$ is a univariate polynomial of degree $\le rdh$, and by property (1) in Part 1, it agrees with at least $(\rho/2)q$ of the (point, evaluation) pairs $\left\{(b, c) : b \in \mathbb{F}_q, c \in S_{j,b}^{(1)}\right\}$. Using Lemma 3, we compute the set $T_{i,j,\ell}^{(1)}$ of at most $4\rho^{-3}$ degree $\le rdh$ polynomials with this agreement.

- We know the values of

$$G_x(\alpha^i \sigma^\ell C_2(b))_{m-1} = G_x(\alpha^{i-1} \sigma^\ell C_2(b))_m$$

for all $b \in \mathbb{F}_q$, so using property (2) in Part 1, we can pick $p_j(b)$ out of $T^{(1)}_{i,j,\ell}$ (by selecting the unique polynomial that agrees with the evaluations of

$$\pi_j(G_x(\alpha^i \sigma^\ell C_1(b))_m)$$

that we already know) for all j. Since $G_x(\alpha^i\sigma^\ell C_1(b))_m = \sum_{j=0}^{d-1} p_j(b)\beta^{q^j}$, we have learned $G_x(\alpha^i\sigma^\ell C_1(b))_m$.

- Repeat with respect to C_2; that is, compute sets of predicted values $S^{(2)}_{j,b}$, use Lemma 3 to compute sets of candidate polynomials $T^{(2)}_{i,j,\ell}$, and use property (3) in Part 1 to pick the "correct" candidate polynomials (since we now know the values $G_x(\alpha^i\sigma^\ell C_1(b))_m$ for all $b \in \mathbb{F}_q$). In the end we have learned $G_x(\alpha^i\sigma^\ell C_2(b))_m$.

Altogether, the above algorithm requires $\mathrm{poly}(m,q)$ time with oracle access to f.

5.3 Part 3: a sequence of interleaved learning steps

Given $i \in [k]$, we show how to use a short sequence of at most dmq interleaved learning steps to learn x_i, given the values of

$$G_x(C_1(b))_{1\ldots m-1} \text{ and } G_x(C_2(b))_{1\ldots m-1} \quad \forall b \in \mathbb{F}_q.$$

We will use $V_{j,\ell}$ as shorthand for "the values of

$$G_x(\alpha^{j-m}\sigma^\ell C_1(b))_m \text{ and } G_x(\alpha^{j-m}\sigma^\ell C_2(b))_m$$

for all $b \in \mathbb{F}_q$." Since $\alpha^{q^d-1} = \alpha^0$ and $\sigma^d = \sigma^0$, we consider the subscripts of $V_{j,\ell}$ modulo $q^d - 1$ and d, respectively. Notice that for all i and all ℓ a single interleaved learning step allows us to learn $V_{i+m,\ell}$ from $V_{i+1,\ell}, V_{i+2,\ell}, \ldots, V_{i+m-1,\ell}$. The key consequence of Property ($\tilde{P}3$) that we use is that $V_{iq,\ell+1}$ can be efficiently computed from $V_{i,\ell}$, and vice versa, for all i and ℓ. This follows from:

$$\begin{aligned}
\sigma(G_x(\alpha^{i-m}\sigma^\ell C_1(b))_m) &= \sigma\widetilde{\mathrm{LDE}}_x(\alpha^i\sigma^\ell C_1(b)) \\
&= \widetilde{\mathrm{LDE}}_x(\sigma(\alpha^i\sigma^\ell C_1(b))) \\
&= \widetilde{\mathrm{LDE}}_x(\alpha^{iq}\sigma^{\ell+1} C_1(b)) \\
&= G_x(\alpha^{iq-m}\sigma^{\ell+1} C_1(b))_m
\end{aligned}$$

for all $b \in \mathbb{F}_q$, which also holds with respect to C_2.

We now proceed to describe the algorithm. Let $c \in [q^d-1]$ be such that $\alpha^c = C_1(0)$, and set $t = mq$. Set $b = \ell(i)-c-1$, and write it in its q-ary representation: $b = \sum_{\ell=0}^{d-1} b_\ell q^\ell$.

- Set $a^{(0)} = 0$. Notice that
$$V_{a^{(0)}+1,0}, V_{a^{(0)}+2,0}, \ldots, V_{a^{(0)}+(m-1),0}$$
are given. Use interleaved learning to learn
$$V_{a^{(0)}+m,0}, V_{a^{(0)}+(m+1),0}, \ldots, V_{a^{(0)}+t,0}.$$

- For $\ell = 1, 2, \ldots, d-1$:
  - Set $a^{(\ell)} = (a^{(\ell-1)} + b_{\ell-1})q^{d-1}$. Notice that
  $$V_{a^{(\ell)}+1,-\ell}, V_{a^{(\ell)}+2,-\ell}, \ldots, V_{a^{(\ell)}+(m-1),-\ell}$$
  can be easily computed from
  $$V_{a^{(\ell-1)}+b_{\ell-1}+q,-(\ell-1)}, V_{a^{(\ell-1)}+b_{\ell-1}+2q,-(\ell-1)}, \ldots, V_{a^{(\ell-1)}+b_{\ell-1}+(m-1)q,-(\ell-1)},$$
  which we know from the previous step. Use interleaved learning to learn
  $$V_{a^{(\ell)}+m,-\ell}, V_{a^{(\ell)}+(m+1),-\ell}, \ldots, V_{a^{(\ell)}+t,-\ell}.$$

- Set $a^{(d)} = (a^{(d-1)} + b_{d-1})q^{d-1}$. Notice that $V_{a^{(d)}+1,-d}$ can be efficiently computed from $V_{a^{(d-1)}+b_{d-1}+q,-(d-1)}$ which we know from the previous step. Output
$$\pi_0(G_x(\alpha^{a^{(d)}+1-m}C_1(0))_m),$$
whose value is contained in $V_{a^{(d)}+1,-d}$.

By an easy induction on ℓ, we see that for $\ell = 1, 2, \ldots d$,

$$a^{(\ell)} = \sum_{j=d-\ell}^{d-1} b_{j-d+\ell} q^j,$$

where all integers are considered modulo q^d-1. We conclude that we have output x_i, since:

$$\begin{aligned}
\pi_0(G_x(\alpha^{a^{(d)}+1-m}C_1(0))_m) &= \pi_0\widetilde{\mathrm{LDE}}_x(\alpha^{a^{(d)}+1}C_1(0)) \\
&= \pi_0\widetilde{\mathrm{LDE}}_x(\alpha^{b+1}C_1(0)) \\
&= \pi_0\widetilde{\mathrm{LDE}}_x(\alpha^{\ell(i)}) = x_i,
\end{aligned}$$

using Property ($\tilde{P}2$) for the last equality. The total number of interleaved learning steps we used was $d(t-m+1) \le dmq$.

5.4 Putting it all together

It remains to describe the advice required for the above procedure to get started. We need to provide:

- degree r curves C_1 and C_2 satisfying the three properties listed in subsection 5.1. Each can be described by r coefficients in $\mathbb{F}_q^d$.
- values of
$$G_x(C_1(b))_{1\ldots m-1} \text{ and } G_x(C_2(b))_{1\ldots m-1}$$
for all $b \in \mathbb{F}_q$. As we have observed, these can be described as $2d(m-1)$ different degree $\le rdh$ univariate polynomials, each requiring rdh coefficients in $\mathbb{F}_q$.
- integer $c \in [q^d - 1]$ for which $\alpha^c = C_1(0)$, and a few miscellaneous items for field arithmetic: the basis for $\mathbb{F}_{q^d}$ over $\mathbb{F}_q$, the irreducible polynomial used to construct the extension field, and the primitive element α.

Altogether the advice string has length $(2rd + 2rd^2h(m-1) + O(d))\log q = \mathrm{poly}(m,q)$. The overall running time with oracle access to f is also $\mathrm{poly}(m,q)$, so after substituting our size s circuit for f, and hardwiring the advice, we get a circuit of size at most $s \cdot \mathrm{poly}(m,q)$ that computes x_i given i, contradicting the hardness of x. This concludes the proof of the main theorem.

6. CONCLUSIONS

We have given an optimal (up to a polynomial) construction of PRGs from hard functions. The basic framework of [18], as extended by [15], has now been used to construct quite a variety of derandomization objects – extractors for a wide range of parameters, optimal HSGs, optimal nondeterministic HSGs (from SV-nondeterministic hardness, a la [13]), and in the present paper, optimal PRGs. We remark that the extension to non-deterministic HSGs in [15] also applies to our PRG construction, so a further consequence of this paper is a construction of optimal non-deterministic PRGs.

It seems that even quite elementary algebra is a useful tool in this arena, and as noted in the introduction it would be interesting to see if the current set of ideas with some new tricks can produce optimal extractors, which remain an important and so-far elusive goal in this area.

7. ACKNOWLEDGEMENTS

We thank Henry Cohn and Ronen Shaltiel for useful discussions.

8. REFERENCES

[1] A. Andreev, A. Clementi, and J. Rolim. A new general derandomization method. *J. ACM*, 45(1), 1998.

[2] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput.*, 13(4):850–864, Nov. 1984.

[3] O. Goldreich, S. Vadhan, and A. Wigderson. Simplified derandomization of BPP using a hitting set generator. Technical Report TR00-004, Electronic Colloquium on Computational Complexity, January 2000.

[4] V. Guruswami and M. Sudan. List decoding algorithms for certain concatenated codes. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, 2000.

[5] V. Guruswami and M. Sudan. Extensions to the Johnson bound. Manuscript, February 2001.

[6] R. Impagliazzo, V. Kabanets, and A. Wigderson. In search of an easy witness: Exponential time vs. probabilistic polynomial time. In *Sixteenth Annual IEEE Conference on Computational Complexity*, pages 1–12, 2001.

[7] R. Impagliazzo, R. Shaltiel, and A. Wigderson. Near-optimal conversion of hardness into pseudo-randomness. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, 1999.

[8] R. Impagliazzo, R. Shaltiel, and A. Wigderson. Extractors and pseudo-randomn generators with optimal seed-length. In *Proceedings of the Thirty-second Annual ACM Symposium on the Theory of Computing*, 21–23 May 2000.

[9] R. Impagliazzo and A. Wigderson. $P = BPP$ if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 220–229, El Paso, Texas, 4–6 May 1997.

[10] V. Kabanets. Easiness assumptions and hardness tests: Trading time for zero error. In *Fifteenth Annual IEEE Conference on Computational Complexity*, pages 150–157, 2000.

[11] A. R. Klivans and D. van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, 1999.

[12] C. J. Lu. Derandomizing arthur-merlin games under uniform assumptions. In *11th Annual International Symposium on Algorithms And Computation*, pages 302–312, 2000.

[13] P. B. Miltersen and N. V. Vinodchandran. Derandomizing Arthur-Merlin games using hitting sets. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, 1999.

[14] N. Nisan and A. Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, Oct. 1994.

[15] R. Shaltiel and C. Umans. Simple extractors for all min-entropies and a new pseudo-random generator. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science*, 2001.

[16] M. Sudan. Decoding of Reed Solomon codes beyond the error-correction bound. *Journal of Complexity*, 13, 1997.

[17] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the xor lemma. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, 1999.

[18] A. Ta-Shma, D. Zuckerman, and S. Safra. Extractors from Reed-Muller codes. In *Proceedings of the 42th Annual IEEE Symposium on Foundations of Computer Science*, 2001.

[19] L. Trevisan. Construction of extractors using pseudorandom generators. In *Proceedings of the 31st ACM Symposium on Theory of Computing*, 1999.

[20] R. Urbanke. Modern coding theory – SS2001. Technical Report DSC-LTHC, EPFL, 2001. Available from `http://lthcwww.epfl.ch/content.php?title=coding2001`

[21] A. C. Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, Chicago, Illinois, 3–5 Nov. 1982. IEEE.