Electronic Colloquium on Computational Complexity, Revision 3 of Report No. 99 (2019)

Nearly Optimal Pseudorandomness From Hardness

Dean Doron*
Department of Computer Science
Stanford University
ddoron@stanford.edu

Dana Moshkovitz†
Department of Computer Science
University of Texas at Austin
danama@cs.utexas.edu

Justin Oh‡
Department of Computer Science
University of Texas at Austin
sjo@cs.utexas.edu

David Zuckerman§
Department of Computer Science
University of Texas at Austin
diz@utexas.edu

Abstract

Existing proofs that deduce $\mathbf{BPP} = \mathbf{P}$ from circuit lower bounds convert randomized algorithms into deterministic algorithms with a large polynomial slowdown. We convert randomized algorithms into deterministic ones with little slowdown. Specifically, assuming exponential lower bounds against randomized single-valued nondeterministic (SVN) circuits, we convert any randomized algorithm over inputs of length n running in time $t \geq n$ to a deterministic one running in time $t^{2+\alpha}$ for an arbitrarily small constant $\alpha > 0$. Such a slowdown is nearly optimal, as, under complexity-theoretic assumptions, there are problems with an inherent quadratic derandomization slowdown. We also convert any randomized algorithm that errs rarely into a deterministic algorithm having a similar running time (with pre-processing). The latter derandomization result holds under weaker assumptions, of exponential lower bounds against deterministic SVN circuits.

Our results follow from a new, nearly optimal, explicit pseudorandom generator fooling circuits of size s with seed length $(1+\alpha)\log s$, under the assumption that there exists a function $f \in \mathbf{E}$ that requires randomized SVN circuits of size at least $2^{(1-\alpha')n}$, where $\alpha = O(\alpha')$. The construction uses, among other ideas, a new connection between pseudoentropy generators and locally list recoverable codes.

*This work was done while at UT Austin, supported by NSF Grant CCF-1705028.
†Supported in part by NSF Grant CCF-1705028 and CCF-1648712.
‡Supported by NSF Grant CCF-1705028.
§Supported in part by NSF Grant CCF-1705028 and a Simons Investigator Award (#409864).

ISSN 1433-8092

Contents

1 Introduction

1.1 Pseudorandom Generators and Derandomization

Randomized algorithms can outperform deterministic algorithms. We know randomized polynomial time algorithms for problems such as polynomial identity testing, factoring polynomials over large fields, and approximating the number of perfect matchings, where the best known deterministic algorithms take exponential time. For other problems such as primality testing and univariate polynomial identity testing, randomized algorithms offer a polynomial speedup over the best known deterministic ones. Finally, property testing has problems that admit incredibly efficient randomized algorithms [GGR98], in fact sublinear, but provably require linear time deterministically. Informally, randomization owes its success to the prevalence of a seemingly paradoxical phenomenon: most courses of action for an algorithm may be good, yet it might be hard to pinpoint one good course of action.

On the other hand, there are upper bounds on the power of randomization. Assuming plausible circuit lower bounds, any randomized algorithm running in time t on inputs of length n can be simulated deterministically by an algorithm running in time polynomial in t and n [NW94, IW97].[1] In other words, under plausible assumptions, $\mathbf{BPP} = \mathbf{P}$. Concretely:

Theorem 1.1 (previously known derandomization [NW94, IW97])**.** *Assume there exists a constant $\alpha < 1$ and a function $f \in \mathbf{DTIME}\left(2^{O(n)}\right)$ such that f requires circuits of size $2^{\alpha n}$ for all sufficiently large n. Let A be a bounded-error probabilistic algorithm, accepting a language[2] L, that on inputs of length n runs in time $t = t(n)$. Then, there exists a deterministic algorithm that accepts L and runs in time $\mathrm{poly}(t, n)$.*

Previous works [NW94, IW97, SU05, Uma03] proving Theorem 1.1 did not focus on optimizing runtime of the resulting polynomial time deterministic algorithm. Thus for reasons inherent to the proof techniques of the above works, the runtime of the deterministic algorithm of Theorem 1.1 is a large polynomial. Indeed, to the best of our knowledge, the best known runtime of the resulting deterministic algorithm is at least $O(t^8)$ [Uma03]. Our main theorem gives a derandomization with running time $t \cdot \max\{t, n\}^{1+\alpha}$ for an arbitrarily small constant $\alpha > 0$. However, our result relies on a few changes to the statement of Theorem 1.1. The most significant change is that we rely on a somewhat stronger assumption than that of Theorem 1.1, namely we require there are no small *randomized single-valued nondeterministic* circuits for f. Our formal definition of randomized single-value nondeterministic (SVN) circuits in Definition 2.4 has some technical subtleties, but for now it suffices to view them as the nonuniform analogue of $\mathbf{MA} \cap \mathbf{coMA}$: the circuit is allowed to additionally take as input a nondeterministic witness and use randomness to verify the witness. We note that the error of the randomized SVN circuit can be taken to be quasi-polynomially small in n.

Theorem 1.2 (general derandomization, see Theorem 9.4)**.** *Assume there exists a small constant $0 < \alpha < 1$ and a function $f \in \mathbf{DTIME}\left(2^{(1+O(\alpha))n}\right)$ such that f requires randomized single-value nondeterministic circuits of size $2^{(1-\alpha)n}$ for all sufficiently large n. Let A be a bounded-error probabilistic*

[1] On first reading, it may be helpful to just consider the case $t \geq n$.

[2] Here, and throughout the paper, languages (decision problems) can be replaced by other problems, e.g., promise problems or functions with non-binary output. Also, note that we consider algorithms in the Turing machine model. Running times in other models, such as the RAM model, may differ, but one can change the hardness assumptions accordingly.

algorithm, accepting a language L, that on inputs of length n runs in time $t = t(n)$. Then, there exists a deterministic algorithm that accepts L and runs in time $t \cdot \max\{t, n\}^{1+O(\alpha)}$.

There are two main differences between the statements of Theorem 1.1 and Theorem 1.2.

1. While Theorem 1.1 requires $f \in \mathbf{DTIME}\left(2^{O(n)}\right)$ that is hard for circuits of size $2^{\Omega(n)}$, we require $f \in \mathbf{DTIME}\left(2^{(1+O(\alpha))n}\right)$ that is hard for circuits of size $2^{(1-\alpha)n}$. This type of strengthened, plausible, assumption is fundamentally necessary in the hardness vs. randomness paradigm to achieve a low polynomial running time. Moreover, previous proof techniques using our strengthened assumption would still inherently yield a large polynomial running time.

2. We require hardness against randomized single-value nondeterministic circuits, rather than standard ones. The need for this stronger assumption stems from our techniques and will be apparent later on. We discuss this in more detail in Section 1.6.

Our derandomized running time is nearly tight for some of the examples above, such as those from property testing. To further illustrate the tightness, consider the problem of *univariate polynomial identity testing*, in which we test the identity of two arithmetic circuits with one variable, degree n, and $O(n)$ wires, over a field of cardinality $\mathrm{poly}(n)$. Univariate polynomial identity testing is solvable in $\widetilde{O}(n)$ randomized time and $\widetilde{O}(n^2)$ deterministic time. Williams [Wil16] showed that coming up with an $n^{1.999}$-time algorithm, even a nondeterministic one, would refute NSETH[3]. Interestingly, the problem of *multivariate* identity testing also admits a randomized $\widetilde{O}(n)$-time algorithm. Thus, under the complexity-theoretic assumption of Theorem 1.2, and assuming NSETH holds, the time complexity of *both* univariate and multivariate identity testing is settled at roughly quadratic.

While our hardness assumption is not ideal, related complexity-theoretic assumptions were used before to derandomize both deterministic and nondeterministic classes, as well as to construct various pseudorandomness primitives. Specifically, hardness against randomized SVN circuits was used in [GSTS03, SU07], and against (deterministic) SVN circuits, which can be seen as the nonuniform analogue of $\mathbf{NP} \cap \mathbf{coNP}$, in [SU05, MV05]. Hardness assumptions against nondeterministic circuits were used in [AK97, MV05, SU06, SU07, BOV07, Dru13], against circuits with $\mathbf{NP}$ gates in [KvM02, GW02] and even against circuits with $\mathbf{PH}$ gates in [TV00, AIKS16, AASY16, AS17].

Towards proving our main theorem, we first provide a derandomization of algorithms that err rarely, a scenario known as *quantified derandomization*. First studied by Goldreich and Wigderson [GW14], in quantified derandomization, the algorithm is assumed to err only on a small *number* of the possible randomness strings (their error *probability* is extremely small). Our derandomization of such algorithms produces deterministic algorithms whose running time is *similar* to the runtime of the randomized algorithm, up to a quadratic preprocessing step.[4]

Theorem 1.3 (quantified derandomization, see Corollary 5.3)**.** *There exists a constant $c \geq 1$ such that the following holds. Assume there exists a constant $\alpha < 1$ and a function $f \in \mathbf{DTIME}\left(2^{(1+O(\alpha))n}\right)$ such that f requires requires SVN circuits of size $2^{(1-\alpha)n}$ for all sufficiently large n.*

[3]The Nondeterministic Strong Exponential-Time Hypothesis asserts that refuting unsatisfiable k-CNFs requires nondeterministic $2^{n-o(n)}$ time for unbounded k.

[4]The preprocessing step involves encoding the truth table of f. Computing the truth table may take quadratic time, but can be made efficient if f admits fast batch evaluation.

Let A be a bounded-error probabilistic algorithm, accepting a language L, that on inputs of length n runs in time $t = t(n)$ and for every input, errs on at most $2^{t^{1-c\alpha}}$ randomness strings. Then, there exists a deterministic algorithm that accepts L and runs in time $t \cdot \max\{t,n\}^{O(\alpha)} + t_P$, where the $t_P = t^{2+O(\alpha)}$ term corresponds to a step that can be precomputed for all algorithms with running time t.

Unconditional quantified derandomization has been successful for restricted classes of computation, and also sufficiently-good quantified derandomization has been shown to imply circuit lower bounds [GW14, Tel18, Tel19, CT19]. We stress that for quantified derandomization, we only assume hardness against (deterministic) SVN circuits, a seemingly much weaker class.

Like Theorem 1.1, we take the black-box approach for derandomization and prove Theorem 1.2 by constructing a *pseudorandom generator*. A pseudorandom generator (PRG) with error ε is a function that maps a short seed to a t-bit string that is indistinguishable from uniform random bits by any time-t algorithm, up to error ε. We can derandomize an algorithm using a PRG by enumerating over all possible seeds, and running the randomized algorithm on the output of the generator on each seed. The number of possible seeds determines the slowdown[5] of the deterministic algorithm, and this number was $\mathrm{poly}(t,n)$ for a large polynomial prior to this work.

Non-explicitly, there exists a pseudorandom generator against all algorithms running in time t on inputs of length n that uses only $O(\max\{t,n\})$ seeds, but it is not necessarily efficiently computable. In this work we construct an explicit PRG with only $\max\{t,n\}^{1+O(\alpha)}$ seeds.

Theorem 1.4 (pseudorandom generator, see Theorem 10.4)**.** *Assume there exists a function $f \in \mathbf{DTIME}\left(2^{(1+O(\alpha))n}\right)$ that requires randomized SVN circuits of size $2^{(1-\alpha)n}$ for some constant $\alpha < 1$ and all sufficiently large n. Then, there exists an explicit PRG*

$$\overline{G}^f : \{0,1\}^{(1+O(\alpha))\log s} \to \{0,1\}^s$$

with error $\varepsilon = n^{-\Omega(\alpha)}$, fooling circuits of size $s = n^{1-O(\alpha)}$. The PRG is computable in time $\max\{t,n\}^{2+O(\alpha)}$.

We consider f on roughly $\log s$ bits of input, so the truth table of f consists of roughly s bits. The pseudorandom generator converts those s bits of (worst-case) hardness into s bits of pseudorandomness.

Let us compare the parameters of Theorem 1.4 to the prior state-of-the-art PRG given by Umans [Uma03]. There, the seed is of length $c_{\mathsf{U}} \log n$ for a large constant $c_{\mathsf{U}} > 1$, and $s = n^{\gamma_{\mathsf{U}}}$ for a small constant $\gamma_{\mathsf{U}} < 1$. Consequently, derandomization using Umans' PRG would incur a slowdown of at least $s^{c_{\mathsf{U}}/\gamma_{\mathsf{U}}}$ in the running time, whereas in this paper we bring this factor down to $s^{1+O(\alpha)}$. Roughly speaking, we manage to transform almost all the hardness to pseudorandom bits ($s = n^{1-O(\alpha)}$), and we manage to do so using a short seed. The downside of Theorem 1.4 compared to previous works is that we assume f is hard for a seemingly stronger class of circuits.

An important milestone in constructing our PRG $\overline{G}^f$ is the construction of a *pseudoentropy generator* (PEG) with an especially small seed, from which Theorem 1.3, our quantified derandomization, follows. Informally, a distribution over $\{0,1\}^n$ has pseudoentropy k if it is computationally-indistinguishable, up to some error, from *some* high min-entropy distribution. Thus a PEG is a function $\{0,1\}^d \to \{0,1\}^s$ such that the output distribution (when the input is uniform) has high pseudoentropy.

[5] We say a derandomization has slowdown of $S = S(n)$ if $t_{\mathsf{det}}/t_{\mathsf{rand}} = S$, where $t_{\mathsf{det}} = t_{\mathsf{det}}(n)$ and $t_{\mathsf{rand}} = t_{\mathsf{rand}}(n)$ are the running times of the relevant deterministic and randomized algorithms on inputs of length n.

Theorem 1.5 (pseudoentropy generator, see Corollary 10.3)**.** *Assume there exists a function* $f \in \mathbf{DTIME}\left(2^{(1+O(\alpha))n}\right)$ *that requires SVN circuits of size* $2^{(1-\alpha)n}$ *for some constant* $\alpha \leq \frac{1}{9}$ *and all sufficiently large* n*. Then, there is an explicit PEG*

$$G^f \colon \{0,1\}^d \to \{0,1\}^s$$

with error $\varepsilon = n^{-\Omega(\alpha)}$*,* $d = 7\alpha \log n$ *and* $s = \widetilde{O}(n^{1-5\alpha})$*, outputting pseudoentropy* $k = n^{1-9\alpha}$ *fooling circuits of size* s*.*

We mention that pseudoentropy generators are analogous to randomness *condensers* in roughly the sense that pseudorandom generators are analogous to randomness *extractors*, where the hard function plays the role of a high min-entropy source [Tre01, TZ04]. The notion of pseudoentropy and pseudoentropy generators was first considered in [HILL99]. Sudan, Trevisan, and Vadhan [STV01] also construct a pseudoentropy generator under the same notion as in [HILL99]. We construct a pseudoentropy generator under a seemingly weaker definition of pseudoentropy (following [BSW03]) that allows us to get a surprisingly short seed.[6]

In the remainder of the introduction we describe the ideas behind the proofs of Theorem 1.4 and Theorem 1.5.

1.2 Beating the Hybrid Argument

Existing proofs of Theorem 1.1 can be viewed as first constructing a pseudorandom generator that extends its seed by a single bit, and then converting it into a pseudorandom generator that outputs t bits. Interestingly, Sudan, Trevisan, and Vadhan [STV01] constructed PRGs with small seed as in Theorem 1.4 for the single bit case. They did this by using binary locally decodable codes to encode the truth table of f. The pseudorandom generator outputs a random bit of the encoding, and hence the number of seeds corresponds to the length of the encoding. Since there are locally decodable codes of linear length and a sub-linear number of queries (e.g., codes obtained from Reed-Muller composed with Hadamard, or tensor codes [Yek12]), there are single bit PRGs with a small number of seeds.

The large loss in time in Theorem 1.1 originates from the extension of a single bit output to t bits of output. The loss has several manifestations in each of the existing proofs of Theorem 1.1. The use of combinatorial designs in [NW94, IW97] inherently doubles the length of the seed. The use of the Reed-Muller code in [SU05, Uma03] inherently limits the length of the generator's output. Moreover, the analyses of these constructions go through the infamously-hard-to-beat *hybrid argument* [FSUV13]. For t bits to be indistinguishable from uniform, it must be that each bit is unpredictable given the previous bits, and the prediction errors add up across the t bits.

More formally, suppose we wish to show that a distribution X on $\{0,1\}^t$ is ε-indistinguishable from uniform for circuits of size s via the hybrid argument. Proceeding with Yao's next bit predictability argument, we must show that for every $i \in [t]$, no circuit $C \colon \{0,1\}^{i-1} \to \{0,1\}$ of size roughly s can predict X_i with probability greater than $\frac{1}{2} + \frac{\varepsilon}{t}$ when fed with $x \sim X_{[1,i-1]}$. Just as one can bound the seed length of an optimal $G \colon \{0,1\}^\ell \to \{0,1\}^t$ that fools circuits of size s using the probabilistic method, one can calculate the seed length of a nonexplicit $G \colon \{0,1\}^\ell \to \{0,1\}^t$ that is

[6] Indeed, our notion of computational entropy allows the high min-entropy distribution to depend on the distinguishing algorithm. See Section 2.5 for the precise details.

ε-unpredictable at all i-s. The calculation shows that such a G has seed length roughly $\log s + 2\log\frac{1}{\varepsilon}$. Thus, for next bit unpredictability error $\frac{\varepsilon}{t}$, the seed length requires at least an additional $2\log t$ bits.

Hence, in order to prove Theorem 1.4 we must beat the hybrid argument. We do that by first constructing a pseudoentropy generator as in Theorem 1.5, later to be transformed to a pseudorandom generator. Since we no longer require uniform-looking bits, we no longer need an error of $\frac{1}{t}$ per bit. Instead, we output a large number of (imperfect) bits with constant error, which can evidently be done with a small number of seeds. Indeed, Barak, Shaltiel and Wigderson [BSW03] suggested that paradigm, of achieving pseudoentropy as a stepping stone towards pseudorandomness as a way to bypass the weakness of the hybrid argument, and here we fulfill this vision.

1.3 Locally Decodable Codes All the Way Down

As explained above, it was known that single bit pseudorandom generators follow from binary locally decodable codes [STV01]. The construction for obtaining a single bit of pseudorandomness is simple: apply the binary locally decodable code to the truth table of a hard function f, and output a random coordinate of the codeword. However, as discussed above, the method of extending single bit to many bit pseudorandom generators incurs inherent losses in the seed length. In our work we propose a natural deviation from the above idea. We instead apply a code that has *large* alphabet to f and output a random coordinate of this code. The hope is that the symbol at this random coordinate (which, for a large alphabet, will be represented by a large number of bits), will in fact have all the pseudorandom bits we need. However, we cannot guarantee that the random symbol will have "perfect pseudorandomness", i.e. will be computationally close to uniform. Instead, we show that such a random coordinate will have high *pseudoentropy*, i.e will be computationally close to a distribution with at least as much min-entropy as the number of pseudorandom bits we wish to output. In fact, our technique shows that *pseudoentropy generators* that output many bits follow from locally decodable codes over a large alphabet.

More accurately, we consider locally *list recoverable* codes. In a list recoverable code $\mathcal{C} \subseteq \Sigma^n$ for agreement ε, we are given oracle access to lists $S_1, \ldots, S_n \subseteq \Sigma$ for which $\sum_{i=1}^{n} |S_i| \leq \ell$, and we are guaranteed that there are at most L codewords $c \in \mathcal{C}$ satisfying $c_i \in S_i$ for at least ε-fraction of the i-s. We say $\mathcal{C}$ admits *local* list recovery if there exist small circuits $A_1, \ldots, A_L$ with oracle access to the lists $S_1, \ldots, S_n$, each A_i having at most Q oracle gates, such that for every codeword $c = \mathcal{C}(x)$ satisfying $c_i \in S_i$ for at least ε-fraction of the i-s, there exists $j \in [L]$ such that $x = A_j(\cdot)$.[7] Initially, list recoverable codes were used as an intermediate step for constructing list decodable codes (e.g., in [GI01, GI02, GI03]) but have since gained independent interest, with several applications and dedicated constructions (e.g., [HIOS15, HRZW17, HW18, RW18]). Moreover, many of the recent list decoding algorithms are in fact algorithms for list recovery.

We prove that locally list recoverable codes give rise to pseudoentropy generators.

Theorem 1.6 (PEGs from locally list recoverable codes, see Theorem 3.1)**.** *Assume $f \in \{0,1\}^t$ is a truth table of a function requiring SVN circuits of size $t^{1-\alpha}$, and let $\alpha' > \alpha$ be any constant.*

Let $\mathcal{C} \colon \{0,1\}^t \to \Sigma^m$ be a locally list recoverable code for agreement ε and input lists size ℓ, for which each decoding circuit has size $s_{\mathcal{C}} \leq t^{1-\alpha'}$ and makes at most Q oracle queries to the lists. Then, $G^f \colon [m] \to \Sigma$

[7] Unlike in standard literature, where algorithmic aspects are crucial, we do not require (and will not achieve) uniform generation of the A_i-s. Also, each query we make to a list S_i gives us a *single* element of S_i, and we do not get to iterate over the entire list. For the exact definition, refer to Section 2.2.

defined by

$$G^f(z) = \mathcal{C}(f)_z,$$

is a PEG with error ε, outputting $k = \frac{\log \ell}{2}$ pseudoentropy fooling circuits of size $\frac{t^{1-\alpha'}}{Q}$.

In order to get a good PEG, the alphabet Σ of the list recoverable code $\mathcal{C}$ must be large, and the lists size ℓ must be large as well. In fact, they both need to be *exponential* in t, while the decoder should run in time smaller than t. It is atypical to require such a large ℓ, and in standard literature, where the decoder typically iterates over the lists, one requires $\ell \ll m$ in order to get a satisfactory bound on L. We will soon briefly discuss the construction of $\mathcal{C}$ and how we facilitate efficient access to the lists.

We work with multiple definitions of pseudoentropy, which we now discuss. First, we show a close relation between locally list recoverable codes and *Yao pseudoentropy*. Roughly, a random variable $X \sim \{0,1\}^n$ has high Yao pseudoentropy if there is no small, computationally enumerable subset A of $\{0,1\}^n$ that "explains" too much of X, i.e., $\Pr[X \in A]$ is significant. (See Section 2.5.1 for the formal definition.) Now, if we apply a locally list recoverable code with large alphabet to a hard function f and pick a random coordinate, then this random variable must have high Yao pseudoentropy. Otherwise, there would be a small, efficiently computable subset of $\{0,1\}^n$ that explains a significant fraction of the codeword. This roughly corresponds to a small list $S_1 \cup \ldots \cup S_n$ that contains many symbols of the codeword. Hence, f can be efficiently recovered from this list, which is a contradiction.

For the purposes of converting pseudoentropy to pseudorandomness, and other applications, it is convenient to consider other notions of pseudoentropy. The notion of *metric pseudoentropy* was first studied by Barak et al. [BSW03] and later in various works in cryptography [DP08, CKLR11, FR12, Wic13, Sko15, FOR15, SGP15]. We say a random variable $X \sim \{0,1\}^n$ has k metric pseudoentropy fooling circuits of size s, up to ε error, if for every circuit $D\colon \{0,1\}^n \to \{0,1\}$ of size s there exists $Y \sim \{0,1\}^n$ with min-entropy k such that $|\mathbb{E}[D(X)] - \mathbb{E}[D(Y)]| \leq \varepsilon$.[8] Metric pseudoentropy differs from the widely-used notion of HILL pseudoentropy [HILL99] where the high min-entropy random variable does not depend on the distinguishing circuit. Although seemingly weaker, as an additional application, this definition allows us to derandomize algorithms that err rarely, as random variables with high metric pseudoentropy cannot have large weight on small sets (see Section 5 for the details).

We prove Theorem 1.6 by formalizing the above connection between locally list recoverable codes and Yao pseudoentropy [Yao82]. We then utilize the fact that one can get the (more convenient) metric pseudoentropy from Yao pseudoentropy if we allow the enumeration circuit to be an SVN circuit (see Section 2.5.1). As a brief sketch of how to do so, we first mention that if a random variable X has high metric pseudoentropy, then for any efficient distinguisher D, the support of D cannot disproportionately explain X (namely, $\Pr[D(X) = 1]$ is bounded). This is quite close to the definition of Yao pseudoentropy; however, we must give a computationally efficient way to enumerate the support of D. To do so, we follow [BSW03] and hash the support of D onto a smaller universe. The preimage of the hash function is an enumeration of the support of D, provided we can use nondeterminism to guess a preimage (and use D itself to verify whether that preimage is correct). By assuming a function f that is hard for SVN circuits, we get a metric pseudoentropy generator. The rest of the details are given in Section 3. We stress that in our analysis of the PEG, nondeterminism is used only to convert Yao pseudoentropy to metric pseudoentropy. Indeed, our

[8] A random variable $Y \sim \{0,1\}^n$ has min-entropy k if for every $y \in \mathrm{Supp}(Y)$, $\Pr[Y = y] \leq 2^{-k}$.

PEG outputs high Yao pseudoentropy even under the assumption that f is only hard for standard circuits.

It is interesting to draw an analogy between Theorem 1.6 and its information-theoretic counterparts (which we formally define in Section 2.4). The work of Ta-Shma and Zuckerman [TZ04], following Trevisan [Tre01], shows the equivalence between extractors with multiple output bits and *soft-decision decoding*, which we will not define here. Roughly speaking, if such codes are equipped with an efficient local decoding procedure, they give rise to PRGs. Soft-decision decoding generalizes list recoverable codes, and so every extractor can be used to construct a list recoverable code with suitable parameters. We argue that this result *interpolates* for lower min-entropies too. As already observed in [GUV09] under a somewhat different terminology, every list recoverable code for agreement ε gives rise to a condenser condensing $k = \log \frac{L}{\varepsilon}$ min-entropy to $k' = \log \frac{\ell}{m}$ min-entropy with error $O(\varepsilon)$, and each such $k \to k'$ condenser with error ε implies a list recoverable code for agreement $O(\varepsilon)$ with $\ell = O(\varepsilon m 2^{k'})$ and $L = 2^k$. For the formal statement and its proof, see Appendix A. Thus, in a way, Theorem 1.6 is a computational manifestation of these connections.

1.3.1 Constructing the List Recoverable Code $\mathcal{C}$

We construct our locally list recoverable code $\mathcal{C} \colon \{0,1\}^t \to \Sigma^m$ over a large alphabet from locally decodable codes over a small alphabet via folding. Specifically, let

$$\mathcal{C}_{\mathsf{LDC}} \colon \{0,1\}^t \to \mathbb{F}_q^{\bar{t}}$$

be some locally list decodable code having relatively high rate. We partition the $\bar{t}$ coordinates of $\mathcal{C}_{\mathsf{LDC}}$ into m disjoint, contiguous blocks of size a so that $m \cdot a = \bar{t}$ for carefully chosen m and a. Namely, we pick $a = t^{1-O(\alpha)}$. For our final code $\mathcal{C}$, we consider each block of a symbols as a single large symbol. We note that this construction gives a large alphabet size, and also implies that m can be very small, roughly $t^{O(\alpha)}$, which in turn implies that the seed length of our PEG G^f is very small. But why does this construction admit a local list recovery procedure for exponential-sized lists?

Consider the task of decoding some entry $x \in [t]$ of f given lists $S_1, \ldots, S_m$, where we are guaranteed that $\mathcal{C}(f)_z \in S_z$ for at least ε-fraction of the z-s. Recall that $\mathcal{C}_{\mathsf{LDC}}$ is itself equipped with a local list decoding procedure, so in particular there exists a randomized circuit A_{LDC} that locally decodes f from a noisy version of it. How do we mimic a good enough noisy version of f?

- Run A_{LDC} on the input x.
- Whenever A_{LDC} wishes to query a certain coordinate of $\mathcal{C}_{\mathsf{LDC}}(f)$, say coordinate $i \in [\bar{t}]$, find the index z of the block that i is contained in.
- We now want to query the list S_z for the correct value of $\mathcal{C}_{\mathsf{LDC}}(f)_i$, and we use a *hardwired advice* string to pinpoint the specific entry in S_z to be read. From the list's entry we can deduce a guess for $\mathcal{C}_{\mathsf{LDC}}(f)_i$, i.e., a coordinate in the noisy version of f.
- There are at least ε-fraction of good lists for which the advice can point to the the correct symbol $\mathcal{C}(f)_z$. Since a folded symbol in the code $\mathcal{C}$ is correct if and only if every symbol in the block is a correct symbol of $\mathcal{C}_{\mathsf{LDC}}$, we have that A_{LDC} must essentially be making queries to a string with at least ε agreement with $\mathcal{C}_{\mathsf{LDC}}(f)$.

For complete details, see Section 4. We stress that nonuniformity plays a crucial role here, throughout the analysis. Also, since the lists $S_1, \ldots, S_m$ are fixed (and in fact, determined by G^f), and $Q \approx m \ll t$, we are allowed to *fix* the advice describing the pointers to all lists, each pointer is of length roughly $\log \ell$, and we do not have to worry about the possibly different query indices for different x-s and different randomness strings. As our only bound on ℓ stems from the bound on the decoding circuit size, $m \log \ell \approx t$ and so ℓ can be exponentially large.

The above construction works for a large agreement ε (say, a constant or $1/\operatorname{polylog}(n)$), which due to Theorem 1.6 implies that our resulting PEG has constant error. The reason is that we inherit the agreement parameter from the locally decodable code, and there ε must be rather high in order for the other parameters of $\mathcal{C}_{\mathsf{LDC}}$ to be small enough for our setting. In Section 1.5 we discuss how we get the lower error stated in Theorem 1.4 and Theorem 1.5.

1.4 From Pseudoentropy to Pseudorandom Bits

We construct our pseudorandom generator $\overline{G}^f$ from Theorem 1.4 by composing the pseudoentropy generator G^f of Theorem 1.5 with a new construction of an extractor with seed length close to $\log n$, supporting $n^{1-\alpha}$ min-entropy for any $\alpha < \frac{1}{2}$.

Theorem 1.7 (short-seed extractor, see Theorem 7.4)**.** *There exists a constant $c \geq 1$ such that the following holds for any constant $\alpha < \frac{1}{2}$. For every positive integer n and every $\varepsilon \geq cn^{-\frac{1}{2}+\alpha}$, there exists an explicit extractor $\mathsf{Ext}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with error ε, supporting min-entropy $k \geq n^{1-\alpha}$, where $d = (1+c\alpha)\log n + c\log\frac{1}{\varepsilon}$ and $m = \frac{1}{c}n^{1-2\alpha}$.*

This adds to the short list of extractors with almost the right dependence on n, currently including [TZS06, Zuc07] and one-bit extractors coming from good list decodable codes. Our construction essentially follows a construction given in [TZS06]; however, there it was analyzed for smaller min-entropies. The construction and its analysis are given in Section 7.

The seed of the pseudorandom generator consists of the (very short) seed of the pseudoentropy generator, as well as the seed of the extractor, which still gives us $(1 + O(\alpha))\log t$. It is not immediately clear why such a composition works, and indeed we make quite a few modifications for our argument to go through, which we will discuss shortly.

Theorem 1.8 (composition, see Lemma 6.3)**.** *Let $\mathsf{Ext}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ be an extractor with error ε supporting min-entropy $k - \log\frac{1}{\varepsilon}$, computable by a circuit of size s_{Ext}.*

Let $X \sim \{0,1\}^n$ be a random variable with pseudoentropy k fooling circuits of size $s' = O(s + s_{\mathsf{Ext}})$, up to ε error, where we allow the circuits to have oracle gates to the problem of additively approximating the acceptance probability of a given circuit to within some fixed constant accuracy. Then, $\mathsf{Ext}(X, U_d)$ fools (deterministic) circuits of size s, up to error $O(\varepsilon)$.

Note that as is typical with this type of argument, we require the PEG output distribution X to fool circuits larger than s to conclude that the extracted output fools size s.

Previously, composition was known to work when the output of the PEG was indistinguishable from a *universal* distribution with high min entropy. i.e., had high HILL pseudoentropy [BSW03].[9] However, the definition of PEGs that we use allows for the high min-entropy random variable to

[9] A notable example is [STV01], where they show that the Nisan-Wigderson PRG [NW94] outputs pseudoentropy when f is only mildly hard.

depend on the distinguisher, and the natural distinguisher for the composed construction invokes the extractor. In this case, the high min-entropy random variable may depend on a specific seed of the extractor, and some of the seeds of the extractors are destined to fail!

The above obstacle is where our stronger class of distinguishers enters the picture. Barak et al. [BSW03] showed that one can move from HILL pseudoentropy to metric pseudoentropy for real-valued distinguishers. Unfortunately, their argument suffers from a considerable loss in circuits size which we cannot afford, and seems inherent.

The idea to overcome this loss is to use an oracle to approximate some averaging of the extractor on all possible seeds. To be a bit more specific, assume towards a contradiction that $\mathsf{Ext}(X, U_d)$ is distinguishable from the uniform distribution by a small circuit C. We show that in such a case, there exists a large set B of x-s for which $p_x = \mathbb{E}[C(\mathsf{Ext}(x, U_d))]$ is far from $p = \mathbb{E}[C(U_m)]$. The value p can be hardwired. Using our oracle, p_x can be approximated. Thus, B can be recognized by a small circuit. Setting the parameters accordingly, this contradicts the fact that X has high pseudoentropy. We defer the rest of the details to Section 6. In Section 9, we show how the above assumption, of having oracle gates to a density approximation problem, can be replaced with randomized SVN circuits.

1.5 Reducing the Error

Recall that the construction presented in Section 1.3 supported high error. Indeed, for derandomizing **BPP** and for quantified derandomization, a constant error ε for our PEGs and PRGs suffice. Nevertheless, there are several reasons to reduce the error. First, a small error PRG can be used to derandomize algorithms with error approaching $\frac{1}{2}$. Second, often when PRGs are used as components in other pseudorandomness constructions, it is necessary to have small error because of other error losses. Third, it is a natural question on its own, and significant research effort has been devoted to reducing the error in other PRGs. The best error we can hope for is $\varepsilon = n^{-\Omega(1)}$, since we wish to keep the seed length $(1 + O(\alpha)) \log n$. We manage to obtain this.

To get a PEG with a better dependence on ε, we improve the construction of our locally list recoverable code $\mathcal{C}$ to support ε-fraction of good lists even when the underlying locally list decodable code $\mathcal{C}_{\mathsf{LDC}}$ supports only a constant fraction $\varepsilon_0 \gg \varepsilon$ of corrupted coordinates. This is done via an expander-based transformation, and the details are given in Section 10. Getting a PRG with ε error almost readily follows, since the extractor of Theorem 1.7 works for such an error.

1.6 On Our Hardness Assumption

Finally, we discuss our hardness assumption. In order to obtain our result, there are two ways our assumption is stronger than in previous works. First, we require that f must be hard for circuits of size at least $2^{(1-\alpha)n}$ for some small α, whereas previous constructions only required hardness at least $2^{\beta n}$ for some $\beta < 1$ (and generally β is thought of as small). The first strengthening of the assumption seems necessary for a derandomization as efficient as ours since the hardness should be comparable to the size of circuits we wish to fool.

Second, we require a function f that is hard to compute for circuits that are allowed to use both nondeterminism and randomness. One might argue that in the non-uniform model of circuits, one can convert randomness into advice using an Adleman type argument and incur only a minor loss in circuit size. However, the size blowup of Adleman's theorem in the case of nondeterministic circuits is larger. This, combined with the fact that we require the size of our hardness to be

$2^{(1-\alpha)n}$ makes such an argument impossible. Indeed, in order to convert a randomized SVN circuit into an SVN circuit, one must union bound over all possible inputs and all possible witness strings. Generally, the witness length can be nearly as large as the circuit itself. In other words, we would need an error probability close to 2^{-s} where s is the size of the circuit. Achieving that requires repeating the circuit nearly s times, incurring a quadratic blowup and making our hardness assumption infeasible as we initially assume hardness against randomized SVN circuits of size $2^{(1-\alpha)n}$. In summary, previous works have similarly used assumptions in which the function f is hard against models of computation that use nondeterminism and randomness. However, we work on the very "high-end" of the hardness assumptions, i.e., assume hardness for very large circuits, and this seems necessary for our proof.

We now discuss the plausibility of our assumption. Our result relies on the assumption that there exists a function f satisfying two competing properties. First, it should be computable by a deterministic TM in time $2^{(1+\alpha)n}$, for some small constant α. Second, it should also be hard for circuits of size $2^{(1-\alpha)n}$ that use nondeterminism, as well as randomness, to verify the nondeterministic witness. We hence need a function that resides in $\mathbf{E}$ but is still "difficult enough" for a non-uniform Merlin-Arthur proof system to verify (i.e., requiring Arthur to run in time at least $2^{(1-\alpha)n}$). Thus, our assumption is plausible if random verification doesn't always achieve significant savings.

Sometimes, random verification does yield savings. For example, Williams [Wil16] proved that a variety of $\#\mathbf{P}$-complete problems have MA proof systems that run in time 2^{cn} for various $c < 1$, whereas the best known deterministic algorithms solving these problems take time $O(2^n)$. He also gave a 3-round interactive proof system running in time $2^{.67n}$ time for QBF, a $\mathbf{PSPACE}$-complete problem. Even if these results make one worry about finding our desired f in $\mathbf{PSPACE}$, it is widely believed that there exists problems computable in time $2^{(1+\alpha)n}$ that require exponential space. Given that such problems lack the structure that Williams exploits, it is plausible that there is such an f where random verification doesn't achieve significant savings, making our hardness assumption true.

1.7 Open Problems

Some challenges remain to be tackled. We list only a few of them.

1. Can we achieve a nearly linear time quantified derandomization without any preprocessing?
2. Can we achieve similar results assuming f is exponentially-hard for SVN – rather than randomized SVN – circuits?
3. Can we derandomize any algorithm that runs in time t on inputs of length n in time close to tn, as opposed to $t \cdot \max\{t, n\}$? This would match the runtime of Adleman's *nonuniform* derandomization [Adl78]. Note that *for every fixed algorithm* on input size n, independent of its runtime, there exists a pseudorandom generator against this algorithm that has only $O(n)$ seeds.

2 Preliminaries

The density of a set $B \subseteq A$ is $\mu_A(B) = \frac{|B|}{|A|}$ (when A is clear from context, we shall omit it). For a positive integer A, we denote by $[A]$ the set $\{1, \ldots, A\}$. For a set A, by $x \sim A$ we mean

x is drawn uniformly at random from the uniform distribution over the elements of A. For a function $f\colon \Omega_1 \to \Omega_2$, we say f is *explicit* if there exists a deterministic procedure that runs in time $\mathrm{poly}(\log|\Omega_1|)$ and computes f. Finally, for $f, g\colon \mathbb{N} \to \mathbb{N}$, we say that $f(n) = \widetilde{O}(g(n))$ if there exists a constant k such that $f(n) = O(g(n)\log^k g(n))$. All the logarithms are in base 2.

2.1 Circuits, Nondeterministic Circuits and Worst-Case Hardness

The size of a Boolean circuit is the number of its *wires*. We will extensively use the fact that $\mathbf{DTIME}(t(n)) \subseteq \mathbf{SIZE}(O(t(n)\log t(n)))$, where $\mathbf{SIZE}(t(n))$ is the set of languages $L \subseteq \{0,1\}^n$ for which there exists a circuit family $\{C_n\}_n$, such that each C_n is of size $t(n)$, and for every $x \in \{0,1\}^n$ it holds that $x \in L$ if and only if $C_n(x) = 1$ [PF79]. For a Boolean-valued circuit C we denote by $\mathrm{Supp}(C)$ the set of all inputs x for which $C(x) = 1$.

We first define the notion of SVN circuits, where a circuit is allowed to use nondeterminism. However, we allow for the possibility that for some inputs, there are no "good" witnesses that can allow the circuit to compute a "correct answer", i.e. we allow the circuit to compute partial functions.

Definition 2.1 (partial function)**.** *A* partial function *mapping n bits to m bits is a function $f\colon \{0,1\}^n \to \{0,1\}^m \cup \{\perp\}$. We say f is* total *if there is no input $x \in \{0,1\}^n$ for which $f(x) = \perp$.*

We will most often work with total functions. For the rest of this paper, a function $f\colon \{0,1\}^n \to \{0,1\}^m$ with $\perp$ omitted from the range denotes a total function.

Definition 2.2 (SVN circuit, [SU05])**.** *Let $f\colon \{0,1\}^n \to \{0,1\}^m \cup \{\perp\}$ be a partial function. A* single-valued nondeterministic (SVN) circuit computing *f is a pair of circuits $C\colon \{0,1\}^n \times \{0,1\}^w \to \{0,1\}^m$ and $C_{\mathsf{check}}\colon \{0,1\}^n \times \{0,1\}^w \to \{0,1\}$ such that for input string $x \in \{0,1\}^n$ and witness string $y \in \{0,1\}^w$, the following holds.*

1. *For every $x \in \{0,1\}^n$, $f(x) \neq \perp$ if and only if there exists $y \in \{0,1\}^w$ such that $C_{\mathsf{check}}(x,y) = 1$.*
2. *For every $x \in \{0,1\}^n$ and $y \in \{0,1\}^w$ such that $C_{\mathsf{check}}(x,y) = 1$, it holds that $C(x,y) = f(x)$.*

The size of an SVN circuit is the sum of the sizes of C and C_{check}.

When only considering SVN circuits that compute total functions with a single bit of output, the above definition can be viewed as the non-uniform analogue of $\mathbf{NP} \cap \mathbf{coNP}$. We often refer to $C_{\mathsf{check}}(x,y)$ as the checking phase, or witness verification circuit, and refer to $C(x,y)$ as the computing circuit. Additionally, we often think of an SVN circuit as a single circuit combining both C and C_{check} (with $m+1$ output bits).

Definition 2.3 (hardness against SVN circuits)**.** *Let $f\colon \{0,1\}^n \to \{0,1\}$. We let $\mathsf{size}_{SVN}(f)$ denote the size of the smallest SVN circuit that computes f. We say f requires exponential-size SVN circuits if $\mathsf{size}_{SVN}(f) > 2^{\delta n}$ for some constant $\delta > 0$.*

Note that if $f\colon \{0,1\}^n \to \{0,1\}$ is computable by an SVN circuit then $\neg f$ is computable by an SVN circuit of the same size.

We also consider SVN circuits where the checking phase may use randomness. When considering total functions with a single bit output, this model can be seen as the nonuniform analogue of $\mathbf{MA} \cap \mathbf{coMA}$.[10]

[10] An alternative definition for randomized SVN circuits was given in [GSTS03, SU07], corresponding to the nonuniform analogue of $\mathbf{AM} \cap \mathbf{coAM}$. Although one can suspect these models to be equivalent, there are some subtleties that arise when one cares about polynomial blowups in size.

Definition 2.4 (randomized SVN circuits). *Let $f\colon \{0,1\}^n \to \{0,1\}^m \cup \{\perp\}$ be a partial function. A* randomized SVN circuit computing f with error $0 \leq \delta < \frac{1}{2}$ *is a pair of circuits $C\colon \{0,1\}^n \times \{0,1\}^w \to \{0,1\}^m$ and $C_{\mathsf{check}}\colon \{0,1\}^n \times \{0,1\}^w \times \{0,1\}^d \to \{0,1\}$ such that for input string $x \in \{0,1\}^n$, witness string $y \in \{0,1\}^w$ and randomness string $r \in \{0,1\}^d$, the following holds.*

1. *For every $x \in \{0,1\}^n$, $f(x) \neq \perp$ if and only if there exists $y \in \{0,1\}^w$ such that*
$$\Pr_{r \sim U_d}[C_{\mathsf{check}}(x,y,r) = 1] \geq 1 - \delta.$$
2. *For every $x \in \{0,1\}^n$ such that $f(x) \neq \perp$, and $y \in \{0,1\}^w$, either $\Pr_{r\sim U_d}[C_{\mathsf{check}}(x,y,r) = 1] \geq 1 - \delta$ or $\Pr_{r\sim U_d}[C_{\mathsf{check}}(x,y,r) = 1] \leq \delta$.*
3. *For every $x \in \{0,1\}^n$ and $y \in \{0,1\}^w$ such that $\Pr_{r\sim U_d}[C_{\mathsf{check}}(x,y,r) = 1] \geq 1 - \delta$, it holds that $C(x,y) = f(x)$.*

The size of a randomized SVN circuit is the sum of the sizes of C and C_{check}.

We can now define hardness for randomized SVN circuits in the natural way.

Definition 2.5 (hardness for randomized SVN circuits). *We say $f\colon \{0,1\}^n \to \{0,1\}$ requires randomized SVN circuits for error $\delta = \delta(n) > 0$ of size $s = s(n)$ if the smallest randomized SVN circuit that computes f with error δ has size at least s.*

It will be useful to consider a related model where the checking phase is deterministic but has access to an oracle that gets a circuit and approximates the fraction of inputs the circuit accepts.

Definition 2.6 ($\mathrm{DensityApprox}_\eta$ gates). *For a fixed error parameter $\eta > 0$, a $\mathrm{DensityApprox}_\eta$ oracle gate takes as input an encoding of a circuit C and outputs $p \in \{0,1\}^{\lceil \log \frac{1}{\eta} \rceil}$, such that $|\mathbb{E}[C] - p| \leq \eta$. In words, the gate outputs an additive η-approximation to the fraction of accepting inputs to C.*

Definition 2.7 ($\mathrm{DensityApprox}_\eta$ SVN circuits). *Let $f\colon \{0,1\}^n \to \{0,1\}^m \cup \{\perp\}$ be a partial function. For fixed error parameter η, a* $\mathrm{DensityApprox}_\eta$ SVN circuit computing f *is a pair of circuits $C\colon \{0,1\}^n \times \{0,1\}^w \to \{0,1\}^m$ and $C^{\mathsf{DA}}_{\mathsf{check}}\colon \{0,1\}^n \times \{0,1\}^w \to \{0,1\}$ (with the superscript denoting access to $\mathrm{DensityApprox}_\eta$ gates) such that the following holds.*

1. *For every $x \in \{0,1\}^n$, $f(x) \neq \perp$ if and only if there exists $y \in \{0,1\}^w$ such that $C^{\mathsf{DA}}_{\mathsf{check}}(x,y) = 1$.*
2. *For every $x \in \{0,1\}^n$ and $y \in \{0,1\}^w$ such that $C^{\mathsf{DA}}_{\mathsf{check}}(x,y) = 1$, it holds that $C(x,y) = f(x)$.*

The superscript in $C^{\mathsf{DA}}_{\mathsf{check}}$ will be omitted when it is clear from context. The size of a $\mathrm{DensityApprox}_\eta$ SVN circuit is the sum of the sizes of C and $C^{\mathsf{DA}}_{\mathsf{check}}$.

Again we can define hardness against $\mathrm{DensityApprox}_\eta$ SVN circuits in a natural way.

Definition 2.8 (hardness against $\mathrm{DensityApprox}_\eta$ SVN circuits). *Let $f\colon \{0,1\}^n \to \{0,1\}$. We let $\mathsf{size}_{SVN^\star_\eta}(f)$ denote the size of the smallest $\mathrm{DensityApprox}_\eta$ SVN circuit that computes f. We say f requires exponential-size $\mathrm{DensityApprox}_\eta$ SVN circuits if $\mathsf{size}_{SVN^\star_\eta}(f) > 2^{\delta n}$ for some constant $\delta > 0$.*

In Section 9 we prove the equivalence between this model and the randomized SVN model of Definition 2.4.

2.2 Error Correcting Codes

Error correcting codes are families of well-separated strings in Σ^n, called *codewords*. If one corrupts a codeword in some (bounded number) of its coordinates, the codeword is still the closest to the corrupted string. This allows *decoding* of the original codeword.

Definition 2.9 (relative distance)**.** *For some alphabet Σ, let $x, y \in \Sigma^n$. The relative distance of x, y, denoted $\delta(x, y)$, is the fraction of coordinates on which x, y differ. That is, $\delta(x, y) = \Pr_{i \sim [n]}[x_i \neq y_i]$.*

In a *linear code* the codewords form a linear subspace, which is a useful structure.

Definition 2.10 (linear code)**.** *Let n, k, q be positive integers with q a prime power and $0 \leq \delta \leq 1$. We say that $\mathcal{C} \colon \mathbb{F}_q^k \to \mathbb{F}_q^n$ is an $[n, k, \delta]_q$ code if $\mathcal{C}$ is a linear transformation and for all $x, y \in \mathbb{F}_q^k$, $\delta(\mathcal{C}(x), \mathcal{C}(y)) \geq \delta$. By abuse of notation, we often use $\mathcal{C}$ to denote $\mathrm{Im}(\mathcal{C}) \subseteq \Sigma^n$.*

List decoding is the idea of decoding after a large number of corruptions (close to the distance of the code). In this case there can be many, but not too many, codewords closest to the corrupted string. In this work we consider codes over a large alphabet in which the relative distance is close to 1.

Definition 2.11 (list decodable codes)**.** *A code $\mathcal{C} \subseteq \Sigma^n$ is (τ, L) list decodable if for every $w \in \Sigma^n$, $|\{c \in \mathcal{C} : \delta(w, c) \leq \tau\}| \leq L$.*

Next we think of codes as *encoding* $x \in \Sigma^k$ using a codeword in Σ^n (which can be done if there are $|\Sigma|^k$ codewords in the code). The *local decoding* problem asks to only decode x_i for $i \in [k]$ given as input (as opposed to $x \in \Sigma^k$ in its entirety). It should be possible to do that while only inspecting a small number Q of symbols of the corrupted codeword.

Definition 2.12 (locally list decodable code)**.** *Let $\mathcal{C} \colon \Sigma^k \to \Sigma^n$. Let Q, L, s be positive integers and $0 < \varepsilon, \zeta < 1$. We say that $\mathcal{C}$ is $(Q, \varepsilon, \zeta, s, L)$ locally list decodable if there exist randomized circuits $A_1, \ldots, A_L$, each of size s that satisfy the following.*

- *Each A_j has oracle access to a received word $r \in \Sigma^n$, and makes at most Q oracle queries to coordinates in r.*
- *For every codeword $c = \mathcal{C}(x)$ such that c agrees with r in at least ε fraction of the coordinates, there exists $j \in [L]$ such that for every $i \in [k]$, we have $A_j(i) \neq x_i$ with probability at most ζ over the randomness of A_j.*

For convenience, we often say in words that $\mathcal{C}$ is an LLDC using Q queries, handling agreement of at least ε, with failure probability ζ, circuit size s, and output list size L.

In *list recoverable codes*, a variant of list decoding, the decoder is given a *list* of symbols for every coordinate where at least ε fraction of the lists contain the symbol of the encoded message (locally decodable codes correspond to lists of size 1).

Definition 2.13 (list recoverable code)**.** *Let $\mathcal{C} \colon \Sigma^k \to \Sigma^n$. For positive integers ℓ, L and $\varepsilon > 0$ we say that $\mathcal{C}$ is (ε, ℓ, L) list recoverable if the following holds. For every sequence of sets $S_1, \ldots, S_n \subseteq \Sigma$ so that $\sum_{i=1}^n |S_i| \leq \ell$ there are at most L codewords $c \in \mathcal{C}$ satisfying $c_i \in S_i$ for at least ε fraction of the i-s.*

Locally list recoverable codes are analogous to locally list decodable codes. There is a variety of ways to define them, and here we choose a form that is convenient for us.

First, we assume an oracle that given a coordinate $i \in [n]$ and an identifier j returns a unique member of the list of the i-th coordinate identified by j. The oracle may return $\perp$ if the element is undefined. Note that this definition is intended to assert that no matter what indexing scheme one uses to access the lists, the decoder can correctly decode the message using *that* oracle. When the lists are short, the decoder may read the lists in their entirety, however in our context the lists are huge and the decoder can only access individual entries in the lists.

Second, the decoder *may depend on the lists*. In particular, the sizes of the lists, or an identifier of a member of each list, may be *hard-wired into the decoder*. Such unusual definition is common in the context of PRGs [STV01], whereas in standard coding definitions, the same decoder should work for all lists. This definition allows us to assume that the decoder does not make queries whose answer is $\perp$. It also allows us to assume that the decoder is *deterministic*! This is because the number of inputs to the decoder is k, the number of indices in $[k]$ one may wish to decode. By repeating the randomized decoder $O(\log k)$ times, it is possible to decrease the error probability below $\frac{1}{k}$, and therefore there exists a fixed randomness string that leads to correct decoding for all $i \in [k]$ (see [Adl78, STV01] for a more detailed argument). Note that the actual queries the decoder makes depend on i, and vary across $[n]$ when i varies across $[k]$.

Definition 2.14 (locally list recoverable code)**.** *Let $\mathcal{C} \colon \Sigma^k \to \Sigma^n$. For positive integers ℓ, L, Q, and $0 < \varepsilon < 1$ we say that $\mathcal{C}$ is $(Q, \varepsilon, \ell, s)$ locally list recoverable if the following holds. Let $\bar{S} = (S_1, \ldots, S_n)$ be any list of subsets, where each $S_z \subseteq \Sigma$ and $\sum_{z=1}^{n} |S_z| \leq \ell$, and let $\mathcal{O}_{\bar{S}}$ be any oracle to $\bar{S}$ satisfying the following properties:*

- *$\mathcal{O}_{\bar{S}}$ takes as input $y \in \{0,1\}^{\log \ell}$ and outputs an element of $\Sigma \cup \{\perp\}$.*
- *For every $z \in [n]$ and every $v \in S_z$, there exists a $y_{(z,v)} \in \{0,1\}^{\log \ell}$ such that $\mathcal{O}_{\bar{S}}(y_{(z,v)}) = v$. Furthermore, each $y_{(z,v)}$ is unique. That is, $y_{(z,v)} \neq y_{(z',v')}$ if $(z,v) \neq (z',v')$.*

Then there exist circuits $A_1, \ldots, A_L$ for some arbitrary L, each of size s, that satisfy the following.

- *Each A_j takes as input $i \in [k]$ and uses at most Q oracle gates to $\mathcal{O}_{\bar{S}}$.*
- *For every codeword $c = \mathcal{C}(x)$ satisfying $c_z \in S_z$ for at least ε fraction of the i-s, there exists $j \in [L]$ such that for every $i \in [k]$, we have $A_j(i) = x_i$. Moreover, A_j never queries $\mathcal{O}_{\bar{S}}$ on an input that yields $\perp$.*

For convenience, we often say in words that $\mathcal{C}$ is a LLRC using Q queries, handling agreement at least ε, with circuit size s and input list size ℓ.

2.3 Random Variables, Min-Entropy

The *support* of a random variable X distributed over some domain Ω is the set of $x \in \Omega$ for which $\Pr[X = x] \neq 0$, which we denote by $\mathrm{Supp}(X)$.

The *statistical distance* between two random variables X and Y on the same domain Ω is defined as $|X - Y| = \max_{A \subseteq \Omega}(\Pr[X \in A] - \Pr[Y \in A])$. If $|X - Y| \leq \varepsilon$ we say X is ε-close to Y and denote it by $X \approx_\varepsilon Y$. We denote by U_n the random variable distributed uniformly over $\{0,1\}^n$. We say a random variable is *flat* if it is uniform over its support.

For a function $f\colon \Omega_1 \to \Omega_2$ and a random variable X distributed over Ω_1, $f(X)$ is the random variable distributed over Ω_2 obtained by choosing x according to X and computing $f(x)$. For a set $A \subseteq \Omega_1$, $f(A) = \{f(x) : x \in A\}$. For every $f\colon \Omega_1 \to \Omega_2$ and two random variables X and Y distributed over Ω_1 it holds that $|f(X) - f(Y)| \leq |X - Y|$.

The *min-entropy* of a random variable X is defined by

$$H_\infty(X) = \min_{x \in \mathrm{Supp}(X)} \log \frac{1}{\Pr[X = x]}$$

For some $\varepsilon > 0$, we define the *smooth min-entropy* of X by

$$H^\varepsilon_\infty(X) = \max_{X' : X' \approx_\varepsilon X} H_\infty(X').$$

A random variable X is an (n, k) source if X is distributed over $\{0,1\}^n$ and has min-entropy at least k. When n is clear from the context we sometimes omit it and simply say that X is a k-source. Every k-source X can be expressed as a convex combination of flat distributions each with min-entropy at least k.

Definition 2.15 (average conditional min-entropy)**.** *Let X, Y be two random variables. The* average conditional min-entropy *is defined by*

$$\widetilde{H}_\infty(X|Y) = -\log\left(\mathbb{E}_{y \sim Y}\left[2^{-H_\infty(X|Y=y)}\right]\right).$$

Lemma 2.16. *Let X, Y be two random variables such that $|\mathrm{Supp}(Y)| \leq 2^\ell$. Then, $\widetilde{H}_\infty(X|Y) \geq H_\infty(X) - \ell$.*

2.4 Condensers, Extractors, Samplers, and Expanders

Definition 2.17 (extractor)**.** *A function*

$$\mathsf{Ext}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$$

is a (k, ε) seeded extractor if the following holds. For every (n, k) source X, $\mathsf{Ext}(X, Y) \approx_\varepsilon U_m$, where Y is uniformly distributed over $\{0,1\}^d$ and is independent of X. We say Ext *is a strong (k, ε) seeded extractor if $(\mathsf{Ext}(X, Y), Y) \approx_\varepsilon (U_m, Y)$.*

Indeed, whenever a source X has sufficient min-entropy and is independent of the seed Y, we are guaranteed that a strong seeded extractor Ext gives us $(\mathsf{Ext}(X, Y), Y) \approx_\varepsilon (U_m, Y)$. The following lemma shows that $(\mathsf{Ext}(X, Y), Y)$ is nearly independent of any random variable $\mathcal{H}$, such that X, Y are independent conditioned on $\mathcal{H}$ and $X|\mathcal{H}$ has sufficient min-entropy.

Lemma 2.18 ([DORS08, CS16])**.** *Let $\mathsf{Ext}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ be a strong (k, ε) extractor. Let X be distributed over $\{0,1\}^n$ and let $\mathcal{H}$ be some random variable such that $\widetilde{H}_\infty(X|\mathcal{H}) \geq k + \log \frac{1}{\varepsilon}$. Let Y be uniformly distributed over $\{0,1\}^d$, independent of X conditioned on $\mathcal{H}$. Then,*

$$(\mathsf{Ext}(X, Y), Y, \mathcal{H}) \approx_{2\varepsilon} (U_m, Y, \mathcal{H}).$$

Definition 2.19 (condenser). *A function*

$$\mathsf{Cond}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$$

is a (k, k', ε) condenser if the following holds. For every (n, k) source X, $H^{\varepsilon}_{\infty}(\mathsf{Cond}(X, Y)) \geq k'$, where Y is uniformly distributed over $\{0,1\}^d$ and is independent of X. We say Cond *is a strong (k, k', ε) condenser if $(\mathsf{Cond}(X, Y), Y)$ is ε-close to some (D, Y) having min-entropy $d + k'$.*

We proceed by defining *density samplers*.

Definition 2.20 (sampler). *Let $\Gamma\colon [N] \times [D] \to [M]$.*

- *We say $x \in [N]$ is* ε-bad for $B \subseteq [M]$ *if*

$$\left| \Pr_{y \sim U_{[D]}} [\Gamma(x, y) \in B] - \mu(B) \right| > \varepsilon.$$

- *We say Γ is a (δ, ε) sampler if for every $B \subseteq [M]$ we have that*

$$|\{x \in [N] : x \text{ is } \varepsilon\text{-bad for } B\}| < \delta N.$$

Lemma 2.21 ([Zuc97]). *Let* Ext$\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *be a (k, ε) extractor. Then,* Ext *is also a $(\delta = 2^{k-n}, \varepsilon)$ sampler.*

Definition 2.22 (expander graph). *We say that an undirected regular graph G is a λ-expander if all eigenvalues of the normalized adjacency matrix of G other than 1 are at most λ in absolute value.*

The following theorem is implied by the constructions of [GG81, RVW02, MRSV19].

Theorem 2.23. *For every constant $0 < \lambda < 1$ there exists a constant integer $d = d(\lambda)$ such that the following holds. For every positive integer n there exists a connected d-regular undirected graph which is a λ-expander. Given a vertex $x \in [n]$ and an edge label $i \in [d]$, the i-th neighbor of x can be computed in time* $\mathrm{polylog(n)}$.

2.5 Pseudoentropy

Next, we discuss *computational* notions of min-entropy, or, (min-) *pseudoentropy*. We start with the standard, widely used, notion of pseudoentropy due to Håstad et al. [HILL99].

Definition 2.24 (HILL pseudoentropy). *Let X be a random variable distributed over $\{0,1\}^n$, an integer s and $\varepsilon > 0$. We say that $H^{\mathrm{HILL}}_{s,\varepsilon}(X) \geq k$ if there exists a random variable $Y \sim \{0,1\}^n$ with $H_{\infty}(Y) \geq k$ such that for every circuit $D\colon \{0,1\}^n \to \{0,1\}$ of size s, $|\mathbb{E}[D(X)] - \mathbb{E}[D(Y)]| \leq \varepsilon$.*

A weaker notion, given by Reingold [Rei03] and by Barak, Shaltiel and Wigderson [BSW03] allows the random variable having true min-entropy to depend on the distinguisher itself.

Definition 2.25 (metric pseudoentropy). *Let X be a random variable distributed over $\{0,1\}^n$, let s be a positive integer, and $\varepsilon > 0$. We say that $H^{\mathrm{metric}}_{s,\varepsilon}(X) \geq k$ if for every circuit $D\colon \{0,1\}^n \to \{0,1\}$ of size s there exists $Y \sim \{0,1\}^n$ such that $H_{\infty}(Y) \geq k$ and $|\mathbb{E}[D(X)] - \mathbb{E}[D(Y)]| \leq \varepsilon$.*

We record the following standard fact in pseudorandomness.

Claim 2.26. *If $X \sim \{0,1\}^n$ is a random variable such that $H^{\varepsilon}_{\infty}(X) \geq k$ then for every set $D \subseteq \{0,1\}^n$ it holds that $\Pr[X \in D] \leq \frac{|D|}{2^k} + \varepsilon$.*

Barak et al. also give such a result in the computational world, when we use metric pseudoentropy.

Lemma 2.27 ([BSW03])**.** *Let X be a random variable distributed over $\{0,1\}^n$, let s be a positive integer, and $\varepsilon > 0$. Then, $H^{\mathrm{metric}}_{s,\varepsilon}(X) \geq k$ if and only if for every circuit $D\colon \{0,1\}^n \to \{0,1\}$ of size s it holds that $\Pr[D(X) = 1] \leq \frac{|\mathrm{Supp}(D)|}{2^k} + \varepsilon$. The same statement holds when we let D to come from a class of circuits closed under complement, including circuits using $\mathrm{DensityApprox}_{\eta}$ oracle gates.*

2.5.1 Yao Pseudoentropy

A different definition for computational entropy was given by Yao [Yao82] who used *compression* rather than *indistinguishability*.

Definition 2.28 (Yao set)**.** *For some positive integers s, n, ℓ and $A \subseteq \{0,1\}^n$, we say that $A \in \mathcal{C}^{\mathrm{Yao}}_{\ell,s}$ if there exist circuits $c\colon \{0,1\}^n \to \{0,1\}^{\ell}$ and $d\colon \{0,1\}^{\ell} \to \{0,1\}^n$, each of size s, such that*

$$A = \{x : d(c(x)) = x\}\,.$$

We refer to c as the compressing circuit and to d as the decompressing one. We may refer to A in words as the set of properly compressible strings for c and d.

We extend Yao's definition to circuits where the decompressor can be an SVN circuit.

Definition 2.29 (NYao set)**.** *For some positive integers s, n, ℓ and $A \subseteq \{0,1\}^n$, we say that $A \in \mathcal{C}^{\mathrm{NYao}}_{\ell,s}$ if there exists a circuit $c\colon \{0,1\}^n \to \{0,1\}^{\ell}$ and an SVN circuit $d\colon \{0,1\}^{\ell} \times \{0,1\}^{w'} \to \{0,1\}^{\ell}$ computing a partial function $f_d\colon \{0,1\}^{\ell} \to \{0,1\}^n \cup \{\perp\}$, both of size s, such that*

$$A = \{x : f_d(c(x)) = x\}\,.$$

We refer to c as the compressing circuit and to d as the decompressing one.

Definition 2.30 (Yao pseudoentropy and NYao pseudoentropy)**.** *Let X be a random variable distributed over $\{0,1\}^n$, let s be a positive integer, and $\varepsilon > 0$. We say that $H^{\mathrm{Yao}}_{s,\varepsilon}(X) \geq k$ if for every $\ell < k$ and $A \in \mathcal{C}^{\mathrm{Yao}}_{\ell,s}$ it holds that $\Pr[X \in A] \leq 2^{\ell-k} + \varepsilon$.*

We say that $H^{\mathrm{NYao}}_{s,\varepsilon}(X) \geq k$ if we allow $A \in \mathcal{C}^{\mathrm{NYao}}_{\ell,s}$ in the above.

It is not clear whether $H^{\mathrm{Yao}}_{s,\varepsilon}(X) \geq k$ implies $H^{\mathrm{HILL}}_{s',\varepsilon'}(X) \geq k'$ for some suitable k', s', ε'.[11] However, Barak et al. [BSW03] showed that if one is considering hardness against polynomial-sized circuits with NP gates, Yao pseudoentropy does imply metric pseudoentropy. We reprove their result, and here we do it for SVN circuits.

[11] Hsiao, Lu and Reyzin [HLR07] studied *conditional* versions of HILL and Yao pseudoentropies and proved that under these definitions the above implication does not hold. Furthermore, they managed to extract more pseudorandom bits using Yao pseudoentropy based techniques than seems possible from HILL pseudoentropy. Wee gave an oracle under which the Yao pseudoentropy is larger than the HILL pseudoentropy by a factor of roughly 2 [Wee04].

Lemma 2.31 (following [BSW03]). *There exists a constant $0 < \gamma < 1$ such that the following holds. Let X be a random variable distributed over $\{0,1\}^n$ and $\varepsilon > 0$. There exists $s_0 = \widetilde{O}(n)$ such that for every $s \geq s_0$, $H^{\mathrm{NYao}}_{s,\varepsilon}(X) \geq k$ implies $H^{\mathrm{metric}}_{\gamma s,\varepsilon}(X) \geq \frac{k}{2}$.*

Proof: Assume towards a contradiction that $H^{\mathrm{metric}}_{\gamma s,\varepsilon}(X) < \frac{k}{2}$ for some universal constant $\gamma < 1$ to be determined later. Thus, by Lemma 2.27, there exists a circuit $D\colon \{0,1\}^n \to \{0,1\}$ of size γs, with $|\mathrm{Supp}(D)| < 2^{k/2}$, for which

$$\Pr[D(X) = 1] > \frac{|\mathrm{Supp}(D)|}{2^{k/2}} + \varepsilon. \tag{1}$$

Denote $t = \log |\mathrm{Supp}(D)|$, and note that $t < \frac{k}{2}$. Let $\mathcal{H} \subseteq \{0,1\}^n \to \{0,1\}^{2t}$ be a two-universal family of hash functions. We recall that a two-universal family of hash functions $\mathcal{H}$ satisfies that for every distinct $x, y \in \{0,1\}^n$ and every $\sigma_1, \sigma_2 \in \{0,1\}^{2t}$, we have

$$\Pr_{h\sim\mathcal{H}}[h(x) = \sigma_1 \wedge h(y) = \sigma_2] = \left(\frac{1}{2^{2t}}\right)^2.$$

For an efficient implementation of $\mathcal{H}$, we can take it to be the set of all affine functions over $\mathbb{F} = \mathrm{GF}(2^n)$, and so $h(x)$ amounts to computing $ax+b$ over $\mathbb{F}$ for some fixed $a, b \in \mathbb{F}$ and truncating the last $n - 2t$ digits (see, e.g., [Vad12, Theorem 3.26]). Arithmetics in $\mathbb{F}$ can be done by circuits of size $s_0 = \widetilde{O}(n)$.

For a random $h \sim \mathcal{H}$ and distinct $x, y \in \{0,1\}^n$, let $I_{x,y}$ be the indicator random variable which is 1 if and only if $h(x) = h(y)$. By the properties of $\mathcal{H}$,

$$\mathbb{E}\left[\sum_{\{x,y\}\subseteq D} I_{x,y}\right] \leq \binom{2^t}{2} \Pr[I_{x,y} = 1] \leq \binom{2^t}{2} 2^{-2t} < 1,$$

so there exists some $h^\star \in \mathcal{H}$ which is one to one on $\mathrm{Supp}(D)$.

Set $\ell = 2t$, and consider the following compressing circuit and decompressing SVN circuit. First, we let $c\colon \{0,1\}^n \to \{0,1\}^\ell$ simply compute $h^\star$. For the decompressing circuit, we aim for the SVN circuit $d\colon \{0,1\}^\ell \times \{0,1\}^n \to \{0,1\}^n$ to compute the partial function that maps every element in the image of $\mathrm{Supp}(D)$ under h^* to its unique preimage in $\mathrm{Supp}(D)$ (and maps everything else to $\perp$). This can be done by the circuit that simply computes $d(z,x) = x$ and outputs $d_{\mathsf{check}}(z,x) = 1$ if and only if both $D(x) = 1$ and $h^\star(x) = z$.[12] Evaluating c simply requires evaluating $h^\star(x)$ which can be done by a circuit of size $s_0 = \widetilde{O}(n)$. Evaluating d requires both computing $h^\star(x)$ and $D(x)$, and so the size of d is $s_0 + \gamma s$. Now observe that

$$\mathrm{Supp}(D) = \{x : \exists y \in \{0,1\}^n \text{ s.t. } d_{\mathsf{check}}(c(x), y) = 1 \text{ and } d(c(x), y) = x\} \in \mathcal{C}^{\mathrm{NYao}}_{\ell, s_0+\gamma s}.$$

Moreover, recall from Equation (1) that

$$\Pr_{x\sim X}[x \in \mathrm{Supp}(D)] > 2^{t-\frac{k}{2}} + \varepsilon > 2^{\ell-k} + \varepsilon.$$

Since the constant γ can be set so that both c and d are of size s, this contradicts the fact that $H^{\mathrm{NYao}}_{s,\varepsilon}(X) \geq k$. ■

[12] The definition of SVN circuits computing partial functions seems crucial here to make the argument from [BSW03] complete. We wish for d to only output elements of $\mathrm{Supp}(D)$. However, since $\mathrm{Supp}(D)$ is of size 2^t, and the input space for d is of larger size, there are inputs with no well defined answer.

2.6 Pseudoentropy Generators and Pseudorandom Generators

We say that a distribution $X \sim \{0,1\}^n$ *ε-fools* a circuit D with n inputs if $D(X) \approx_\varepsilon D(U_n)$. A pseudorandom generator against a class $\mathcal{C}$ is a function whose output distribution fools any function from $\mathcal{C}$.

Definition 2.32 (PRG)**.** *Let $\mathcal{C} \subseteq \{0,1\}^n \to \{0,1\}$ be a class of functions. We say that $G\colon \{0,1\}^n \to \{0,1\}$ ε-fools $\mathcal{C}$ (or, is an ε-*PRG against *$\mathcal{C}$) if for every $C \in \mathcal{C}$,*

$$|\mathbb{E}[C(G(U_d))] - \mathbb{E}[C(U_n)]| \leq \varepsilon.$$

When $\mathcal{C}$ is the class of functions computable by circuits of size s, we say that G ε-fools circuits of size s.

In this work, we will also construct weaker variants. When the output of a generator is not pseudorandom but does have high pseudoentropy, we say that the generator is a *pseudoentropy generator*. Our pseudoentropy generators will output random variables having high *metric* pseudoentropy.

Definition 2.33 (metric PEG)**.** *We say that $G\colon \{0,1\}^d \to \{0,1\}^n$ is a (k, s, ε) metric pseudoentropy generator (PEG) if*

$$H^{\text{metric}}_{s,\varepsilon}(G(Y)) \geq k,$$

where Y is the uniform distribution over d bits. We say that G is a strong (k, s, ε) metric pseudoentropy generator if $H^{\text{metric}}_{s,\varepsilon}(Y \circ G(Y)) \geq k$.

One can show that for $k = n$ the above definition coincides with the standard definition of PRGs [BRSW12].

3 A Pseudoentropy Generator From Worst-Case Hardness

In this section we show how to construct a metric pseudoentropy generator from a function f that is hard for SVN circuits. The idea is to show that if a code $\mathcal{C}$ is locally list recoverable, then the symbol at a random coordinate of $\mathcal{C}(f)$ has high Yao pseudoentropy, where we identify f with its truth table in $\{0,1\}^n$. This then implies that it must also have high metric pseudoentropy by Lemma 2.31.

Let $f\colon \{0,1\}^{\log n} \to \{0,1\}$ be such that every SVN circuit computing f has size at least $n^{1-\alpha_0}$ for some constant $\alpha_0 < \frac{1}{6}$. That is, $\text{size}_{SVN}(f) \geq n^{1-\alpha_0}$. Let

$$\mathcal{C}\colon \{0,1\}^n \to \Sigma^m$$

be some error correcting code. Define $G^f\colon [m] \to \Sigma \equiv \{0,1\}^{\log|\Sigma|}$ so that

$$G^f(z) = \mathcal{C}(f)_z.$$

Theorem 3.1. *Keeping the above notation, let ε, α be constants such that $\varepsilon > 0$ and $\alpha_0 < \alpha \leq \frac{1}{6}$. Assume $\mathcal{C}$ is $(Q, \varepsilon, \ell, s_{\mathcal{C}})$ locally list recoverable so that $s_{\mathcal{C}} = O(n^{1-\alpha})$, and $\frac{n^{1-\alpha}}{Q} \geq s_0$ for some $s_0 = \widetilde{O}(\log|\Sigma|)$, where the precise $\widetilde{O}(\cdot)$ dependence is determined by Lemma 2.31. Then G^f is a strong (k, s, ε) metric PEG for $k = \frac{\log \ell}{2}$ and $s = O\left(\frac{n^{1-\alpha}}{Q}\right)$.*

Proof: Let Z be the uniform distribution over $[m]$. We first show that

$$H^{\mathrm{NYao}}_{s',\varepsilon}\left(Z \circ G^f(Z)\right) \geq k'$$

for $s' = \frac{n^{1-\alpha}}{Q}$ and $k' = \log \ell$.

Assume towards a contradiction that $H^{\mathrm{NYao}}_{s',\varepsilon}(Z \circ G^f(Z)) < k'$. Then there exists a set $A \subseteq [m] \times \Sigma$, with $A \in \mathcal{C}^{\mathrm{NYao}}_{k',s'}$, such that

$$\Pr[Z \circ G^f(Z) \in A] > \varepsilon.$$

As $A \in \mathcal{C}^{\mathrm{NYao}}_{k',s'}$, there exist a compressing circuit and a decompressing SVN circuit $c\colon [m] \times \Sigma \to \{0,1\}^{k'}$ and $d\colon \{0,1\}^{k'} \times \{0,1\}^{w} \to [m] \times \Sigma$ of size s', with d computing a partial function $f_d\colon \{0,1\}^{k'} \to ([m] \times \Sigma) \cup \{\perp\}$ such that for every $(z,\sigma) \in A$ we have that $f_d(c(z,\sigma)) \neq \perp$ and $f_d(c(z,\sigma)) = (z,\sigma)$.

Consider the sets $S_1, \ldots, S_m \subseteq \Sigma$ defined as follows:

$$S_z = \{\sigma \in \Sigma : (z,\sigma) \in A\}.$$

Notice that $\sum_{z \in [m]} |S_z| \leq 2^{k'} = \ell$. Let $f'_d : \{0,1\}^{k'} \to \Sigma \cup \{\perp\}$ be the partial function that computes f_d but omits the element of $[m]$ from the output. We claim that f'_d behaves exactly like an oracle for $\bar{S} = (S_1, \ldots, S_m)$ as in Definition 2.14. This is because for every $(z,\sigma) \in A$, we know that $f'_d(c(z,\sigma)) = \sigma$. Furthermore, every $c(z,\sigma)$ is unique since c is one to one on A.

Moreover, for at least ε fraction of z-s we know that $C(f)_z \in S_z$. Thus, by considering f'_d as an oracle gate for $\bar{S}$, our definition of locally list recoverable codes implies there exists a circuit $\mathcal{A}$ computing f that is of size at most $s_{\mathcal{C}}$, and uses at most Q oracle queries to f'_d.

We can replace each oracle query in $\mathcal{A}$ with a circuit that computes the SVN decompressor circuit d on the appropriate witness string in $\{0,1\}^w$, and outputs the resulting symbol in Σ. In all, the final SVN circuit will use a witness string in $\{0,1\}^{wQ}$, where each block of w bits serves as the witness to d for one of the Q oracle queries. The resulting circuit will output 1 as the check bit if and only if the check bits of all Q copies of the decompressing circuit d is 1. Since the size of the decompressing circuit is at most s', we have a final circuit of size

$$O(s_{\mathcal{C}} + Q \cdot s') = O(n^{1-\alpha})$$

that computes f, which is a contradiction of the hardness of f.

Knowing that $H^{\mathrm{NYao}}_{s',\varepsilon}\left(Z \circ G^f(Z)\right) \geq k'$, and $s' = \widetilde{\Omega}(\log |\Sigma|)$, applying Lemma 2.31 proves the theorem. ∎

4 A Locally List Recoverable Code From Local List Decoding

In this section, we construct locally recoverable codes useful for constructing a pseudoentropy generator. Theorem 3.1 suggests that in order to construct a metric pseudoentropy generator as in Theorem 1.6, we must construct, for sufficiently small constant α, a locally list recoverable code $\mathcal{C}\colon \{0,1\}^n \to \Sigma^m$ with alphabet size $|\Sigma| = 2^{n^{1-O(\alpha)}}$, that can recover from ε agreement using roughly n^α queries. This gives pseudoentropy against circuits of size $s = n^{1-O(\alpha)}$.

For our ultimate goal of constructing a PRG, we would like the number of pseudorandom bits obtained after extraction to be comparable to the hardness s, and so we would also like the input

lists size ℓ to be approximately $2^{n^{1-O(\alpha)}}$. Expecting the extractor to use $(1+O(\alpha))\log n$ bits for its seed, we can only afford the seed of the PEG to be of length $O(\alpha)\log n$, i.e., $m = n^{O(\alpha)}$.

One way to achieve these parameters is to take a locally list *decodable* code having relatively large rate, namely having blocklength $\widetilde{O}(n)$. Dividing the codeword into roughly n^{β} blocks of $n^{1-\beta}$ symbols each, for $\beta = O(\alpha)$, and treating each block as a single symbol yields the right parameters for m and $|\Sigma|$. We show that such a construction in fact yields the locally list recoverable code we need.[13]

Towards proving correctness of such an approach to construct locally list recoverable codes, we utilize our nonuniform definition of list recovery. Specifically, we construct a circuit that takes as advice a pointer for each list $S_1, \ldots, S_m$, as these are independent of the specific coordinate we wish to decode. These pointers then naturally induce a string for the original locally list decodable code, to which the circuit has oracle access. Thus, the circuit we construct need to only efficiently *list decode* the induced string.[14] Let y be an upper bound on the length of the advice pointer for a given list S_z. To address the fact that the size of the decoding circuit $s_{\mathcal{C}}$ must be at most $n^{1-\alpha}$, we need to satisfy $m \cdot y < n^{1-\alpha}$. This puts an upper bound on y, which implies that we can handle lists of size $\ell = 2^{n^{1-O(\alpha)}}$.

Following [STV01], we employ the list decoding properties of the Reed-Muller code. More specifically, we use the following theorem.

Theorem 4.1 ([STV01]). *Let $\mathcal{C}_{\mathsf{RM}}$ denote the $\left[q^t, \binom{t+d}{d}, 1-\frac{d}{q}\right]_q$ Reed-Muller code. That is, the code consisting of all t-variate polynomials of total degree d over $\mathbb{F}_q$. Then for any constant $1 > \zeta > 0$, $\mathcal{C}_{\mathsf{RM}}$ is locally list decodable using $Q = O_{\zeta}(q^2)$ queries, handling $2\sqrt{2}\sqrt{\frac{d}{q}}$ fraction of agreement, with failure probability ζ, circuit size $s = \mathrm{poly}(t, q)$, and output list size $L = O(q^t)$. For a constant t, one can achieve $s = \widetilde{O}_t(q^4)$.*

To see why one can achieve $s = \widetilde{O}_t(q^4)$, we briefly touch upon the details in [STV01]. The decoding circuit for the Reed-Muller code works by performing list decoding for *univariate* polynomials along q lines in $\mathbb{F}_q^t$. The latter can be done by first solving a system of at most $O(q)$ linear equations in $O(q)$ unknowns over $\mathbb{F}_q$ (which takes time $O(q^3)$) to find a certain bivariate polynomial over $\mathbb{F}_q$. Next, the algorithm finds roots of the bivariate polynomial [Sud97]. Finding such roots requires at most $O(q)$ applications of factoring univariate polynomials, which takes time at most $O(q^2)$ by fastest known results [KU11, ALRS98]. Thus overall, for constant t, the size of the decoding circuit s is at most $\widetilde{O}(q(q^3+q^3)) = \widetilde{O}(q^4)$.

4.1 The Construction

Our construction is a simple "folding" of the Reed-Muller code, where we increase the alphabet size by concatenating the symbols of multiple coordinates into larger symbols. (The folding technique has been widely used in coding theory, ever since Guruswami and Rudra's capacity-achieving list-decoding algorithm for folded Reed-Solomon codes [GR08].)

[13] We note that in order to use Theorem 3.1, we need the hardness s to satisfy $s = \widetilde{\Omega}(\log|\Sigma|)$. We choose our constants below carefully (hidden in the $\beta = O(\alpha)$ above) in order to address this issue.

[14] In the uniform setting, this amount to "trying all possible ℓ^m options". We leave it as an open problem whether we can do substantially better in our regime of parameters.

Given a positive integer n and constants $0 < \varepsilon < 1$ and $0 < \alpha < \frac{1}{6}$, we construct $\mathcal{C} \colon \{0,1\}^n \to \Sigma^m$ that is locally list recoverable by a small circuit using $n^{O(\alpha)}$ queries, with input lists size $\ell = 2^{n^{1-O(\alpha)}}$. The construction goes as follows.

- In Theorem 4.1, set $\zeta = \frac{1}{3}$, $t = \frac{1}{\alpha}$ and $q = \frac{8}{\varepsilon^2} \cdot d$ with d chosen so that $n = \binom{t+d}{d} \log q$. Working out the parameters, one can see that $d = \widetilde{O}_{\varepsilon,\alpha}(n^\alpha)$ and $q = \widetilde{O}_{\varepsilon,\alpha}(n^\alpha)$ (here, the ε, α subscripts hide a $\frac{1}{\alpha}$ and $\frac{1}{\varepsilon^2}$ factor respectively). The resulting code

  $$\mathcal{C}_{\mathsf{RM}} \colon \{0,1\}^n \to \mathbb{F}_q^{\bar{n}=cn}$$

  is a

  $$\left(Q = \widetilde{O}_{\varepsilon,\alpha}(n^{2\alpha}), \varepsilon, \zeta = \frac{1}{3}, s_{\mathsf{RM}} = \widetilde{O}(n^{4\alpha}), L = O(n)\right)$$

  locally list decodable code for $c = \left(\frac{8}{\varepsilon^2 \alpha}\right)^{\frac{1}{\alpha}} \log n$.

- Set $a = n^{1-5\alpha}$, let $m = \frac{\bar{n}}{a} = \widetilde{O}_{\varepsilon,\alpha}(n^{5\alpha})$, and define

  $$\mathcal{C} \colon \{0,1\}^n \to \Sigma^m,$$

  where $\Sigma = \mathbb{F}_q^a$, so that for every $f \in \{0,1\}^n$ and $z \in \{0, \ldots, m-1\}$,

  $$\mathcal{C}(f)_z = \mathcal{C}_{\mathsf{RM}}(f)_{az} \circ \ldots \circ \mathcal{C}_{\mathsf{RM}}(f)_{az+a-1}.$$

Above, for convenience, indices for $\mathcal{C}(f)$ start at 0. In words, we simply divide the codeword $\mathcal{C}_{\mathsf{RM}}(f)$ into m contiguous blocks of size a, and treat each block as a single symbol.[15] Note that the rate of $\mathcal{C}$ is $\frac{n}{\bar{n} \log q} = \Omega_{\varepsilon,\alpha}\left(\frac{1}{\log^2 n}\right)$, and is equivalent to the rate of $\mathcal{C}_{\mathsf{RM}}$ as indeed folding preserves rate.

We note that we do not make use of any special property of the Reed-Muller code, and in fact any efficient locally list decodable code that has relatively high rate and handles arbitrarily small fraction of agreement will work just as well. In particular, we made no further attempt to reduce the alphabet size q or the decoding runtime, which dictates how small we must take α to be and may improve the constant terms in $O(\alpha)$. Moreover, improving the dependence of Q on ε may lead to better results for nonconstant ε. We discuss it further below in Section 4.4.

4.2 Analysis

We can now prove that our code is locally list recoverable. As mentioned above, we construct a circuit that takes as advice a pointer for each list $S_1, \ldots, S_m$ and the pointers then induce a string in Σ^m to which the circuit has oracle access. Thus, given oracle access to $\bar{S} = (S_1, \ldots, S_m)$, we run the local decoder for the Reed-Muller code, and answer its queries by querying the appropriate S_z using the advice pointer. We also need to show that the induced codeword for the Reed-Muller code has at least ε-fraction of correct symbols, which would arise from the fact that we have ε-fraction of good lists for which the advice can point to the correct folded symbol.

[15] For simplicity, we assume that $\bar{n}$ is divisible by a. To address this issue in full technicality, we can simply add at most $a - 1$ dummy symbols to $\mathcal{C}_{\mathsf{RM}}(f)$. This will have negligible effect on the parameters of our list recoverable code.

Theorem 4.2. *For any positive integer n and any constants $0 < \varepsilon < 1$ and $0 < \alpha \leq \frac{1}{6}$, the code $\mathcal{C}$ constructed above is $(Q, \varepsilon, \ell, s_{\mathcal{C}})$ locally list recoverable for $Q = \widetilde{O}(n^{2\alpha})$, $\ell = 2^{n^{1-8\alpha}}$ and $s_{\mathcal{C}} = n^{1-\alpha}$.*

Proof: To describe our local list recovery algorithm, we give a family of circuits with oracle access to $\bar{S}$, indexed by advice, one of which successfully decodes the desired codeword. We describe our decoding algorithm when given the correct advice, which shows that one circuit successfully decodes f.

Towards this end, fix any codeword $\mathcal{C}(f)$ for $f \in \{0,1\}^n$ satisfying $\mathcal{C}(f)_z \in S_z$ for at least ε-fraction of $z \in [m]$. We begin by describing a *randomized* circuit for computing $\mathcal{C}(f)$. Consider

$$A_{adv}^{\mathcal{O}_{\bar{S}}}(x, r)$$

which takes as input $x \in [n]$, uses randomness string r, has access to some fixed oracle $\mathcal{O}_{\bar{S}}$ for lists $\bar{S} = S_1, \ldots, S_m$, and aims to compute f_x. The decoder circuit $A_{adv}^{\mathcal{O}_{\bar{S}}}$ implements the following procedure.

1. We first set up some preliminaries.
   - $A_{adv}^{\mathcal{O}_{\bar{S}}}$ is hardwired with advice $adv = (y_1, \ldots, y_m, \mathsf{RM})$, where each $y_z \in \{0,1\}^{\log \ell}$ points to some list entry in the z-th list. We assume the correct advice is given. Specifically, for each z such that $\mathcal{C}(f)_z \in S_z$, let the correct y_z be such that $\mathcal{O}_{\bar{S}}(y_z) = \mathcal{C}(f)_z$, otherwise let y_z be an arbitrary string such that $\mathcal{O}_{\bar{S}}(y_z) \neq \perp$. Note that the size of the advice is $m \log \ell = \widetilde{O}(n^{1-2\alpha})$, and only depends on f and $\bar{S}$ and not on the input x.
   - The rest of the advice RM points to a decoding circuit for the Reed-Muller code, and we assume the correct circuit is given. That is, let A_{RM} be the randomized circuit that computes f_x for every x, with probability at least $\frac{2}{3}$ when given oracle access to a word in $\mathbb{F}_q^{\bar{n}}$ with at most $1 - \varepsilon$ fraction of errors from $\mathcal{C}_{\mathsf{RM}}(f)$.
2. Let r be the number of random bits required by A_{RM}. Run the decoding circuit $A_{\mathsf{RM}}(x, r)$. For every $j \in [Q]$, supply the answer to the j-th oracle query that A_{RM} makes to some coordinate $i \in [\bar{n}]$ as follows:
   (a) Let $z = \left\lfloor \frac{i}{a} \right\rfloor$, observing that the i-th coordinate of $\mathcal{C}_{\mathsf{RM}}$ is a part of the z-th of $\mathcal{C}$.
   (b) Use the advice string y_z and the oracle access to $\bar{S}$ to query the element $\widetilde{v} = \mathcal{O}_{\bar{S}}(y_z) \in \mathbb{F}_q^a$.
   (c) Return the appropriate symbol $\widetilde{v}_h \in \mathbb{F}_q$ (for $h \in [a]$) that corresponds to the symbol for the i-th coordinate of A_{RM}. That is, return $\widetilde{v}_h$ for $h = i \pmod a$.
3. Return the output of $A_{\mathsf{RM}}(x, r)$.

First, we claim:

Claim 4.3. *$A_{adv}^{\mathcal{O}_{\bar{S}}}$ can be computed by a circuit of size*

$$\widetilde{O}(n^{1-2\alpha})$$

which makes at most $Q = \widetilde{O}(n^{2\alpha})$ oracle queries to $\mathcal{O}_{\bar{S}}$.

Proof: First, the number of queries is at most the number of queries of the local decoding algorithm A_{RM}.

The inputs to the circuit are $x \in \{0,1\}^{\log n}$ and the randomness r. Since Reed-Muller decoding is the only place our algorithm uses randomness, the size r is subsumed by the size of A_{RM}. The advice that we hardwire to the circuit has length $\widetilde{O}(n^{1-2\alpha})$.

Next, the size of A_{RM} itself is $\widetilde{O}(n^{4\alpha})$, and for each of the $Q = \widetilde{O}(n^{2\alpha})$ queries, we perform some simple modular arithmetic to compute the appropriate indices of $\mathcal{C}$ and $\widetilde{v}$. Such arithmetic can be done in $\widetilde{O}(a) = \widetilde{O}(n^{1-5\alpha})$. Overall, the total size of the circuit is

$$\log n + \widetilde{O}(n^{1-2\alpha}) + \widetilde{O}(n^{4\alpha}) + \widetilde{O}(n^{1-5\alpha}) = \widetilde{O}(n^{1-2\alpha})$$

where we used the fact that $\alpha \leq \frac{1}{6}$. ∎

We shall now prove the correctness of $A_{adv}^{\mathcal{O}_{\bar{S}}}$.

Lemma 4.4. *Fix f and oracle $\mathcal{O}_{\bar{S}}$ for $\bar{S} = S_1, \ldots, S_m$ as above. Then for every $x \in [n]$,*

$$\Pr_{r \sim R}\left[A_{adv}^{\mathcal{O}_{\bar{S}}}(x,r) = f(x)\right] \geq \frac{2}{3}.$$

Proof: We show that the queries performed by A_{RM} are essentially queries to a word with agreement at least ε with $\mathcal{C}_{\mathsf{RM}}(f)$. Define $w \in \mathbb{F}_q^{\bar{n}}$ as follows:

$$w_i = \mathcal{O}_{\bar{S}}\left(y_{\lfloor i/a \rfloor}\right)_{i \pmod a}$$

Note that since all advice points to a symbol that is not $\perp$, we indeed have $w \in \mathbb{F}_q^{\bar{n}}$. In words, w is simply the "unfolded" version of the concatenation of all the oracle calls to the y-s. Observe that by construction, A_{RM} essentially queries symbols from w. Let $\mathbf{G}_{\mathsf{RM}}$ be the set of all $i \in [\bar{n}]$ such that $w_i = \mathcal{C}_{\mathsf{RM}}(f)_i$.

Now, by assumption, for at least ε fraction of $z \in [m]$, we have $\mathcal{O}_{\bar{S}}(y_z) = \mathcal{C}(f)_z$. Let $\mathbf{G}$ denote the set of all such z-s. For every $z \in \mathbf{G}$, and every $h \in [a]$, we know that

$$w_{a \cdot z + h} = \mathcal{C}_{\mathsf{RM}}(f)_{a \cdot z + h},$$

thus since $\frac{|\mathbf{G}|}{m} \geq \varepsilon$, we have that $\frac{|\mathbf{G}_{\mathsf{RM}}|}{\bar{n}} \geq \frac{|\mathbf{G}| \cdot a}{m \cdot a} \geq \varepsilon$. ∎

So far we constructed a randomized circuit that computes $\mathcal{C}(f)$ with high probability, so what remains is constructing a deterministic counterpart. This follows from a standard amplification argument. Lemma 4.4 tells us that the randomized circuit $A_{adv}^{\mathcal{O}_{\bar{S}}}$ is incorrect with probability at most $\frac{1}{3}$. We can thus construct a new circuit that makes $M = O(\log n)$ repeated runs of $A_{adv}^{\mathcal{O}_{\bar{S}}}$ with independent randomness and takes the majority vote. By Chernoff, the error probability of the new amplified circuit is at most $\frac{1}{2n}$ and so there is a fixing of the randomness string so that the amplified circuit computes f on all indices. Since the size of the unamplified circuit was $\widetilde{O}(n^{1-2\alpha})$, the final circuit is of size $\widetilde{O}(n^{1-2\alpha} \cdot M) \leq n^{1-\alpha}$. Moreover, the number of queries is $\widetilde{O}(n^{2\alpha} \cdot M) = \widetilde{O}(n^{2\alpha})$. ∎

We finish this section with a calculation of the time required to compute a given coordinate of our code $\mathcal{C}$, which relates directly to the runtime of our final PRGs.

Claim 4.5. *Given $z \in [m]$ and $f \in \{0,1\}^n$, computing $\mathcal{C}(f)_z$ can be done in time $\widetilde{O}(n)$. Moreover, given $f \in \{0,1\}^n$, computing the entire codeword $\mathcal{C}(f)$ can be done in time $\widetilde{O}(n)$ as well.*

Proof: For any z, computing $\mathcal{C}(f)_z$ (as well as computing all of $\mathcal{C}(f)$) only requires computing the Reed-Muller encoding $\mathcal{C}_{\mathsf{RM}}(f)$ at multiple points. To do so, we utilize the following result on fast multivariate interpolation and multipoint evaluation due to van der Hoeven and Schost.

Theorem 4.6 ([VDHS13])**.** *For a t-variate polynomial P over field $\mathbb{F}$, and the values of P on $I \subseteq \mathbb{F}^t$, one can interpolate a representation of P (coefficients in a monomial basis) in time*

$$O(t|I| \log^2 |I| \log\log |I|).$$

Moreover, given such a representation of P, we can evaluate the values of P on arbitrary I in the same amount of time.

Recall that in our setting, $t = \frac{1}{\alpha}$. To compute both $\mathcal{C}(f)_z$ for fixed z (and all of $\mathcal{C}(f)$), we first interpolate the message f to a representation of a polynomial. The interpolation set I is of size $O(n)$. By the above theorem this takes time $\widetilde{O}(n)$. Next, we compute the values of the polynomial on $n^{1-5\alpha}$ points (for fixed z), or $\widetilde{O}(n)$ points (to compute the entire support). In either case, this again requires time $\widetilde{O}(n)$. ∎

4.3 Applying Our Code for Pseudoentropy Generators

Combining Theorem 4.2 with Theorem 3.1 gives the following.

Theorem 4.7. *For any constant $\varepsilon > 0$ and every positive integer n the following holds. Assume*

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

is such that $\mathsf{size}_{SVN}(f) > n^{1-\alpha_0}$ for some constant $\alpha_0 < \frac{1}{7}$. Let α be any constant such that $\alpha_0 < \alpha < \frac{1}{7}$. Then, there exists a function

$$G^f\colon \{0,1\}^d \to \{0,1\}^a$$

that is a (k, s, ε) SVN metric PEG for $k = \frac{n^{1-7\alpha}}{2}$, $s \geq n^{1-4\alpha}$, $d = 5\alpha \log n + O(\log\log n)$ and $a = \widetilde{O}(n^{1-5\alpha})$.

Given oracle access to f, the support of G^f takes $\widetilde{O}(n)$ time to compute. Moreover, if $f \in \mathbf{DTIME}(n^{c_f})$ for some $c_f \geq 1$, the support of G^f can be computed in time $\widetilde{O}(n^{c_f+1})$.

Proof: Let $\mathcal{C}\colon \{0,1\}^n \to \Sigma^m$ with $\Sigma = \mathbb{F}_q^{n^{1-5\alpha}}$, $q = \widetilde{O}(n^\alpha)$ and $m = \widetilde{O}(n^{5\alpha})$, be the $(Q, \varepsilon, \ell, s_{\mathcal{C}})$ locally list recoverable for $Q = \widetilde{O}(n^{2\alpha})$, $\ell = 2^{n^{1-7\alpha}}$ and $s_{\mathcal{C}} = O(n^{1-\alpha})$ that is guaranteed to us by Theorem 4.2. We note that $\frac{n^{1-\alpha}}{Q} = \Omega(n^{1-4\alpha}) \geq \widetilde{O}(\log |\Sigma|) = \widetilde{O}(n^{1-5\alpha})$ for a large enough n. Thus, by applying Theorem 3.1, we get that

$$G^f\colon \{0,1\}^d \to \{0,1\}^a$$

is a (k, s, ε) metric PEG with $d = \log m = 5\alpha \log n + O(\log\log n)$ and $a = \log |\Sigma| = \widetilde{O}(n^{1-5\alpha})$ for $k = \Omega(n^{1-7\alpha})$ and $s = \frac{n^{1-\alpha}}{Q} \geq n^{1-4\alpha}$.

Finally, to compute the support of the PEG, we can first compute f at every point in time $O(n^{c_f+1})$. We then essentially have oracle access to f, and so by Claim 4.5 we can compute the support in time $\widetilde{O}(n)$. Thus, overall the time to compute the support is $\widetilde{O}(n^{c_f+1})$. ∎

Note that an SVN metric PEG is also a metric PEG with the same parameters.

4.4 General Application of Our Paradigm

To conclude this section, we lay out possible settings of parameters for our paradigm for pseudoentropy via list recovery. In the construction of our code we take a locally list decodable code using Q queries and divide it into m blocks of size a so that $m \cdot a = \bar{n}$. Denoting $s^\star$ as the hardness of f, then using the idea of decoding via advice pointers, we get that the amount of pseudoentropy is roughly $\log \ell = \frac{s^\star}{m}$ and the pseudoentropy is for circuits of size roughly $\frac{s^\star}{Q}$. The seed length of a PEG constructed this way is $\log m$, and the output length is a. We chose our specific m and a along with the Reed-Muller code for convenience in nailing down specific details ($s^\star$ slightly larger than a, linear encoding time, efficient decoding time as a specific polynomial in Q, etc.). We note that the entropy rate of our PEG, $\frac{\log \ell}{a}$, is roughly $\frac{s^\star}{ma} = \frac{s^\star}{\bar{n}}$. This means that the pseudoentropy rate is governed by the rate of the LDC, and its relation to the hardness of the message (the hardness of the pseudoentropy, on the other hand, is governed by the number of queries of the LDC).

Other regimes of m and a may also be of interest. We record the following theorem relating all the parameters above.

Theorem 4.8. *For a positive integer n, let $\mathcal{C}\colon \{0,1\}^n \to (\Sigma_{\mathsf{LDC}})^{\overline{n}}$ be a code that is*

$$\left(Q, \varepsilon, \zeta = \frac{1}{3}, s_{\mathsf{LDC}}, L = O(n)\right)$$

locally list decodable. That is, it is decodable from ε agreement, with error probability $\frac{1}{3}$, using Q queries. Let $s^\star = s^\star(n)$ be some function of n such that for every n, there exists a function $f\colon \{0,1\}^{\log n} \to \{0,1\}$ with $\mathsf{size}_{SVN}(f) > s^\star$. Finally, let m, a be positive integers.

There exists s_0, with $s_0 = \widetilde{O}(a \cdot \log|\Sigma_{\mathsf{LDC}}|)$, such that if the following conditions hold,

- $m \geq \frac{1}{\varepsilon}$,
- $m \cdot a = \bar{n}$,
- $s_0 \leq \frac{s^\star}{2Q}$, *and*
- $s_{\mathsf{LDC}} \leq \frac{s^\star}{4}$,

then there exists a function

$$G^f\colon \{0,1\}^{\log m} \to \{0,1\}^{a \cdot \log|\Sigma_{\mathsf{LDC}}|}$$

that is a (k, s, ε) SVN metric PEG for $k = \Theta\left(\frac{s^\star}{m}\right)$ and $s = \Theta\left(\frac{s^\star}{Q}\right)$.

In Section 5.1 we discuss an instantiation of the above theorem with parameters for improved quantified derandomization.

To understand the limits of the above theorem, we consider some extreme settings of parameters. In our general paradigm of splitting an LDC into m blocks of size a, we see that the seed length of the resulting generator is $\log m$, and that the pseudoentropy is $k = \Theta\left(\frac{s^\star}{m}\right)$. This suggests that decreasing m can only improve our result. In the extreme case, we may try to set $m = 1$. This would yield a PEG that requires no seed, and outputs $\Theta(s^\star)$ bits of pseudoentropy. However, examining the analysis, we see this means that for decoding f, we must hardwire as advice a single compressed string that correctly decompresses via some circuit to all of $\mathcal{C}(f)$. In other words, we

get a list recoverable code that can only handle 100% agreement. This yields a trivial error of $\varepsilon = 1$ (and indeed, such a choice does not satisfy the conditions of the above theorem).

The slightly less extreme example of setting $m = \frac{1}{\varepsilon}$ is also problematic, but for a different reason. In this case, we would supposedly get a seed length of $\log \frac{1}{\varepsilon}$, and about $\varepsilon \cdot s^\star$ bits of metric pseudoentropy. However, here the third condition of the theorem, namely that $s_0 \leq \frac{s^\star}{2Q}$, cannot be satisfied. This is because we have $a = \varepsilon \bar{n}$, and so we require $\widetilde{O}(\varepsilon \bar{n} \cdot \log |\Sigma_{\mathsf{LDC}}|) \leq \frac{s^\star}{2Q}$. At the very least, this would require the hardness $s^\star$ to be at least $\Omega(n)$, however any function can be computed by circuits of that size. We still allow such a setting in the theorem to indicate that it may still yield a meaningful result for *Yao pseudoentropy*, as in this case the offended constraint is not needed.

5 Derandomizing Algorithms That Err Rarely

In Theorem 4.7 we proved that G^f is a metric PEG. This is already useful by itself, e.g., since it allows us to derandomize algorithms that err rarely.

Claim 5.1. *Let $C\colon \{0,1\}^n \to \{0,1\}$ be a circuit of size s for which there exists $b \in \{0,1\}$ such that C evaluates to b on at most $B = B(n)$ of its possible inputs. Let $X \sim \{0,1\}^n$ be such that $H^{\mathrm{metric}}_{s,\varepsilon=1/8}(X) \geq k$ for $k \geq \log B + 3$. Then,*

$$\Pr[C(X) = 1 - b] > \frac{1}{2}.$$

Proof: By Lemma 2.27,

$$\Pr[C(X) = b] \leq \frac{B}{2^k} + \frac{1}{8} \leq \frac{1}{4},$$

so the claim follows immediately. ■

Quite surprisingly, since our PEG has a very short seed, we are able to derandomize with almost no slowdown.

Lemma 5.2. *There exists a constant $c \geq 1$ such that the following holds. For positive integers n and $t \geq n$,[16] let $L \subseteq \{0,1\}^n$ and let $A\colon \{0,1\}^n \times \{0,1\}^t \to \{0,1\}$ be a probabilistic algorithm running in time t for which there exists $B = B(t)$ such that for every $x \in \{0,1\}^n$,*

$$\Pr_{y \sim U_t}[A(x,y) \neq L(x)] \leq \frac{B}{2^t}.$$

Let $G\colon \{0,1\}^d \to \{0,1\}^t$ be a $(k, s, \varepsilon = 1/8)$ metric PEG for $s = ct \log t$ and $k \geq \log B + 3$.

Then, there exists a deterministic algorithm $A_{\mathsf{D}}\colon \{0,1\}^n \to \{0,1\}$ that accepts L and runs in time $2^d t + t_P$, where t_P is for computing $\mathrm{Supp}(G)$ and can be precomputed for all algorithms with running time t.

Proof: Fix some $x \in \{0,1\}^n$ and let $C_x\colon \{0,1\}^t \to \{0,1\}$ be the circuit that computes $A(x,\cdot)$, of size s. By our assumption on A, C_x outputs $1 - L(x)$ on at most B of its inputs. By Claim 5.1, we know that

$$\Pr[C_x(G(U)) = L(x)] > \frac{1}{2}.$$

[16] From here onwards, we assume $t \geq n$ only for simplicity. When $t < n$, the circuit that computes $A(x,\cdot)$ is of size $O(n \log n)$ rather than $O(t \log t)$ and s is chosen accordingly.

Hence, the standard way of constructing A_{D} would be to first compute the set

$$I = \left\{ G(z) : z \in \{0,1\}^d \right\},$$

which is independent of C so can be though of as a preprocessing step for all circuits of a certain size, and then run $A(x,y)$ for every $y \in I$. A_{D} would then return the majority vote. ■

Combining Lemma 5.2 and Theorem 4.7, we get the following corollary.

Corollary 5.3. *There exists a constant $\widetilde{c} \geq 1$ such that the following holds for all positive integers n and $t \geq n$. Assume that for every positive integer w there exists a function $f\colon \{0,1\}^w \to \{0,1\}$ computable in deterministic time $2^{c_f w}$ for some $c_f \geq 1$, for which $\mathsf{size}_{SVN}(f) > 2^{(1-\alpha_0)w}$ for some universal constant $\alpha_0 < \frac{1}{\widetilde{c}}$. Fix any α such that $\alpha_0 < \alpha \leq \frac{1}{\widetilde{c}}$.*

Let $L \subseteq \{0,1\}^n$ and let $A\colon \{0,1\}^n \times \{0,1\}^t \to \{0,1\}$ be a probabilistic algorithm running in time t such that for every $x \in \{0,1\}^n$,

$$\Pr_{y \sim U_t}[A(x,y) \neq L(x)] \leq 2^{-t+t^{1-\widetilde{c}\alpha}}.$$

Then, there exists a deterministic algorithm $A_{\mathsf{D}}\colon \{0,1\}^n \to \{0,1\}$ that accepts L and runs in time $t^{1+\widetilde{c}\alpha} + t_P$, where the $t_P = t^{1+c_f+O(\alpha)}$ term corresponds to a step that can be precomputed for all algorithms with running time t.

That is, the derandomization slowdown of every randomized algorithm running in time t which errs with probability at most $2^{-t+t^{1-O(\alpha)}}$, under our complexity-theoretic assumptions, is at most $t^{O(\alpha)}$.

Proof: Set $w = \log(t^{1+c\alpha})$ for $c \geq 1$ soon to be determined, $W = 2^w$, and let $f\colon \{0,1\}^w \to \{0,1\}$ be the guaranteed hard function. Let

$$G^f\colon \{0,1\}^d \to \{0,1\}^t$$

be the $(k, s, \varepsilon = 1/8)$ metric PEG given in Theorem 4.7 with $d \leq 6\alpha \log n$. By Theorem 4.7, we know the output of the PEG is of length $\widetilde{O}(W^{1-5\alpha})$, so we set c such that $\widetilde{O}(W^{1-5\alpha}) = t$ which gives $5 < \frac{5}{1-5\alpha} < c < \frac{6}{1-6\alpha}$, where we assume $\alpha < \frac{1}{6}$. Set $\widetilde{c} = 8c$ and set $B = 2^{t^{1-\widetilde{c}\alpha}}$. Note that

$$k = \frac{W^{1-7\alpha}}{2} = \frac{1}{2} t^{7c\alpha^2} \cdot t^{1-(c+7)\alpha} + 3 \geq \log B + 3.$$

Also,

$$s = W^{1-4\alpha} \geq t^{1+(c-4)\alpha-4c\alpha^2} \geq t^{1+\alpha-4c\alpha^2} = \omega(t \log t),$$

since $4c\alpha^2 = \frac{\widetilde{c}\alpha^2}{2} \leq \frac{\alpha}{2}$. Finally, observe that $d \leq \widetilde{c}\alpha \log n$ and so the corollary follows from Lemma 5.2. ■

5.1 Improved Results from Better Codes and Stronger Assumptions

As mentioned in Section 4.4, our general paradigm allows us to use different LDCs and parameters for stronger results. We show here that a constant rate LDC that uses $n^{o(1)}$ queries, together with an even stronger assumption can give improved results for quantified derandomization. Specifically, we assume a function that is hard for randomized SVN circuits of size $n^{1-o(1)}$ and use the code below due to Kopparty et al., based on multivariate multiplicity codes.

Theorem 5.4 ([KRZSW18], Theorem 6.1). *For any constant $\varepsilon > 0$, there exists a code $\mathcal{C}\colon \{0,1\}^W \to (\Sigma_{\mathsf{LDC}})^{\overline{W}=cW}$ with $|\Sigma_{\mathsf{LDC}}| = O_\varepsilon(1)$ and $c = O_\varepsilon(1)$ that is*

$$\left(Q = \exp\left(O_\varepsilon\left(\log^{\frac{3}{4}}(W)(\log\log(W))^{\frac{1}{4}}\right)\right), \varepsilon, \zeta = \frac{1}{3}, s_{\mathsf{LDC}} = \mathrm{poly}(Q), L = O(W)\right)$$

locally list decodable. That is, it is decodable from ε agreement, with error probability $\frac{1}{3}$, using $Q = W^{o(1)}$ queries. Moreover, $\mathcal{C}$ can be encoded in time $\mathrm{poly}(W)$.

Using the above code, we get an analogue of Corollary 5.3.

Theorem 5.5. *Let w be a positive integer and set $W = 2^w$. There exists an $\alpha = \alpha(w) = o(1)$ satisfying $\alpha \gg \frac{\log w}{w}$ such that if for every w there exists a function $f\colon \{0,1\}^w \to \{0,1\}$ computable in deterministic time $2^{c_f w}$ for some $c_f \geq 1$, for which $\mathsf{size}_{SVN}(f) > 2^{(1-\alpha)w}$, then the following holds.*

For positive integers n and $t \geq n$, let $L \subseteq \{0,1\}^n$ and let $A\colon \{0,1\}^n \times \{0,1\}^t \to \{0,1\}$ be a probabilistic algorithm running in time t such that for every $x \in \{0,1\}^n$,

$$\Pr_{y \sim U_t}[A(x,y) \neq L(x)] \leq 2^{-t+t^{1-5\alpha}}.$$

Then, there exists a deterministic algorithm $A_{\mathsf{D}}\colon \{0,1\}^n \to \{0,1\}$ that accepts L and runs in time $t^{1+o(1)} + t_P$, where the t_P term corresponds to a step that can be precomputed for all algorithms with running time t.

Proof (sketch): Let W be the length of a truth table of a function that is hard for randomized SVN circuits of size $O(W^{1-\alpha})$ for $\alpha = \alpha(W) > 0$ for α soon to be determined. Our general paradigm (see Theorem 4.8) of splitting the LDC into m blocks of size a (for m soon to be determined) gives a metric pseudoentropy generator with

- Pseudoentropy $\Theta(\frac{W^{1-\alpha}}{m})$,
- Hardness $\Theta(\frac{W^{1-\alpha}}{Q})$, and,
- Output length (in bits) $a' = \frac{\overline{W}}{m} \cdot \log|\Sigma_{\mathsf{LDC}}|$.[17]

Recalling that the metric pseudoentropy result only holds if the hardness is at least $\widetilde{O}(a')$, we get the constraint $\frac{W^{1-\alpha}}{Q} \geq \frac{\overline{W}}{m}\log|\Sigma_{\mathsf{LDC}}|w$, or in other words $m \geq O_\varepsilon(1) \cdot QW^\alpha w$. Let α be the exponent (as a function of W) such that $W^\alpha = Q$. That is,

$$\alpha = \Theta_\varepsilon\left(\frac{\log w}{w}\right)^{\frac{1}{4}}.$$

Next, set $m = Q^3 w$. Then, as in Corollary 5.3, in order to fool a circuit $C_x\colon \{0,1\}^t \to \{0,1\}$ of size $O(t\log t)$ that errs only on some small set B, we first set W such that $t = W^{1-3\alpha}$ (note that α is $o(1)$ both as a function of W and as a function of t). This ensures that the output length of the PEG is

$$\frac{\overline{W}}{Q^3 w}\log|\Sigma_{\mathsf{LDC}}| \leq W^{1-3\alpha} = t.$$

[17] The size of the decoding circuit must also be smaller than $W^{1-\alpha}$. Since the size of the decoding circuit is $\mathrm{poly}(Q)$, and $Q = W^{o(1)}$, we have that $W^{o(1)} < W^{1-\alpha}$ if α is also $o(1)$.

Padding the length so that it is exactly t yields a distribution with pseudoentropy

$$\frac{W^{1-\alpha}}{Q^3 w} \geq W^{1-5\alpha} = t^{\frac{1-5\alpha}{1-3\alpha}} \geq t^{1-5\alpha}$$

fooling circuits of size

$$\frac{W^{1-\alpha}}{Q} = W^{1-2\alpha} = t^{\frac{1-2\alpha}{1-3\alpha}} \geq t^{(1-2\alpha)(1+3\alpha)} \geq t^{1+O(\alpha)} \geq \omega(t \log t),$$

where the last inequality follows from the fact that $Q \geq \log t$. Thus, we can in indeed fool the circuit C_x provided the set B is such that $\log B + 3 \leq t^{1-o(1)}$. ■

We mention that we can also use multiplicity codes in our original construction, however, no parameters besides the hardness of the PEG output will improve unless we also assume a function which is hard for circuits of size $n^{1-o(1)}$ as above. We used Reed-Muller mainly for simplicity.

6 Extracting Randomness from Pseudoentropy

To derandomize general algorithms, and not just algorithms that err rarely, we need to convert pseudoentropy to pseudorandomness (i.e., to bits that are indistinguishable from uniform). When the pseudoentropy is HILL pseudoentropy one can do this conversion using an extractor [BSW03]. In this section we show how to use an extractor *even when the pseudoentropy is metric*, by requiring indistinguishability by $\mathrm{DensityApprox}_\eta$ SVN circuits (see Definition 2.7). A more detailed explanation follows.

Let X be a distribution over $\{0,1\}^n$ whose pseudoentropy is at least k. Let $\mathsf{Ext}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ be an extractor, and let $y \in \{0,1\}^d$ be a typical seed for the extractor. If X has high HILL pseudoentropy then there exists a random variable X_0 over $\{0,1\}^n$ whose min entropy is at least k, and X_0 is indistinguishable from X. If $\mathsf{Ext}(X,y)$ were not indistinguishable from uniform, then there would have been a distinguisher $C\colon \{0,1\}^m \to \{0,1\}$ that distinguishes $\mathsf{Ext}(X,y)$ from uniform. But then $D(x) = C(\mathsf{Ext}(x,y))$ is a distinguisher that distinguishes X from X_0 (because $\mathsf{Ext}(X_0,y)$ is nearly uniform for almost all y-s). Hence, $\mathsf{Ext}(X,y)$ must be pseudorandom when X has HILL pseudoentropy.

If X only has metric pseudoentropy, there no longer exists a canonical X_0. Instead, for every distinguisher $D\colon \{0,1\}^n \to \{0,1\}$ for X, there is a different variable X_D of min entropy k that is indistinguishable for D. When $D(x) = C(\mathsf{Ext}(x,y))$, the distinguisher, and hence the source of the extractor, depend on the seed y, and so we have no guarantee on the output of the extractor.

To circumvent this problem, we give a different distinguisher that estimates $\mathbb{E}[C(\mathsf{Ext}(x,U_d))]$ rather than use the actual seed y. We remark that this yields a real-valued (rather than Boolean) distinguisher. However, we can in fact still use a simple Boolean distinguisher: hard-wire $\mathbb{E}[C(U_m)]$ and output 1 iff $\mathbb{E}[C(\mathsf{Ext}(x,U_d))]$ deviates from $\mathbb{E}[C(U_m)]$ (using $\mathrm{DensityApprox}_\eta$ to estimate the former).

Several comments are due:

- Barak et al. [BSW03] show how one can get high HILL pseudoentropy from a random variable having high metric pseudoentropy, but their argument has two disadvantages. First, it loses quite a bit in parameters, which we cannot afford here. Second, it implicitly assumes

metric pseudoentropy for real-valued distinguishers. Unlike HILL pseudoentropy, different notions of metric pseudoentropy (randomized/deterministic distinguishers, $\{0,1\}/[0,1]$-valued distinguishers) do not seem to be equivalent. A number of works address this issue explicitly and distinguish between each notion of metric pseudoentropy (see, e.g., [CKLR11, FR12, FOR15, SGP15]).

- Barak et al. also show how to extract from Yao pseudoentropy, but only if the extractor is *reconstructive*. Although our pseudoentropy generator from Section 3 does output bits with high Yao pseudoentropy, we do not know of a construction of a reconstructive extractor having nearly-optimal seed length.

Recall the definition of $\mathrm{DensityApprox}_\eta$ gates from Definition 2.6. We now formally define metric* pseudoentropy.

Definition 6.1 (metric* pseudoentropy)**.** *Let X be a random variable distributed over $\{0,1\}^n$, let s be a positive integer and $\eta, \varepsilon > 0$. We say that $H^{\mathrm{metric}^\star_\eta}_{s,\varepsilon}(X) \geq k$ if for every circuit D of size s over inputs of length n, having oracle gates to $\mathrm{DensityApprox}_\eta$, there exists $Y \sim \{0,1\}^n$ such that $H_\infty(Y) \geq k$ and $|\mathbb{E}[D(X)] - \mathbb{E}[D(Y)]| \leq \varepsilon$.*

For our purposes, taking $\eta = \varepsilon$ will suffice, in which case we denote $H^{\mathrm{metric}^\star_\varepsilon}_{s,\varepsilon}(X) = H^{\mathrm{metric}^\star}_{s,\varepsilon}(X)$.

We can then naturally define the notion of a metric* pseudoentropy generator.

Definition 6.2 (metric* PEG)**.** *For integers k, s, and error parameters ε, η, we say that $G\colon \{0,1\}^d \to \{0,1\}^n$ is a (k, s, ε) $\mathrm{metric}^\star_\eta$ pseudoentropy generator (PEG) if*

$$H^{\mathrm{metric}^\star_\eta}_{s,\varepsilon}(G(Y)) \geq k,$$

where Y is the uniform distribution over d bits. We say that G is a strong (k, s, ε) $\mathrm{metric}^\star_\eta$ pseudoentropy generator if $H^{\mathrm{metric}^\star_\eta}_{s,\varepsilon}(Y \circ G(Y)) \geq k$.

For our purposes, taking $\eta = \varepsilon$ will suffice, in which case we simply say G is a (k, s, ε) metric* pseudoentropy generator (PEG).

We now prove that one can extract pseudorandomness from metric* pseudoentropy.

Lemma 6.3. *For all positive integers s, t, n, k, d, m and a constant $\varepsilon > 0$ the following holds. Suppose $\mathsf{Ext}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a $(k - \log\frac{1}{\varepsilon}, \varepsilon)$ extractor so that for every $x \in \{0,1\}^n$, $\mathsf{Ext}(x, \cdot)$ is computable by a circuit of size t. Let X be a random variable distributed over $\{0,1\}^n$ so that $H^{\mathrm{metric}^\star}_{s',\varepsilon}(X) \geq k$ for $s' = \widetilde{O}(s + t)$. Then, $\mathsf{Ext}(X, U_d)$ 10ε-fools circuits of size s.*

Proof: Towards a contradiction, assume there exists a distinguisher $C\colon \{0,1\}^m \to \{0,1\}$ of size s so that

$$|\mathbb{E}[C(\mathsf{Ext}(X, U_d))] - \mathbb{E}[C(U_m)]| > 10\varepsilon.$$

We define $D_C\colon \{0,1\}^n \to \{0,1\}$, a circuit having a single $\mathrm{DensityApprox}_\varepsilon$ gate as follows:

- D_C is hardwired with $p = \mathbb{E}[C(U_m)]$, truncated to $\lceil \log\frac{1}{\varepsilon} \rceil$ bits, denoted by $\widetilde{p}$.
- On input x, D_C computes the description of the circuit for $C(\mathsf{Ext}(x, \cdot))$, having size $s + t$.

- The latter description is fed to the $\mathrm{DensityApprox}_{\varepsilon}$ gate, let $\widetilde{p}_x$ be the gate's output.
- D_C returns 1 iff $|\widetilde{p} - \widetilde{p}_x| > 3\varepsilon$.

The size of D_C is $\widetilde{O}(s+t)$.

Let $p_x = \mathbb{E}[C(\mathsf{Ext}(x, U_d))]$. By our assumption on C, we have that $\mathbb{E}_{x\sim X}|p_x - p| > 10\varepsilon$, so there exists a set $B \subseteq \{0,1\}^n$ satisfying $\Pr[X \in B] > 5\varepsilon$ so that for every $x \in B$ it holds that $|p_x - p| > 5\varepsilon$, and so also $|\widetilde{p}_x - \widetilde{p}| > 5\varepsilon - 2\varepsilon = 3\varepsilon$ (where we have 2ε instead of ε due to the truncation of $\widetilde{p}$ above). Hence, $\mathbb{E}[D_C(X)] > 5\varepsilon$.

Now, by Lemma 2.21, we know that

$$|\{x \in \{0,1\}^n : |p_x - p| > \varepsilon\}| < \varepsilon \cdot 2^k$$

and so $|\mathrm{Supp}(D_C)| < \varepsilon \cdot 2^k$. Altogether,

$$\Pr[D_C(X) = 1] - \frac{|\mathrm{Supp}(D_C)|}{2^k} > \varepsilon,$$

contradicting the fact that $H^{\mathrm{metric}^\star}_{s',\varepsilon}(X) \geq k$. ∎

7 Extractors With Near-Optimal Seed Length

We give an explicit strong seeded extractor with near-optimal (in n) seed that supports min-entropy $n^{1-\alpha}$ for every $\alpha < \frac{1}{2}$. Specifically, for $\varepsilon = n^{-o(1)}$, our seed length is $(1 + O(\alpha)) \log n$, which gives a left-degree of $n^{1+O(\alpha)}$. A very similar construction was given in [TZS06], but there the analysis was only done for a constant entropy rate. For our construction we will use three ingredients given in the following theorems.

The first theorem, due to Ta-Shma, Zuckerman and Safra gives an extractor that achieves seed length very close to $\log n$; however, it only outputs a small portion of the min-entropy.

Theorem 7.1 ([TZS06]). *For all positive integers n, k, m and $\varepsilon \leq \frac{1}{2}$ such that $3m\sqrt{n}\log\frac{n}{\varepsilon} \leq k \leq n$, there exists an explicit strong (k,ε) extractor $\mathsf{TZS}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^{\Omega(m)}$ for $d = \log n + O(\log\frac{1}{\varepsilon}) + O(\log m)$.*

The next extractor is Trevisan's, with improved analysis due to Raz, Reingold and Vadhan. We use it to output a constant fraction of the min-entropy.

Theorem 7.2 ([Tre01, RRV02]). *There exists a constant $c_{\mathsf{Tre}} \geq 1$ such that following holds. For all positive integers n, k and $\varepsilon > 0$ there exists an explicit strong (k,ε) extractor $\mathsf{Tre}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ for $d = c_{\mathsf{Tre}}\log^2\frac{n}{\varepsilon}$ and $m = \frac{k}{4}$.*

Lastly, we will need the following condenser by Reingold, Shaltiel and Wigderson whose purpose is to output a shorter string than the input source with the same entropy rate, using a very short seed.

Theorem 7.3 ([RSW06]). *There exist constants $c_{\mathsf{RSW}} \geq 1$ and $0 < \gamma_{\mathsf{RSW}} < 1$ such that the following holds. For all positive integers n, k, r and $\varepsilon \geq c_{\mathsf{RSW}}\sqrt{\frac{r}{k}}$ there exists an explicit strong $(k, k' = \gamma_{\mathsf{RSW}}\frac{k}{r}, \varepsilon)$ condenser $\mathsf{RSW}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ for $d = c_{\mathsf{RSW}}(\log\log n + \log r + \log\frac{1}{\varepsilon})$ and $m = \frac{n}{r}$.*

Our construction, following [TZS06], proceeds along a familiar paradigm. The extractor TZS uses a short seed but extracts relatively few bits, whereas the extractor Tre outputs a constant fraction of the min-entropy but requires a long seed. We are able to utilize both advantages without suffering the drawbacks by using RSW, with a very short seed, to condense the source X. The output length of RSW will be only half the min-entropy, enabling us to use TZS to extract (from the same X) a seed for Tre. Namely, our extractor outputs $\mathsf{Ext}(X, Y_1 \circ Y_2) = \mathsf{Tre}(\mathsf{RSW}(X, Y_1), \mathsf{TZS}(X, Y_2))$, where Y_1 and Y_2 are independent uniform seeds. Note that $\mathsf{RSW}(X, Y_1)$ and $\mathsf{TZS}(X, Y_2)$ can be dependent, however we will argue that they are *close* to being independent, and therefore we can apply Tre.

We are now ready for the details. We are given a positive integer n, $\varepsilon > 0$, some constant $\alpha < \frac{1}{2}$ and $k \geq n^{1-\alpha}$. As promised, we will make use of the following ingredients.

- The strong $(k, \gamma_{\mathsf{RSW}}\frac{k}{r}, \varepsilon)$ condenser $\mathsf{RSW}\colon \{0,1\}^n \times \{0,1\}^{d_1} \to \{0,1\}^{m_1}$ for $r = 2n^\alpha$, where $m_1 = \frac{1}{2}n^{1-\alpha}$ and $d_1 = c_{\mathsf{RSW}}(\log\log n + \log(2n^\alpha) + \log\frac{1}{\varepsilon})$. We require $\varepsilon \geq c_{\mathsf{RSW}}\sqrt{\frac{r}{k}} = \Omega\left(\frac{1}{n^{\frac{1}{2}-\alpha}}\right)$.
- The strong (k', ε) extractor $\mathsf{TZS}\colon \{0,1\}^n \times \{0,1\}^{d_2} \to \{0,1\}^{d_3}$ where $k' = \frac{1}{4}n^{1-\alpha}$, $d_3 = c_{\mathsf{Tre}}\log^2\frac{m_1}{\varepsilon} = O(\log^2\frac{n}{\varepsilon})$ and $d_2 = \log n + O(\log\frac{1}{\varepsilon}) + O(\log d_3)$. Note that indeed k' is large enough.
- The strong (k'', ε) extractor $\mathsf{Tre}\colon \{0,1\}^{m_1} \times \{0,1\}^{d_3} \to \{0,1\}^m$ for $k'' = \frac{\gamma_{\mathsf{RSW}}}{2}n^{1-2\alpha}$ and $m = \frac{k''}{4} = \frac{\gamma_{\mathsf{RSW}}}{8}n^{1-2\alpha}$.

Given $x \in \{0,1\}^n$, $y_1 \in \{0,1\}^{d_1}$ and $y_2 \in \{0,1\}^{d_2}$, we output

$$\mathsf{Ext}(x, y_1 \circ y_2) = \mathsf{Tre}(\mathsf{RSW}(x, y_1), \mathsf{TZS}(x, y_2)).$$

Theorem 7.4. *There exists a constant $c \geq 1$ such that the following holds for any constant $\alpha < \frac{1}{2}$. The function $\mathsf{Ext}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ above is an explicit strong (k, ε) extractor for every positive integer n, $k \geq n^{1-\alpha}$ and $\varepsilon \geq cn^{-\frac{1}{2}+\alpha}$, where $d = (1 + c\alpha)\log n + c\log(\frac{1}{\varepsilon})$ and $m = \frac{1}{c}n^{1-2\alpha}$.*

Proof: The proof will mimic block-source extraction techniques, but is self-contained. Denote $A = \mathsf{RSW}(X, Y_1)$ and $B = \mathsf{TZS}(X, Y_2)$. By the properties of TZS, we know that

$$(B, Y_2) \approx_\varepsilon U_{d_3} \times Y_2.$$

By the properties of RSW, there exists A', which is ε-close to A, for which $H_\infty(Y_1 \circ A') \geq k'' + d_1$ (note that A' depends on Y_1). By Lemma 2.16, and since X and Y_1 are independent,

$$\widetilde{H}_\infty(X|Y_1 \circ A') \geq H_\infty(X) - m_1 = \frac{1}{2}n^{1-\alpha} \geq \frac{1}{4}n^{1-\alpha} + \log\frac{1}{\varepsilon}.$$

Since X and Y_2 are independent of each other and of Y_1, and since A' is independent of Y_2, we know that X, Y_2 are also independent conditioned on $Y_1 \circ A'$. Applying Lemma 2.18 on the strong extractor TZS we get that

$$(A', B, Y_1, Y_2) \approx_{2\varepsilon} (A', U_{d_3}, Y_1, Y_2).$$

Applying Tre, we deduce that

$$(\mathsf{Tre}(A', B), Y_1, Y_2) \approx_{2\varepsilon} (\mathsf{Tre}(A', U_{d_3}), Y_1, Y_2).$$

Now, $\mathsf{Tre}(A', U_{d_3}) \approx_\varepsilon U_m$ so by the triangle inequality,

$$(\mathsf{Tre}(A', B), Y_1, Y_2) \approx_{3\varepsilon} (U_m, Y_1, Y_2).$$

Accounting for the distance between A and A', we finally get that

$$(\mathsf{Ext}(X, Y_1 \circ Y_2), Y) \approx_{4\varepsilon} (U_m, Y),$$

as desired. To conclude, observe that

$$d = d_1 + d_2 = (1 + O(\alpha)) \log n + O\left(\log \frac{1}{\varepsilon}\right).$$

The explicitness follows from the explicitness of each component. ■

7.1 Computing the Extractor by a Small Circuit

Usually, when constructing extractors, we require the computation to be done in polynomial time or in linear space. In this work, we must exercise a more fine-grained analysis, since the size it takes to compute Ext corresponds to the minimal circuit size we will be able to fool with a nearly-optimal small slowdown (see Lemma 6.3 for the precise parameters). We prove that there exists a circuit of nearly-linear size that computes Ext. But before doing so, we will need to argue that a few basic pseudorandomness primitives are also computable in nearly-linear size.

7.1.1 Efficient Asymptotically-Good Codes

An asymptotically-good code is a family of binary codes having a constant rate and a constant relative distance. An example is Justesen's code [Jus72].

Lemma 7.5 (following [Jus72])**.** *There exists a constant $0 < \delta < 1$ such that the following holds. For every positive integer n there exists an explicit $[\bar{n}, n, \delta]_2$ code $\mathcal{C}_{\mathsf{Jus}} \colon \{0,1\}^n \to \{0,1\}^{\bar{n}}$ of rate $\frac{n}{\bar{n}} \geq \delta$.*

Moreover, for every positive integer n there exists a circuit C_n of size $\widetilde{O}(n)$ so that for every $x \in \{0,1\}^n$, $C_n(x)$ computes $\mathcal{C}_{\mathsf{Jus}}(x)$.

Proof: The code is constructed as follows. For some $\bar{n} = O(n)$ and $m = O(\log n)$, let $\mathbb{F}_q$ be a finite field of cardinality $q = 2^m$. Set $k = \frac{n}{m}$, $\bar{k} = \frac{\bar{n}}{2m}$ and let $\{\alpha_1, \ldots, \alpha_{\bar{k}}\}$ be some distinct elements of $\mathbb{F}_q^\star$. Given $x \in \{0,1\}^n \equiv \mathbb{F}_q^k$, let p_x be the univariate polynomial $p_x(\alpha) = \sum_{i=0}^{k-1} x_i \alpha^i$. Then, the encoding is given by

$$\mathcal{C}_{\mathsf{Jus}}(x) = ((p_x(\alpha_1), \alpha_1 p_x(\alpha_1)), \ldots, (p_x(\alpha_{\bar{n}}), \alpha_{\bar{n}} p_x(\alpha_{\bar{n}}))) \in \{0,1\}^{\bar{k}},$$

where we interpreted each element of $\mathbb{F}_1$ as a string in $\{0,1\}^m$. The fact that the Justesen's code is asymptotically good (i.e., that one can take such parameters and maintain a constant relative distance) is by now standard [Jus72, GRS19], but we still need to justify that fact that the encoding can be done via a small circuit.

For simplicity, consider hardwiring $\{\alpha_1, \ldots, \alpha_{\bar{k}}\}$, which takes $m\bar{k} + O(1) = O(n)$ bits. Each bit of $\mathcal{C}_{\mathsf{Jus}}(x)$ is obtained by evaluating p_x on some α_i, either multiplying by α_i or not, and then taking a specific coordinate in the resulting element's binary encoding. For the evaluation step, we can use fast univariate multi-point evaluation which can be done by a circuit of size $\bar{k} \cdot \mathrm{poly}(\log q) = \widetilde{O}(n)$ (see, e.g., [vzGG13, Section 10]). All other operations can be done, per output bit, by a circuit of size $\mathrm{poly}(\log q)$, so overall computing $\mathcal{C}_{\mathsf{Jus}}(x)$ can be done by a circuit of size $\widetilde{O}(n)$. ■

7.1.2 Efficient List-Decodable Codes

Good binary list-decodable codes are implied by explicit construction of small ε-*balanced codes* close to the Gilbert-Varshamov (GV) bound. We will not define these object explicitly, and just say that they give rise to $(\frac{1}{2}-\varepsilon, \varepsilon^{-2})$ list-decodable codes of length $\frac{n}{\varepsilon^c}$ for some constant $c \geq 2$. The constant c, as well as achieving small rate with non-optimal dependence on n, has been subject to an important research. Naor and Naor [NN93] obtain $c = 3$, and Ta-Shma [Ta-17] recently achieved a near-optimal $c = 2 + \alpha$ for every constant $\alpha > 0$. For our construction we use the following code having $c = 4 + \alpha$ for every constant $\alpha > 0$ (for simplicity, we state it for $c = 5$). The construction, based on distance amplification via expander walks, is given in [Ta-17] and was inspired by an unpublished result by Rozenman and Wigderson.

Lemma 7.6 (following [Ta-17])**.** *For every positive integer n and $\varepsilon = \varepsilon(n)$ there exists a binary error correcting code $\mathcal{C}\colon \{0,1\}^n \to \{0,1\}^{\bar{n}}$ which is $(\frac{1}{2}-\varepsilon, \varepsilon^{-2})$ list-decodable, for $\bar{n} = O(\frac{n}{\varepsilon^5})$, computable by a circuit C_n of size $\widetilde{O}(\bar{n})$.*

Proof: Let $\mathcal{C}_{\mathsf{Jus}}\colon \{0,1\}^n \to \{0,1\}^{cn}$ be Justesen's code, given to us in Lemma 7.5, for some constant $c > 1$. Let $G = (V = [cn], E)$ be a Ramanujan λ-expander of degree D, where we take D to be a large enough constant so that λ is half the bias of $\mathcal{C}_{\mathsf{Jus}}$.[18] We think of every vertex $v \in V$ as being labeled with some row of the $cn \times n$ generating matrix A_{Jus} of $\mathcal{C}_{\mathsf{Jus}}$. We refer to that labelling by $l(v) \in \{0,1\}^n$.

We now describe how one obtains each row of the $\bar{n} \times n$ generating matrix A of $\mathcal{C}$. For $t = 5\log_D \frac{1}{\varepsilon}$, each such row is the sum, modulo 2, of the codewords that appear along a length-t walk over G. That is, each row out of the $\bar{n} = n \cdot D^t = \frac{n}{\varepsilon^5}$ possible rows is indexed by some path $p_i = v_0^i \sim \ldots \sim v_t^i$ in G and its value is

$$\sum_{j=0}^{t} l(v_j^i) \in \{0,1\}^n .$$

Thus, given $x \in \{0,1\}^n$, computing $\mathcal{C}(x)$ is done by outputting

$$Ax = \begin{pmatrix} P_1^\dagger \\ \vdots \\ P_{\bar{n}}^\dagger \end{pmatrix} \cdot A_{\mathsf{Jus}} \cdot x,$$

where P_i is the vector of length cn in which $P_i(v) = 1$ wherever $v \in p_i$ and 0 elsewhere.

For simplicity, consider hardwiring the encoding of each of the $\bar{n}$ possible paths. That is, for every $i \in [\bar{n}]$ we keep a string of length $(t+1)\log(cn)$ storing the vertices along p_i. This hardwiring takes $\widetilde{O}(\bar{n})$ bits overall. Now, given $x \in \{0,1\}^n$, computing $A_{\mathsf{Jus}} \cdot x = \mathcal{C}_{\mathsf{Jus}}(x)$ is done once, which by Lemma 7.5 can be computed with a circuit of size $\widetilde{O}(n)$. To get $\mathcal{C}(x)$ from $\mathcal{C}_{\mathsf{Jus}}(x)$ we need to sum, for each $i \in [\bar{n}]$, the $t+1$ coordinates in $\mathcal{C}_{\mathsf{Jus}}(x)$ that corresponds to the path p_i. As we have those coordinates hardwired, it takes only $\widetilde{O}(t\bar{n}) = \widetilde{O}(\bar{n})$ size, and overall the lemma follows. ■

We note that if we insist on a *uniform* computation of $\mathcal{C}$ we can still work with good enough explicit expanders and *compute* the vectors p_i instead of fixing them (see Theorem 2.23). This would take $\widetilde{O}(\frac{n}{\varepsilon^c})$ time and possibly deteriorate the constant c by a bit.

[18] By Ramanujan here we mean that λ is optimally related to the degree D, i.e., $\lambda \approx \frac{2}{\sqrt{D}}$. We do not insist on having $\lambda \leq \frac{2\sqrt{D-1}}{D}$ and do allow some slackness, and such graphs exist for every n [F+03].

7.1.3 Almost k-wise Independent Sample Spaces

Definition 7.7 (almost k-wise distribution)**.** *A distribution $(X_1, \ldots, X_n)$ over $\{0,1\}^n$ is (k, ε, p) independent if for every subset $\{i_1, \ldots, i_k\} \subseteq [n]$, $(X_{i_1}, \ldots, X_{i_k})$ is ε-close to the distribution over k-bit strings where all bits are independent and each of them takes the value 1 with probability p.*

Constructing almost k-wise distribution with optimal support size, at least for the $p = \frac{1}{2}$ case, is well-known. In what follows, we argue that sampling from the support of such a distribution can be done by a small circuit. For simplicity, we restrict ourselves to the case of $k = 2$.

Lemma 7.8. *For every positive integer n, $\varepsilon = \varepsilon(n)$ and $q = q(n)$ the following holds. Set $\ell = 2^q \frac{4q^2 \log^2(qn)}{\varepsilon^2}$. Then, there exists a circuit $C_n \colon [\ell] \to \{0,1\}^n$ of size $\widetilde{O}(qn + 2^q \frac{\log n}{\varepsilon^2})$ such that for every $y \in [\ell]$, $C_n(y)$ computes the y-th element of a $(2, \varepsilon, 2^{-q})$ independent sample space in some fixed order.*

Proof: Constructing almost k-wise sample spaces for $p = \frac{1}{2}$ is done by combining truly k-wise independence and small-bias sample spaces. Following, e.g., [RSW06], we unfold the extension to a general $p = 2^{-q}$ and argue that it can be done efficiently. Set $h = 2q \log(qn)$, and so $\ell = (\frac{h}{\varepsilon'})^2$ for $\varepsilon' = 2^{-q/2}\varepsilon$. We use the following two ingredients:

- Let $B \subseteq \{0,1\}^h$ be an ε'-biased sample space.[19] Alon et al. [AGHP92] gives us a simple construction with support size ℓ.
- Let A be the $nq \times h$ generator matrix of a $(2q, 0, \frac{1}{2})$ independent sample space. The matrix A is some suitable Vandermonde matrix, converted to $\mathbb{F}_2$ in a canonical way.

Then, given $y \in [\ell]$, we output the y-th element of the almost k-wise sample space as follows.

1. Let $x \in \{0,1\}^h$ be the y-th element of B. As B is small (and of course, independent of y) we can hardwire B into our circuit, and it takes only $O(\ell h)$ bits.
2. Compute $w = Ax \in \{0,1\}^{nq}$. The matrix-vector multiplication can be done efficiently, exploiting the special structure of A. Specifically, one can use discrete Fourier transform to compute w by a circuit of size $\widetilde{O}(nq)$.
3. Partition w into n consecutive blocks $W_1, \ldots, W_n$, each of length q. Output $z \in \{0,1\}^n$ such that $z_i = 1$ if and only if all the bits in W_i are 1. This step can be implemented by a circuit of size $O(nq)$.

Overall, the three steps above can be implemented by a circuit of size $\widetilde{O}(nq) + O(\ell h) = \widetilde{O}(qn + 2^q \frac{\log n}{\varepsilon^2})$, as required. ∎

Again, a note about uniform computation is in order. Here too we have an algorithm running in time $\widetilde{O}(n + 2^q) \cdot \mathrm{poly}(\frac{1}{\varepsilon})$, by explicitly computing the y-th element of B. We skip the details.

[19] Namely, for any nonempty $S \subseteq [h]$, we require that $\left|\Pr_{b \sim B}[\bigoplus_{i \in S} b_i = 1] - \frac{1}{2}\right| \leq \varepsilon$.

7.1.4 Efficiently Computing Ext

Finally, we are ready to prove that Ext can be computed by a small circuit.

Lemma 7.9. *There exists a constant $c \geq 1$ such that the following holds. For some constant $0 < \alpha < \frac{1}{2}$, positive integer n and $\varepsilon > 0$, let $\mathsf{Ext}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ be the $(k = n^{1-\alpha}, \varepsilon)$ strong extractor given in Theorem 7.4. For any fixed $x \in \{0,1\}^n$, the function*

$$\mathsf{Ext}(x, \cdot)\colon \{0,1\}^d \to \{0,1\}^m$$

can be computed by a circuit C_x of size $\widetilde{O}(\frac{n}{\varepsilon^c})$, and it takes a circuit of size $\widetilde{O}(\frac{n}{\varepsilon^c})$ to compute the encoding of C_x.

Proof: We keep the same notation as in our construction.

- The extractor $\mathsf{TZS}\colon \{0,1\}^n \times \{0,1\}^{d_2} \to \{0,1\}^{d_3}$ from Theorem 7.1 is computed as follows. For $x \in \{0,1\}^n$ and $y_2 \in \{0,1\}^{d_2}$, we view x as a bivariate polynomial $f_x\colon \mathbb{F}^2 \to \mathbb{F}$ of total degree $h \approx \sqrt{n}$ where $|\mathbb{F}| = h \cdot \mathrm{poly}(d_3/\varepsilon)$. Then, for each $i \in [d_3]$,

  $$\mathsf{TZS}(x, y_2)_i = \mathcal{C}(f_x(a_1 + i, a_2))_j,$$

  where y_2 is interpreted as $(a_1, a_2, j) \in \mathbb{F}^2 \times [\ell]$ and $\mathcal{C}\colon \mathbb{F} \to [\ell]$ is $(\frac{1}{2} - \rho, \rho^{-1})$ list decodable for $\rho = \Omega(\varepsilon^2/d_3^2)$. Given $x \in \{0,1\}^n$, evaluating f_x on d_3 inputs can be naively done by a circuit of size $d_3 \cdot h^2 \cdot \mathrm{polylog}(|\mathbb{F}|) = \widetilde{O}(n \log \frac{1}{\varepsilon})$. Applying $\mathcal{C}$, by Lemma 7.6 takes size $\widetilde{O}(\frac{n}{\rho^5}) = \widetilde{O}(\frac{n}{\varepsilon^{10}})$. Thus, $\widetilde{O}(\frac{n}{\varepsilon^{10}})$ is also the size it takes to *prepare* the encoding of the circuit that computes $\mathsf{TZS}(x, \cdot)$.

- The condenser $\mathsf{RSW}\colon \{0,1\}^n \times \{0,1\}^{d_1} \to \{0,1\}^{m_1}$ from Theorem 7.3 can be computed as follows. Let $\mathcal{C}_{\mathsf{Jus}}\colon \{0,1\}^n \to \{0,1\}^{\bar{n}=O(n)}$ be Justesen's code from Lemma 7.5. Let

  $$\left\{ S_y \in \{0,1\}^{\bar{n}} : y \in \{0,1\}^{d_1} \right\}$$

  be a $(2, \varepsilon, p)$ independent sample space for $p = \Theta(\frac{m_1}{\bar{n}})$, where we identify each S_y as a subset of $[\bar{n}]$. We also require that $|S_y| = pn$ for every y. Although it is not guaranteed by the construction of Lemma 7.8, [RSW06] show that enforcing this constraint by adding or removing arbitrary indices from each set is good enough. For $x \in \{0,1\}^n$ and $y_1 \in \{0,1\}^{d_1}$, the construction is given by

  $$\mathsf{RSW}(x, y_1) = \mathcal{C}(x)_{S_{y_1}}.$$

  Computing the encoding of a circuit that computes $\mathsf{RSW}(x, \cdot)$ starts by computing $\mathcal{C}(x)$, which can be done in size $\widetilde{O}(n)$ using Lemma 7.5. On input y_1, computing $\mathcal{C}(x)_{S_{y_1}}$ can be implemented in size $\widetilde{O}(\bar{n} \log \frac{1}{p} + \frac{\log \bar{n}}{p\varepsilon^2}) = \widetilde{O}(\frac{n}{\varepsilon^2})$, by Lemma 7.8. Thus, it takes $\widetilde{O}(\frac{n}{\varepsilon^2})$ size to compute the encoding of the circuit $\mathsf{RSW}(x, \cdot)$, which is also an upper bound on its size.

- The extractor $\mathsf{Tre}\colon \{0,1\}^{m_1} \times \{0,1\}^{d_3} \to \{0,1\}^m$ goes as follows. Let $\mathcal{C}\colon \{0,1\}^{m_1} \to \{0,1\}^{\overline{m_1}}$ be a $(\frac{1}{2} - \rho, \rho^{-2})$ list decodable code, for $\rho = \Omega(\varepsilon/m)$. Let

  $$\{S_i \subseteq [d_3] : i \in [m]\}$$

be a weak design, wherein each $|S_i| = \log(\overline{m_1})$ and for all $i \neq j$, $\sum_{j<i} 2^{|S_i \cap S_j|} \leq 2m$. Then, given $x \in \{0,1\}^{m_1}$ and $y \in \{0,1\}^{d_3}$, for each $i \in [m]$ we have that

$$\mathsf{Tre}(x,y)_i = \hat{x}(y|_{S_i}),$$

where we denoted $\hat{x} = \mathcal{C}(x)$. Given $x \in \{0,1\}^{m_1}$ and $y \in \{0,1\}^{d_3}$, a circuit outputting $\mathsf{Tre}(x,y)$ can be constructed as follows. The sets $S_1, \ldots, S_m$ can be hard-coded to the circuit, which takes $m \cdot d_3 = O(n)$ bits. As $\overline{m_1}$ is large, we should not compute $\hat{x}$ in full, but rather compute $y|_{S_1}, \ldots, y|_{S_m}$ at first, and then proceed to computing $\hat{x}(y|_{S_1}), \ldots, \hat{x}(y|_{S_m})$.

Computing $y|_{S_1}, \ldots, y|_{S_m}$ is immediate once we have $S_1, \ldots, S_m$ and can be done in size $O(md_3) = O(n)$. For computing $\hat{x}(y|_{S_1}), \ldots, \hat{x}(y|_{S_m})$, we revisit the proof of Lemma 7.6. We see that we can first compute $\mathcal{C}_{\mathsf{Jus}}(x)$, by a circuit of size $\widetilde{O}(m_1) = O(n)$, and then for every $z = y|_{S_i} \in \{0,1\}^{\log(\overline{m_1})}$, we interpret z as a walk p_z of length $t = O(\log \frac{m}{\varepsilon})$ over an expander with fully-explicit neighbourhood function (see Theorem 2.23). This takes size $\mathrm{polylog}(n, \frac{1}{\varepsilon})$. From p_z we can compute $\hat{x}(z)$ as described in the proof of Lemma 7.6. We conclude that a circuit of size $O(n)$ suffices to compute Tre.

Overall, accumulating the sizes, the lemma follows. ∎

Our extractor is also time-efficient.

Lemma 7.10. *For some constant $0 < \alpha < \frac{1}{2}$, positive integer n and a constant $\varepsilon > 0$, let $\mathsf{Ext}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ be the $(k = n^{1-\alpha}, \varepsilon)$ strong extractor given in Theorem 7.4. Then, given $x \in \{0,1\}^n$ and $y \in \{0,1\}^d$, $\mathsf{Ext}(x,y)$ is computable in time $\widetilde{O}_\varepsilon(n)$.*

We skip the details, and just note that to derive Lemma 7.10 from the above discussion, it is left to verify that computing the weak design can be done efficiently. Indeed, inspecting the construction of [RRV02], this can be done in time $m \cdot \mathrm{polylog}(n) = O(n)$.

8 PRGs With Nearly Optimal Slowdown

8.1 A Pseudoentropy Generator for Stronger Distinguishers

From Section 6, we see that in order to obtain pseudorandomness via extraction, we need pseudoentropy against a stronger model of circuits. Namely we need metric* pseudoentropy. We now show that under a stronger hardness assumption on f (namely hardness against SVN circuits with $\mathrm{DensityApprox}$ gates), we can construct a PEG for metric* pseudoentropy. Before, we showed that we can generate a random variable with high metric pseudoentropy by first generating a random variable with high NYao pseudoentropy and applying Lemma 2.31. We follow the same framework of going through Yao pseudoentropy via list recoverable codes, and we slightly change the hardness assumptions and definitions to handle SVN circuits with $\mathrm{DensityApprox}$ gates.

Definition 8.1. *($\mathrm{NYao}^\star_\eta$ sets) For positive integers s, n and ℓ, a constant $0 < \eta < 1$, and $A \subseteq \{0,1\}^n$, we say that $A \in \mathcal{C}^{\mathrm{NYao}^\star_\eta}_{\ell,s}$, if there exists a circuit $c\colon \{0,1\}^n \to \{0,1\}^\ell$ and a $\mathrm{DensityApprox}_\eta$ SVN circuit $d\colon \{0,1\}^\ell \times \{0,1\}^{w_d} \to \{0,1\}^n$ computing a partial function $f_d\colon \{0,1\}^\ell \to \{0,1\}^n \cup \{\perp\}$, both c and d are of size at most s, and*

$$A = \{x : f_d(c(x)) = x\}.$$

We refer to c as the compressing circuit and to d as the decompressing one.

Definition 8.2 ($\text{NYao}^\star_\eta$ pseudoentropy). *Let X be a random variable distributed over $\{0,1\}^n$, a positive integer s, and $\varepsilon, \eta > 0$. We say that $H^{\text{NYao}^\star_\eta}_{s,\varepsilon}(X) \geq k$ if for every $\ell < k$ and $A \in \mathcal{C}^{\text{NYao}^\star_\eta}_{\ell,s}$,*

$$\Pr[X \in A] \leq 2^{\ell - k} + \varepsilon.$$

For our purposes, taking $\eta = \varepsilon$ will suffice, in which case we denote $H^{\text{NYao}^\star_\eta}_{s,\varepsilon}(X) = H^{\text{NYao}^\star}_{s,\varepsilon}(X)$.

It is easy to verify that, just as before, high $\text{NYao}^\star_\eta$ pseudoentropy still implies high metric* pseudoentropy (with a nearly identical proof).

Lemma 8.3. *There exists a constant $0 < \gamma < 1$ such that the following holds. Let X be a random variable distributed over $\{0,1\}^n$ and $\varepsilon > 0$. There exists $s_0 = \widetilde{O}(n)$ such that for every $s \geq s_0$, $H^{\text{NYao}^\star}_{s,\varepsilon}(X) \geq k$ implies $H^{\text{metric}^\star}_{\gamma s,\varepsilon}(X) \geq \frac{k}{2}$.*

Proof: We only point out the slight changes to the proof of Lemma 2.31.

Assuming towards a contradiction that $H^{\text{metric}^\star}_{\gamma s,\varepsilon}(X) < \frac{k}{2}$ we invoke Lemma 2.27, to conclude that there exists an SVN circuit using oracle gates to $\text{DensityApprox}_\varepsilon$, with small support, that distinguishes X from uniform. Using the same hash function construction as in Lemma 2.31, this implies that there exist compressor and decompressor circuits c, d, where c simply computes the hash function, and d uses nondeterminism to guess the inverse of the hash function and then uses D to check if the inverse is in $\text{Supp}(D)$. Noting that d is a $\text{DensityApprox}_\varepsilon$ SVN circuit gives $H^{\text{NYao}^\star}_{s,\varepsilon}(X) < k$. ∎

We can now construct a PEG for metric* pseudoentropy by following the same steps as in Section 3. For some fixed $\varepsilon > 0$, let $f\colon \{0,1\}^{\log n} \to \{0,1\}$ be such that every SVN circuit with oracle gates to $\text{DensityApprox}_\varepsilon$ computing f has size at least $n^{1-\alpha_0}$ for some constant $\alpha_0 < \frac{1}{6}$. That is, $\text{size}_{SVN^\star_\varepsilon}(f) \geq n^{1-\alpha_0}$. Let

$$\mathcal{C}\colon \{0,1\}^n \to \Sigma^m$$

be some error correcting code. Define $G^f\colon [m] \to \Sigma \equiv \{0,1\}^{\log|\Sigma|}$ so that

$$G^f(z) = \mathcal{C}(f)_z,$$

where we identify f by its truth-table in $\{0,1\}^n$.

Theorem 8.4. *Keeping the above notation, let ε, α be constants such that $\varepsilon > 0$ and $\alpha_0 < \alpha \leq \frac{1}{6}$. Assume $\mathcal{C}$ is $(Q, \varepsilon, \ell, s_\mathcal{C})$ locally list recoverable so that $s_\mathcal{C} = O(n^{1-\alpha})$, and $\frac{n^{1-\alpha}}{Q} \geq s_0$ for some $s_0 = \widetilde{O}(\log|\Sigma|)$, where the precise $\widetilde{O}(\cdot)$ dependence is determined by Lemma 2.31. Then, G^f is a strong (k, s, ε) metric* PEG for $k = \frac{\log \ell}{2}$ and $s = O\left(\frac{n^{1-\alpha}}{Q}\right)$.*

The proof of the above theorem is identical to that of Theorem 3.1, but at the final step, we invoke our newly proven Lemma 8.3 instead of Lemma 2.31. Again, combining Theorem 4.2 with the above result gives the following.

Theorem 8.5. *For a constant $\varepsilon > 0$ and every positive integer n the following holds. Assume*

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

is such that $\mathsf{size}_{SVN^\star_\varepsilon}(f) > n^{1-\alpha_0}$ *for some constant* $\alpha_0 < \frac{1}{6}$*. Let* α *be any constant such that* $\alpha_0 < \alpha \leq \frac{1}{6}$*. Then, there exists a function*

$$G^f \colon \{0,1\}^d \to \{0,1\}^m$$

that is a (k, s, ε) *metric** *PEG for* $k = \frac{n^{1-7\alpha}}{2}$*,* $s \geq n^{1-4\alpha}$*,* $d = 5\alpha \log n + O(\log\log n)$ *and* $m = \widetilde{O}(n^{1-5\alpha})$*.*

Given oracle access to f*, the support of* G^f *takes* $\widetilde{O}(n)$ *time to compute. Moreover, if* $f \in \mathbf{DTIME}(n^{c_f})$ *for some* $c_f \geq 1$*, the support of* G^f *can be computed in time* $\widetilde{O}(n^{c_f+1})$*.*

8.2 Constructing the PRG

All ingredients are now in place for our PRG transforming almost all the hardness to pseudorandom bits using a nearly optimal seed length.

Theorem 8.6. *There exists a constant* $c \geq 7$ *such that the following holds for every positive integer* n *and a constant* $\varepsilon > 0$*. Assume* $f \colon \{0,1\}^{\log n} \to \{0,1\}$ *satisfies* $\mathsf{size}_{SVN^\star_{\varepsilon/10}}(f) > n^{1-\alpha_0}$ *for some universal constant* $\alpha_0 < \frac{1}{c}$*. Let* α *be any constant such that* $\alpha_0 < \alpha \leq \frac{1}{c}$*. Then, there exists a function*

$$\overline{G}^f \colon \{0,1\}^{(1+c\alpha)\log s} \to \{0,1\}^s$$

which is an ε*-PRG against circuits of size* $s = n^{1-c\alpha}$*.*

The support of $\overline{G}^f$ *can be computed in time* $s^{\frac{2+c\alpha}{1-c\alpha}}$ *given oracle access to the truth table of* f*. Moreover, if* $f \in \mathbf{DTIME}(s^{c_f})$ *for some* $c_f \geq 1$ *then the support of* $\overline{G}^f$ *can be computed in time* $s^{\frac{\zeta}{1-c\alpha}}$ *for* $\zeta = \max\{2 + c\alpha, c_f + 1\}$*.*

Proof: Set $\varepsilon' = \frac{\varepsilon}{10}$. Let

$$G^f \colon \{0,1\}^{d_1 = 6\alpha\left(1+\frac{7}{2}\alpha\right)\log s'} \to \{0,1\}^{m_1}$$

be the (k, s', ε') metric* PEG guaranteed by Theorem 8.5, where $s' = n^{1-\frac{7}{2}\alpha}$, $m_1 = n^{1-4\alpha}$ and $k = n^{1-8\alpha}$. Let

$$\mathsf{Ext} \colon \{0,1\}^{m_1} \times \{0,1\}^{d_2} \to \{0,1\}^m$$

be the $(k - \log\frac{1}{\varepsilon'}, \varepsilon')$ extractor guaranteed to us by Theorem 7.4,[20] so that for every $x \in \{0,1\}^{m_1}$, $\mathsf{Ext}(x, \cdot)$ is computable by a circuit of size $t = \widetilde{O}_\varepsilon(m_1) = o(s')$. By Theorem 7.4, $d_2 = (1 + c_2\alpha)\log m_1 + O_\varepsilon(1)$ and $m = m_1^{1-c_2\alpha}$ for some universal constant $c_2 \geq 1$. Let

$$\overline{G}^f \colon \{0,1\}^{d = d_1 + d_2} \to \{0,1\}^m$$

be such that for $(y_1, y_2) \in \{0,1\}^{d_1} \times \{0,1\}^{d_2}$,

$$\overline{G}^f(y_1, y_2) = \mathsf{Ext}(G^f(y_1), y_2).$$

Denote $X = G^f(U_{d_1})$. By the properties of the PEG, $H^{\mathrm{metric}^\star}_{s',\varepsilon'}(X) \geq k$. Thus by Lemma 6.3, there exists a $c' = \widetilde{O}(1)$ (where the $\widetilde{O}$ hides factors of $\log(s+t)$) such that $\mathsf{Ext}(X, U_{d_2})$ $10\varepsilon'$-fools circuits

[20] One can achieve this by invoking Theorem 7.4 with, for example, $k' = n^{1-(8.001)\alpha}$

of size $s = \frac{1}{c'}s' - t \geq \frac{1}{2c'}s' \geq \frac{1}{2}n^{1-(\frac{7}{2}+.001)\alpha}$. Note that $m = n^{(1-4\alpha)(1-c_2\alpha)} \leq s$, and that

$$\begin{aligned}
d &= d_1 + d_2 \\
&= 6\alpha\left(1+\frac{7}{2}\alpha\right)\log s' + (1+c_2\alpha)\log m_1 + O_\varepsilon(1) \\
&\leq 6\alpha\left(1+\frac{7}{2}\alpha\right)\log s + (1+c_2\alpha)\left(1-\frac{7}{2}\alpha\right)(1-4\alpha)(1+7\alpha)\log s + O_\varepsilon(1) \\
&\leq \left(6\alpha + \frac{6\cdot 7}{2}\alpha^2\right)\log s + (1+3\alpha+4\alpha c_2)\log s + O_\varepsilon(1) \\
&\leq 10\alpha\log s + (1+3\alpha+4\alpha c_2)\log s \\
&\leq (1+13\alpha+4\alpha c_2)\log s,
\end{aligned}$$

where we use the fact that $\alpha \leq \frac{1}{7}$. The first part of the theorem then holds by taking $c = \max\left\{4c_2+13, \frac{7}{2}+.001\right\} = 4c_2+13$.

Finally, we address the time it takes to compute the support of $\overline{G}^f$. If we are given oracle access to f, it requires $\widetilde{O}(n) = \widetilde{O}(s^{\frac{1}{1-c\alpha}})$ time to compute the support of the PEG G^f. If we assume $f \in \mathbf{DTIME}(s^{c_f})$, then it takes time $n^{c_f+1} = s^{\frac{c_f+1}{1-c\alpha}}$ to compute f at every input and so overall it takes time

$$\widetilde{O}\left(s^{\frac{c_f+1}{1-c\alpha}}\right)$$

to compute the support of the PEG. By Lemma 7.10, the extractor takes time $\widetilde{O}_\varepsilon(m_1)$ on a single seed. To compute the support of $\overline{G}^f$, we compute the support of the PEG, and for every element of the support, we run the extractor on every seed. This takes time

$$\widetilde{O}(n^{5\alpha})\cdot 2^{d_2}\cdot\widetilde{O}_\varepsilon(m_1) = \widetilde{O}_\varepsilon\left(n^{5\alpha}m_1^{2+c_2\alpha}\right) = \widetilde{O}_\varepsilon\left(n^{5\alpha+(1-4\alpha)(2+c_2\alpha)}\right) \leq s^{\frac{2+c\alpha}{1-c\alpha}}.$$

Thus overall, if we have oracle access to f, computing the support of $\overline{G}^f$ takes time $s^{\frac{2+c\alpha}{1-c\alpha}}$. If we assume $f \in \mathbf{DTIME}(s^{c_f})$, then it takes time $s^{\frac{\zeta}{1-c\alpha}}$ for $\zeta = \max\{2+c\alpha, c_f+1\}$. ∎

Indeed, PRGs allow for a black-box derandomization, and ours allow for a black-box derandomization with only an almost linear slowdown.

Theorem 8.7. *There exists a constant $\widetilde{c} \geq 1$ such that the following holds. Let $L \subseteq \{0,1\}^n$ and $A\colon \{0,1\}^n \times \{0,1\}^t \to \{0,1\}$ be a probabilistic algorithm running in time $t = t(n) \geq n$ such that for every $x \in \{0,1\}^n$,*

$$\Pr_{y\sim U_t}[A(x,y) \neq L(x)] < \frac{1}{2} - \varepsilon$$

for some constant $\varepsilon > 0$. Assume for every positive integer m there exists a function $f\colon \{0,1\}^m \to \{0,1\}$ computable in deterministic time $2^{c_f m}$ for some c_f, for which $\mathsf{size}_{SVN^\star_{\varepsilon/10}}(f) > 2^{(1-\alpha_0)m}$ for some universal constant $\alpha_0 < \frac{1}{\widetilde{c}}$. Let α be such that $\alpha_0 < \alpha \leq \frac{1}{\widetilde{c}}$.

Then, there exists a deterministic algorithm $A_{\mathsf{D}}\colon \{0,1\}^n \to \{0,1\}$ that accepts L and runs in time

$$t^{2+\widetilde{c}\alpha} + t_P,$$

where the term $t_P = t^{\gamma(1+\widetilde{c}\alpha)}$ for $\gamma = \max\{2+\widetilde{c}\alpha, c_f+1\}$ corresponds to a step that can be precomputed for all algorithms with running time t. That is, the slowdown of every randomized algorithm running in time t, under our complexity-theoretic assumptions, is at most $t^{1+O(\alpha)}$.

Proof: Let c be the constant guaranteed to us by Theorem 8.6 and set $c' = \frac{c+1}{1-c^2\alpha} > 0$. Set $m = \log(t^{1+c'\alpha})$, $M = 2^m$, and let $f\colon \{0,1\}^{\log(t^{1+c'\alpha})} \to \{0,1\}$ be the guaranteed hard function. Let

$$\overline{G}^f\colon \{0,1\}^{d=(1+c\alpha)\log s} \to \{0,1\}^s$$

be the ε-PRG fooling circuits of size s guaranteed to us by Theorem 8.6, with $s = M^{1-c\alpha}$. We note that $s \geq t^{1+\alpha} = \omega(t\log t)$. Furthermore, we can see that $d \leq (1+c'\alpha)\log t$ since $d = (1+c\alpha)(1-c\alpha)\log M < \log M$, and $(1+c'\alpha)\log t = \log M$.

Consider truncating the output length of $\overline{G}^f$ down to length t. Fix some $x \in \{0,1\}^n$ and let $C_x\colon \{0,1\}^t \to \{0,1\}$ be the circuit that computes $A(x,\cdot)$, of size s. By the properties of the PRG,

$$\left|\Pr[C_x(U_t)=1] - \Pr[C_x(\overline{G}^f(U_d))=1]\right| \leq \varepsilon,$$

so for $x \in L$, $\Pr[C_x(\overline{G}^f(U_d))=1] > \frac{1}{2}$ and for $x \notin L$, $\Pr[C_x(\overline{G}^f(U_d))=1] < \frac{1}{2}$. Hence, the standard way of constructing A_{D} would be to first compute the set

$$I = \left\{\overline{G}^f(z) : z \in \{0,1\}^d\right\},$$

which is independent of C_x so can be thought of as a preprocessing step for all circuits of a certain size, and then run $A(x,y)$ for every $y \in I$. A_{D} would then accept if and only if the majority of runs returned 1. The parameters immediately follow.

The time t_P represents the time it takes to compute the support of the PRG. As Theorem 8.6 states, computing the support of the PRG takes time $s^{\frac{\gamma'}{1-c\alpha}} = t^{\gamma'(1+c'\alpha)}$, for $\gamma' = \{2+c\alpha, c_f+1\}$. The theorem holds by setting $\widetilde{c} = \max\{c^2, c', 7\}$. ∎

9 Assuming Hardness for Randomized SVN Circuits

Having access to true randomness, we can solve $\mathrm{DensityApprox}_\eta$ efficiently by randomly sampling inputs to the circuit whose density we wish to approximate. This motivates using a hardness assumption for randomized SVN circuits (see Definition 2.4).

Lemma 9.1. *Let $f\colon \{0,1\}^n \to \{0,1\}$ be computable by an SVN circuit of size $s = s(n)$ having oracle gates to $\mathrm{DensityApprox}_\eta$ for some fixed constant $\eta > 0$. Then, for any $0 < \delta < \frac{1}{2}$, f is computable by a randomized SVN circuit with error δ, and size $O(s\log s\log\frac{1}{\delta})$.*

Proof: Let $C\colon \{0,1\}^n \times \{0,1\}^w \to \{0,1\}$ be the SVN circuit of size s that computes f and uses Q oracle queries to $\mathrm{DensityApprox}_\eta$. Set $d = c\frac{\log(Q/\delta)}{\eta^2}s$ for a constant $c > 0$ to be determined later, and let $C'\colon \{0,1\}^n \times \{0,1\}^w \times \{0,1\}^d \to \{0,1\}$ be the circuit that gets as input $x \in \{0,1\}^n$, $y \in \{0,1\}^w$ and $r = (r_1,\ldots,r_Q) \in \{0,1\}^d$ and is constructed as follows.

The circuit C' acts like C (both the check phase and the compute phase), however the i-th $\mathrm{DensityApprox}_{\varepsilon}$ gate g_i is replaced with the following circuit $\mathcal{G}_i$. Let $\mathcal{Z}_i$ be the input circuit to g_i. Notice that $\sum_{i\in[Q]} \mathsf{size}(\mathcal{Z}_i) \leq s$.

We let $r_i = (r_i^1, \ldots, r_i^T)$ be the randomness to $\mathcal{G}_i$, for $T = c\frac{\log(Q/\delta)}{\eta^2}$. Note each $|r_i^j| \leq |z_i|$. On r_i, $\mathcal{G}_i$ computes $\mathcal{Z}_i(r_i^j)$ for $j \in [T]$ and computes the average over outputs, truncated to $\lceil \log \frac{1}{\varepsilon} + 1 \rceil$ bits. Note that the size of $\mathcal{G}_i$ is at most $T \cdot \mathsf{size}(\mathcal{Z}_i) + \mathsf{size}(\mathcal{Z}_i)$ and so the size of C' is at most $T \cdot s + s$. Using the fact that $Q \leq s$, we get that the total size of C' is at most $O(s \log s \log \frac{1}{\delta})$

For correctness, fix some input x. As C is an SVN circuit, there exists some $y \in \{0,1\}^m$ for which $C(x,y)_{\mathsf{check}} = 1$, and whenever $C(x,y)_{\mathsf{check}} = 1$ it holds that $C(x,y) = f(x)$. Fix any y such that $C(x,y)_{\mathsf{check}} = 1$. By the Chernoff bound, there exists a constant $c > 0$ for which

$$\Pr_{r_i \sim U}\left[\left|\frac{1}{T}\sum_{j=1}^{T} Z_i(r_i^j) - \mathbb{E}(Z_i)\right| > \frac{\eta}{2}\right] \leq \frac{\delta}{Q}$$

for every $i \in [Q]$. Thus, by a union bound, we are guaranteed that with probability at least $1-\delta$ over $r \sim U_d$, all $\mathcal{G}_i$-s return an η-approximation of $\mathbb{E}(\mathcal{Z}_i)$.

Let $\mathbf{G}_C$ be the set of all y-s for which $C(x,y)_{\mathsf{check}} = 1$. Let $\mathbf{G}_{C'}$ be the set of all y-s for which $\Pr_{r\sim U_d}[C'(x,y,r)_{\mathsf{check}} = 1] \geq 1-\delta$. First, suppose $y \in \mathbf{G}_C$. Then we know that if all $\mathcal{G}_i$-s return an η-approximation of $\mathbb{E}(\mathcal{Z}_i)$, then C' must behave exactly like C. Therefore, $\Pr_{r\sim U_d}[C'(x,y,r)_{\mathsf{check}} = 1] \geq 1-\delta$ and so $y \in \mathbf{G}_{C'}$. Next, suppose $y \notin \mathbf{G}_C$. Then $C(x,y)_{\mathsf{check}} = 0$. Again, if all $\mathcal{G}_i$-s return an η-approximation of $\mathbb{E}(\mathcal{Z}_i)$, then C' must also output 0 as the check flag. Thus, $\Pr_{r\sim U_d}[C'(x,y,r)_{\mathsf{check}} = 0] \geq 1-\delta$. Since $\delta < \frac{1}{2}$, this means that on witness y, the circuit C' cannot also output 1 as the check flag with probability at least $1-\delta$, so $y \notin \mathbf{G}_{C'}$.

The argument above shows that $\mathbf{G}_C = \mathbf{G}_{C'}$. Moreover, it shows that for every x, y, either $\Pr_{r\sim U_d}[C'(x,y,r)_{\mathsf{check}} = 1] \geq 1-\delta$ or $\Pr_{r\sim U_d}[C'(x,y,r)_{\mathsf{check}} = 1] \leq \delta$. Thus C' is a proper randomized SVN circuit computing $f(x)$. ■

Observing that our definition of randomized and DensityApprox SVN only has the checking circuit use the randomness or oracle gates (with the compute phase only using the witness), we can show that the converse also holds.

Lemma 9.2. *Let $\eta > 0$ be any constant, and let $\frac{1}{2} - \eta > \delta > 0$. Let $f\colon \{0,1\}^n \to \{0,1\}$ be computable by a randomized SVN circuit C of size $s = s(n)$ with error δ. Then, f is also computable by a* $\mathrm{DensityApprox}$ *SVN circuit of size $\widetilde{O}(s)$ using one* $\mathrm{DensityApprox}_{\eta}$ *gate.*

Proof: Consider the following $\mathrm{DensityApprox}$ SVN circuit C'. The compute circuit of C' is identical to that of C. The check circuit of C', on input x, y, computes a description of $C(x,y,\cdot)_{\mathsf{check}}$, and feeds that description into a $\mathrm{DensityApprox}_{\eta}$. The check outputs 1 if the output of the $\mathrm{DensityApprox}_{\eta}$ gate is at least $\frac{1}{2}$, and outputs 0 otherwise.

Since for every y we have $\Pr_{r\sim U_d}[C(x,y,r)_{\mathsf{check}} = 1] \geq 1-\delta$ or $\Pr_{r\sim U_d}[C(x,y,r)_{\mathsf{check}} = 1] < \delta$, the $\mathrm{DensityApprox}_{\eta}$ gate is accurate enough to distinguish between the two. Since the description of $C(x,y,\cdot)_{\mathsf{check}}$ is at most $\widetilde{O}(s)$, the overall size of C' is $\widetilde{O}(s)$. ■

Utilizing Lemma 9.1 we can rephrase Theorem 8.6 and Theorem 8.7 with our new hardness assumption. Note that by inspecting Lemma 9.1 we can take the error of the randomized SVN

circuit to be quasi-polynomially small in the input length (e.g. $\delta = 2^{-\log^{c'} n}$ for any constant c').[21]

Theorem 9.3. *There exists a constant $c \geq 7$ such that the following holds for every positive integer n and a constant $\varepsilon > 0$. Assume $f\colon \{0,1\}^{\log n} \to \{0,1\}$ requires randomized SVN circuits for error δ, of size $n^{1-\alpha_0}$ for some universal constant $\alpha_0 < \frac{1}{c}$, and $\delta = 2^{-\log^{c'} n}$ for an arbitrary constant c'. Let α be any constant such that $\alpha_0 < \alpha \leq \frac{1}{c}$. Then, there exists a function*

$$\overline{G}^f\colon \{0,1\}^{(1+c\alpha)\log s} \to \{0,1\}^s$$

which is an ε-PRG against circuits of size $s = n^{1-c\alpha}$.

The support of $\overline{G}^f$ can be computed in time $s^{\frac{2+c\alpha}{1-c\alpha}}$ given oracle access to the truth table of f. Moreover, if $f \in \mathbf{DTIME}(s^{c_f})$ for some $c_f \geq 1$ then the support of $\overline{G}^f$ can be computed in time $s^{\frac{\gamma}{1-c\alpha}}$ for $\gamma = \max\{2 + c\alpha, c_f + 1\}$.

Theorem 9.4. *There exists a constant $\widetilde{c} \geq 1$ such that the following holds. For any n, let $L \subseteq \{0,1\}^n$ and $A\colon \{0,1\}^n \times \{0,1\}^t \to \{0,1\}$ be a probabilistic algorithm running in time $t = t(n) \geq n$ such that for every $x \in \{0,1\}^n$,*

$$\Pr_{y \sim U_t}[A(x,y) \neq L(x)] < \frac{1}{2} - \varepsilon$$

for some constant $\varepsilon > 0$. Assume for every positive integer m there exists a function $f\colon \{0,1\}^m \to \{0,1\}$ such that:

- *f is computable in deterministic time $2^{c_f m}$ for some c_f, and,*
- *For $\delta = 2^{-\log^{c'} n}$ for an arbitrary constant $c' > 0$, f requires randomized SVN circuits for error δ of size at least $2^{(1-\alpha_0)m}$, where $\alpha_0 < \frac{1}{\widetilde{c}}$ is some universal constant.*

Then, there exists a deterministic algorithm $A_{\mathsf{D}}\colon \{0,1\}^n \to \{0,1\}$ that accepts L and runs in time

$$t^{2+\widetilde{c}\alpha} + t_P,$$

where the term $t_P = t^{\gamma(1+\widetilde{c}\alpha)}$ for $\gamma = \max\{2 + \widetilde{c}\alpha, c_f + 1\}$ corresponds to a step that can be precomputed for all algorithms with running time t. That is, the slowdown of every randomized algorithm running in time t, under our complexity-theoretic assumptions, is at most $t^{1+O(\alpha)}$.

10 Lower Error PEGs and PRGs

In this section, we show how to get PEGs and PRGs with error $n^{-\gamma}$ for a small constant γ. Overall, our PEG and PRG have seed lengths $O(\gamma)\log n$ and $(1 + O(\gamma))\log n$, respectively. Such an error is optimal, up to the constant multiplicative factors, as the seed length must be at least $\log(1/\varepsilon)$ (and for PRGs, at least $\log(n/\varepsilon)$).

As the extractor presented in Section 7 already achieves $n^{-\gamma}$ error for a sufficiently small γ, the main challenge is to construct a list recoverable code that can handle $n^{-\gamma}$ *agreement*. As before, we take the length-n truth table of a hard function f, encode it via a locally list decodable code,

[21] In fact, we can even choose δ to be 2^{-n^γ} for a small $\gamma = \gamma(\alpha)$, but for simplicity we state the theorem for $\delta = 2^{-\log^{c'} n}$ where c' can be an arbitrary constant.

and consolidate the symbols of the code into $n^{O(\alpha)}$ larger symbols of size $n^{1-O(\alpha)}$ each. Previously, in Section 4, we showed that by simply concatenating consecutive symbols (or, "folding"), an ε agreement in the consolidated symbols implies an ε agreement in the original LDC symbols. The same technique does not work when $\varepsilon = n^{-\gamma}$, as current locally decodable codes have bad dependence on ε and so we incur a large blowup in parameters.

The idea is to instead consolidate symbols in such a manner that wherever the small $n^{-\gamma}$ fraction of good larger symbols might be, at least $1/3$ fraction of the original symbols are contained in at least one of the good larger symbols. Samplers with $n^{-\gamma}$ error naturally satisfy such a requirement (in fact, even their one-sided version, called hitters, suffices). For this to make sense, we need the decoder circuit to be hardwired with information about which lists are good. This is yet another place where we leverage the non-uniform nature of our decoding, recalling that the good lists are only a function of the hard function itself.

We note that using expander- (or sampler-) based transformations in coding theory is quite common, and can be found in several other works, e.g., in [ABN+92, AEL95, GI02, GI03, KMRZS17, DHK+19].

10.1 A New Locally List Recoverable Code

For the reader's convenience and continuity with the previous result, we once again start with the Reed-Muller code. However, as before, other locally list decodable codes are viable. Viewing $[\bar{n}]$, the set of coordinates of the RM code, as a set of vertices, we consolidate symbols according to the bipartite graph of a sampler $\Gamma\colon [\bar{n}] \times [D] \to [m]$.

10.1.1 The Construction

We first state the existence of an explicit sampler. We defer the proof to Appendix B.

Theorem 10.1 (following [GW97])**.** *There exists a constant $b > 1$ such that the following holds for every positive integer N and any constants $\delta, \beta \in (0,1)$ and $0 < \gamma < \frac{1-\beta}{b}$. There exists an explicit bi-regular $(\delta, \varepsilon = N^{-\gamma})$ sampler*

$$\Gamma\colon [N] \times [D] \to [M = N^{\beta}]$$

for $D = N^{b\gamma}$, and a circuit $C\colon [N] \to [M]^D$ of size $\widetilde{O}(D)$ that on input $x \in [N]$ outputs $\Gamma(x,1), \ldots, \Gamma(x,D)$.

Moreover, Γ is bi-regular and the function $\Gamma^{-1}\colon [M] \times [a] \to [N]$ for $a = \frac{ND}{M} = N^{1+b\gamma-\beta}$ is fully-explicit, i.e., given $v \in [M]$ and $i \in [a]$, $\Gamma^{-1}(v,i)$ can be computed in $\mathrm{polylog}(N)$ time.

Now, let α be a small constant, which we again think of as the exponent of our hardness assumption. Let $\beta = 6\alpha$, and let $\gamma = \frac{\alpha}{b}$ for the constant b given in Theorem 10.1. Assume $\alpha < \frac{1}{7}$, so that the condition $\alpha < 1 - 6\alpha = 1 - \beta$ is met. The locally list recoverable code is constructed as follows.

- In Theorem 4.1, set $\zeta = \frac{1}{3}$, $\varepsilon = \frac{1}{3}$, $t = \frac{1}{\alpha}$ and $q = \frac{8}{\varepsilon^2} \cdot d$ with d chosen so that $n = \binom{t+d}{d} \log q$. Again, working out the parameters, one can see that $d = \widetilde{O}_\alpha(n^\alpha)$ and $q = \widetilde{O}_\alpha(n^\alpha)$ (here, the α subscripts hide a $\frac{1}{\alpha}$ factor). The resulting code

$$\mathcal{C}_{\mathsf{RM}}\colon \{0,1\}^n \to \mathbb{F}_q^{\bar{n}=cn}$$

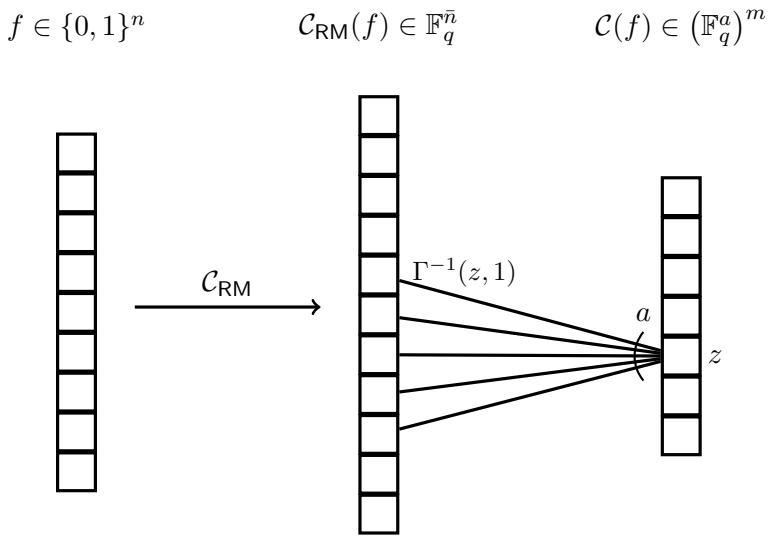

Figure 1: The construction of our locally recoverable code. The middle layer is the Reed-Muller encoding of f with symbols in $\mathbb{F}_q$. A symbol at coordinate z in our final code is an element of $\mathbb{F}_q^a$, where the a elements of $\mathbb{F}_q$ are determined by $\Gamma^{-1}(z)$.

is a

$$\left(Q = \widetilde{O}_\alpha(n^{2\alpha}), \varepsilon = \frac{1}{3}, \zeta = \frac{1}{3}, s_{\mathsf{RM}} = \widetilde{O}(n^{4\alpha}), L = O(n)\right)$$

locally list decodable code for $c = \left(\frac{72}{\alpha}\right)^{\frac{1}{\alpha}} \log n$. Note that this time, while ε is set to be constant, we aim at a non-constant error for our PEG.

- Let $m = \bar{n}^\beta = \widetilde{O}_\alpha(n^{6\alpha})$ and let
$$\Gamma \colon [\bar{n}] \times [D] \to [m]$$
be a biregular $(\frac{2}{3}, \bar{n}^{-\gamma} = \widetilde{O}_\alpha(n^{-\gamma}))$ sampler guaranteed to us by Theorem 10.1. The right-degree of the sampler is $a = \bar{n}^{1+b\gamma-\beta} = \widetilde{O}_\alpha(n^{1-5\alpha})$. The left-degree is $D = \bar{n}^{b\gamma} = \widetilde{O}_\alpha(n^\alpha)$.

- Define
$$\mathcal{C} \colon \{0,1\}^n \to \Sigma^m$$
where $\Sigma = \mathbb{F}_q^a$, so that for every $f \in \{0,1\}^n$ and $z \in [m]$,
$$\mathcal{C}(f)_z = \mathcal{C}_{\mathsf{RM}}(f)_{\Gamma^{-1}(z,1)} \circ \ldots \circ \mathcal{C}_{\mathsf{RM}}(f)_{\Gamma^{-1}(z,a)}.$$
See also Figure 1.

10.1.2 The Analysis

We prove the following theorem about $\mathcal{C}$.

Theorem 10.2. *Let $b > 1$ be as in Theorem 10.1. For any positive integer n and constant $0 < \alpha \leq \frac{1}{7}$, setting $\gamma = \frac{\alpha}{b}$, the code $\mathcal{C}$ constructed above is $(Q, \varepsilon, \ell, s_{\mathcal{C}})$ locally list recoverable for $Q = \widetilde{O}(n^{2\alpha})$, $\varepsilon = \frac{2}{n^\gamma}$, $\ell = 2^{n^{1-8\alpha}}$ and $s_{\mathcal{C}} = n^{1-\alpha}$.*

Proof: We describe a similar decoding algorithm to that of Theorem 4.2, using advice. The main observation for the new decoder is that the advice can also encode *which* lists contain a correct symbol. In other words, the advice can tell the recovery circuit which specific lists are in agreement with the encoded message. This means that for any query to the RM code, it is still efficient to brute force over all neighbors of the queried coordinate (as long as the left degree of the samplers is at most $n^{1-O(\alpha)}$). Thus we can answer the query with the correct symbol for a large fraction of queries if a large fraction of coordinates have at least one neighbor that has a good list – a property that the sampler guarantees.

Towards this end, fix any codeword $\mathcal{C}(f)$ for $f \in \{0,1\}^n$ satisfying $\mathcal{C}(f)_z \in S_z$ for at least ε-fraction of $z \in [m]$. We describe a randomized circuit for computing $\mathcal{C}(f)$. Consider

$$A_{adv}^{\mathcal{O}_{\bar{S}}}(x, r)$$

which takes as input $x \in [n]$, uses randomness string r, has access to some fixed oracle $\mathcal{O}_{\bar{S}}$ for lists $\bar{S} = S_1, \ldots, S_m$, and aims to compute f_x. The decoder circuit $A_{adv}^{\mathcal{O}_{\bar{S}}}$ implements the following procedure.

1. We first describe the advice.
   - $A_{adv}^{\mathcal{O}_{\bar{S}}}$ is hardwired with advice $adv = (y_1, \ldots, y_m, g_1, \ldots, g_m, \mathsf{RM})$, where each $y_z \in \{0,1\}^{\log \ell}$ points to some list entry in the z-th list. We assume the correct advice is given. Specifically, for each z such that $\mathcal{C}(f)_z \in S_z$, let the correct y_z be such that $\mathcal{O}_{\bar{S}}(y_z) = \mathcal{C}(f)_z$, otherwise let y_z be an arbitrary string such that $\mathcal{O}_{\bar{S}}(y_z) \neq \perp$. Note that the size of the advice is $m \log \ell = \widetilde{O}(n^{1-2\alpha})$, and only depends on f and $\bar{S}$ and not on the input x.
   - Each $g_1, \ldots, g_m$ is a single bit. In the correct advice, $g_z = 1$ if and only if $\mathcal{C}(f)_z \in S_z$.
   - The rest of the advice RM points to a decoding circuit for the Reed-Muller code, and we assume the correct circuit is given. That is, let A_{RM} be the randomized circuit that computes f_x for every x, with probability at least $\frac{2}{3}$ when given oracle access to a word in $\mathbb{F}_q^{\bar{n}}$ with at most $\frac{2}{3}$ fraction of errors from $\mathcal{C}_{\mathsf{RM}}(f)$.
2. Run the decoding circuit $A_{\mathsf{RM}}(x, r)$. For every $j \in [Q]$, supply the answer to the j-th oracle query that A_{RM} makes to some coordinate $i \in [\bar{n}]$ as follows:
   (a) Compute $\Gamma(i, 1), \ldots, \Gamma(i, D)$.
   (b) Check if any of the $g_{\Gamma(i,1)}, \ldots, g_{\Gamma(i,D)}$ are 1. If none of them are, return an arbitrary symbol in $\mathbb{F}_q$.
   (c) Otherwise, pick an arbitrary $d \in [D]$ such that $g_{\Gamma(i,d)} = 1$. Let $z = \Gamma(i, d)$. Use the advice string y_z and the oracle access to $\bar{S}$ to query the element $\widetilde{v} = \mathcal{O}_{\bar{S}}(y_z) \in \mathbb{F}_q^a$.
   (d) Return the appropriate symbol $\widetilde{v}_h \in \mathbb{F}_q$ (for $h \in [a]$) that corresponds to the symbol for the i-th coordinate of A_{RM}. That is, return $\widetilde{v}_h$ for h such that $\Gamma^{-1}(z, h) = i$.
3. Return the output of $A_{\mathsf{RM}}(x, r)$.

First we argue that the size of such a circuit is small. The inputs to the circuit are $x \in \{0,1\}^{\log n}$ and the randomness r. Since Reed-Muller decoding is the only place our algorithm uses randomness, the length of r is subsumed by the size of A_{RM}. The advice that we hardwire to the circuit,

excluding RM, has length $\widetilde{O}(n^{1-2\alpha}) + \widetilde{O}(n^{6\alpha})$. Next, the size of A_{RM} itself is $\widetilde{O}(n^{4\alpha})$, and for each of the $Q = \widetilde{O}(n^{2\alpha})$ queries, we compute $\Gamma(i,1), \ldots, \Gamma(i,D)$, and compare the results to the advice bits $g_1, \ldots, g_m$. This will add $Q \cdot \widetilde{O}(D) = \widetilde{O}(n^{3\alpha})$ to the size of the circuit. For each query, we also compute a single inverse $\Gamma^{-1}(z,h)$ which adds another $\mathrm{polylog}(\bar{n})$ additive factor. Overall, the size of the circuit is at most

$$\log n + \widetilde{O}(n^{1-2\alpha}) + \widetilde{O}(n^{6\alpha}) + \widetilde{O}(n^{4\alpha}) + \widetilde{O}(n^{4\alpha}) = \widetilde{O}(n^{1-2\alpha}).$$

The derandomized circuit obtained via repeated amplification, as we did in Section 4.2, incurs another $\log n$ factor, so overall the deterministic circuit will have size at most $n^{1-\alpha}$.

It if left to show that the randomized circuit described above errs with probability at most $\frac{1}{3}$. As before, we show that the Reed-Muller decoder essentially queries a word with $\frac{1}{3}$ agreement with $\mathcal{C}_{\mathsf{RM}}(f)$. Let $\mathbf{G} \subseteq [m]$ be the set of density at least $\varepsilon = \frac{2}{n^\gamma}$ of all z-s such that S_z contains the correct symbol $\mathcal{C}(f)_z$.

Consider the induced Reed-Muller word w obtained by setting $w_i = \mathcal{C}_{\mathsf{RM}}(f)_i$ if $\Gamma(i,d) \in \mathbf{G}$ for some d, and w_i is set arbitrarily otherwise. By the properties of the sampler Γ, at least $\frac{1}{3}$ of the Reed-Muller code coordinates have at least one neighbor in $\mathbf{G}$, and so w has at least $\frac{1}{3}$ agreement with $\mathcal{C}_{\mathsf{RM}}(f)$. By construction, the Reed-Muller decoder essentially makes queries to w. ■

This list recoverable code immediately implies a pseudoentropy generator with small error.

Corollary 10.3. *Let $b > 1$ be as in Theorem 10.1. For any positive integer n, assume*

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

is such that $\mathsf{size}_{SVN}(f) > n^{1-\alpha_0}$ for some constant $\alpha_0 < \frac{1}{8}$. Let α be any constant such that $\alpha_0 < \alpha < \frac{1}{8}$ and let $\gamma = \frac{\alpha}{b}$. Then, there exists a function

$$G^f\colon \{0,1\}^d \to \{0,1\}^a$$

that is a (k,s,ε) SVN metric PEG for $\varepsilon = \frac{2}{n^\gamma}$, $k = \frac{n^{1-8\alpha}}{2}$, $s \geq n^{1-4\alpha}$, $d = 6\alpha \log n + O(\log\log n)$ and $a = \widetilde{O}(n^{1-5\alpha})$.

Given oracle access to f, the support of G^f takes $\widetilde{O}(n)$ time to compute. Moreover, if $f \in \mathbf{DTIME}(n^{c_f})$ for some $c_f \geq 1$, the support of G^f can be computed in time $\widetilde{O}(n^{c_f+1+\alpha})$.

The correctness proof is identical to the proof of Theorem 4.7. For the computation time, all that is changed is the way we consolidate the symbols, and Theorem 10.1 tells us we can do it efficiently.

10.2 Low Error PRG

Just as before, if we now assume hardness against randomized SVN circuits, we can combine our new low error PEG with the previous extractor to obtain a low error PRG, now instantiating the extractor with a smaller error.

Theorem 10.4. *There exists a constant $c \geq 7$ such that the following holds for every positive integer n. Assume $f\colon \{0,1\}^{\log n} \to \{0,1\}$ requires randomized SVN circuits for error δ, of size $n^{1-\alpha_0}$ for some*

universal constant $\alpha_0 < \frac{1}{c}$, and $\delta = 2^{-\log^{c'} n}$ for an arbitrary constant c'. Let α be any constant such that $\alpha_0 < \alpha \leq \frac{1}{c}$. Let $\gamma = \frac{\alpha}{c}$, and let $\varepsilon = c \cdot n^{-\gamma}$. Then, there exists a function

$$\overline{G}^f : \{0,1\}^{(1+c\alpha)\log s} \to \{0,1\}^s$$

which is an ε-PRG against circuits of size $s = n^{1-c\alpha}$.

The support of $\overline{G}^f$ can be computed in time $s^{\frac{2+c\alpha}{1-c\alpha}}$ given oracle access to the truth table of f. Moreover, if $f \in \mathbf{DTIME}(s^{c_f})$ for some $c_f \geq 1$ then the support of $\overline{G}^f$ can be computed in time $s^{\frac{\zeta}{1-c\alpha}}$ for $\zeta = \max\{2 + c\alpha, c_f + 1\}$.

Proof (sketch): First fix α and let $\gamma = \frac{\alpha}{b}$ for the b as in Theorem 10.1. As before, we can first prove the result assuming hardness for $\mathrm{DensityApprox}$ gates. This is simply repeating the arguments of Theorem 8.6. In order to do so, we want to instantiate our extractor Ext from Theorem 7.4 with error $n^{-\gamma}$ and use Lemma 6.3 that guarantees pseudorandomness from high metric pseudoentropy. However, unlike previous sections, we now must work with $\mathrm{DensityApprox}_\eta$ gates with non-constant $\eta = n^{-\gamma}$. This gives us a PRG from hardness against SVN circuits with $\mathrm{DensityApprox}_\eta$ gates. For the seed length of our PRG, we add a factor of

$$(1 + c_{\mathsf{Ext}} \cdot O(\alpha)) \log n + c_{\mathsf{Ext}} \log(n^\gamma),$$

to the seed length of the PEG, where c_{Ext} is the constant guaranteed by Theorem 7.4. Choosing c large enough to subsume constants relevant to α (such as c_{Ext} and b) yields the result.

Next, to get a result for randomized SVN circuits, we once again replace our $\mathrm{DensityApprox}_\eta$ gates with random sampling. First, instantiate our just now proven result for low error PRGs assuming hardness against $\mathrm{DensityApprox}_\eta$ circuits of size $n^{1-2\alpha-2\gamma}$. That is, a distinguisher for the PRG implies a $\mathrm{DensityApprox}_\eta$ circuit of size $s = n^{1-2\alpha-2\gamma}$ computing f. Tracing the parameters of Lemma 9.1, we see that an $\mathrm{DensityApprox}_\eta$ SVN circuit of size s is computable by a randomized SVN circuit of size $\widetilde{O}(s \cdot n^{2\gamma}) = \widetilde{O}(n^{1-2\alpha}) < n^{1-\alpha}$.

■

11 Acknowledgements

We are deeply thankful to Ronen Shaltiel for pointing out an error in a previous version, for suggesting an elegant way of significantly simplifying the contents of Section 6, and for very useful discussions about hardness assumptions. We are also grateful to Noga Ron-Zewi for valuable discussions on list-recoverable codes that led to a simpler construction. We thank Roei Tell for detailed comments and suggestions on a preliminary version of this paper. Finally, we thank Scott Aaronson and Lijie Chen for suggesting open problem (3), and Omer Reingold, Amnon Ta-Shma, and Ryan Williams for helpful and interesting discussions.

References

[AASY16] Benny Applebaum, Sergei Artemenko, Ronen Shaltiel, and Guang Yang. Incompressible functions, relative-error extractors, and the power of nondeterministic reductions. *Computational Complexity*, 25(2):349–418, 2016.

[ABN+92] Noga Alon, Jehoshua Bruck, Joseph Naor, Moni Naor, and Ron M. Roth. Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs. *IEEE Transactions on Information Theory*, 38(2):509–516, 1992.

[Adl78] Leonard Adleman. Two theorems on random polynomial time. In *Proceedings of the 19th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1978)*, pages 75–83. IEEE, 1978.

[AEL95] Noga Alon, Jeff Edmonds, and Michael Luby. Linear time erasure codes with nearly optimal recovery. In *Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1995)*, pages 512–519. IEEE, 1995.

[AGHP92] Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple constructions of almost k-wise independent random variables. *Random Structures & Algorithms*, 3(3):289–304, 1992.

[AIKS16] Sergei Artemenko, Russell Impagliazzo, Valentine Kabanets, and Ronen Shaltiel. Pseudorandomness when the odds are against you. In *Proceedings of the 31st Annual Conference on Computational Complexity (CCC 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[AK97] Vikraman Arvind and Johannes Köbler. On resource-bounded measure and pseudorandomness. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 235–249. Springer, 1997.

[ALRS98] Sigal Ar, Richard J. Lipton, Ronitt Rubinfeld, and Madhu Sudan. Reconstructing algebraic functions from mixed data. *SIAM Journal on Computing*, 28(2):487–510, 1998.

[AS17] Sergei Artemenko and Ronen Shaltiel. Pseudorandom generators with optimal seed length for non-boolean poly-size circuits. *ACM Transactions on Computation Theory (TOCT)*, 9(2):6, 2017.

[BOV07] Boaz Barak, Shien Jin Ong, and Salil Vadhan. Derandomization in cryptography. *SIAM Journal on Computing*, 37(2):380–400, 2007.

[BRSW12] Boaz Barak, Anup Rao, Ronen Shaltiel, and Avi Wigderson. 2-source dispersers for $n^{o(1)}$ entropy, and Ramsey graphs beating the Frankl-Wilson construction. *Annals of Mathematics*, 176(3):1483–1544, 2012.

[BSW03] Boaz Barak, Ronen Shaltiel, and Avi Wigderson. Computational analogues of entropy. In *Approximation, Randomization, and Combinatorial Optimization – Algorithms and Techniques*, pages 200–215. Springer, 2003.

[CKLR11] Kai-Min Chung, Yael Tauman Kalai, Feng-Hao Liu, and Ran Raz. Memory delegation. In *Annual Cryptology Conference*, pages 151–168. Springer, 2011.

[CS16] Gil Cohen and Leonard J. Schulman. Extractors for near logarithmic min-entropy. In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2016)*, pages 178–187. IEEE, 2016.

[CT19] Lijie Chen and Roei Tell. Bootstrapping results for threshold circuits "just beyond" known lower bounds. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC 2019)*, pages 34–41. ACM, 2019.

[DHK+19] Irit Dinur, Prahladh Harsha, Tali Kaufman, Inbal Livni Navon, and Amnon Ta Shma. List decoding with double samplers. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2019)*, pages 2134–2153. SIAM, 2019.

[DORS08] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM Journal on Computing*, 38(1):97–139, 2008.

[DP08] Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2008)*, pages 293–302. IEEE, 2008.

[Dru13] Andrew Drucker. Nondeterministic direct product reductions and the success probability of sat solvers. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2013)*, pages 736–745. IEEE, 2013.

[F+03] Joel Friedman et al. Relative expanders or weakly relatively ramanujan graphs. *Duke Mathematical Journal*, 118(1):19–35, 2003.

[FOR15] Benjamin Fuller, Adam O'neill, and Leonid Reyzin. A unified approach to deterministic encryption: New constructions and a connection to computational entropy. *Journal of Cryptology*, 28(3):671–717, 2015.

[FR12] Benjamin Fuller and Leonid Reyzin. Computational entropy and information leakage. *IACR Cryptology ePrint Archive*, 2012:466, 2012.

[FSUV13] Bill Fefferman, Ronen Shaltiel, Christopher Umans, and Emanuele Viola. On beating the hybrid argument. *Theory of Computing*, 9(26):809–843, 2013.

[GG81] Ofer Gabber and Zvi Galil. Explicit constructions of linear-sized superconcentrators. *Journal of Computer and System Sciences*, 22(3):407–420, 1981.

[GGR98] Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *Journal of the ACM (JACM)*, 45(4):653–750, July 1998.

[GI01] Venkatesan Guruswami and Piotr Indyk. Expander-based constructions of efficiently decodable codes. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2001)*, pages 658–667. IEEE, 2001.

[GI02] Venkatesan Guruswami and Piotr Indyk. Near-optimal linear-time codes for unique decoding and new list-decodable codes over smaller alphabets. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC 2002)*, pages 812–821. ACM, 2002.

[GI03] Venkatesan Guruswami and Piotr Indyk. Linear time encodable and list decodable codes. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC 2003)*, pages 126–135. ACM, 2003.

[Gol97] Oded Goldreich. A sample of samplers: A computational perspective on sampling. *def*, 1:2n, 1997.

[GR08] Venkatesan Guruswami and Atri Rudra. Explicit codes achieving list decoding capacity: Error-correction with optimal redundancy. *IEEE Transactions on Information Theory*, 54(1):135–150, 2008.

[GRS19] Venkatesan Guruswami, Atri Rudra, and Madhu Sudan. Essential coding theory. *Draft available at https://cse.buffalo.edu/faculty/atri/courses/coding-theory/book*, 2019.

[GSTS03] Dan Gutfreund, Ronen Shaltiel, and Amnon Ta-Shma. Uniform hardness versus randomness tradeoffs for Arthur-Merlin games. *Computational Complexity*, 12(3-4):85–130, 2003.

[GUV09] Venkatesan Guruswami, Christopher Umans, and Salil Vadhan. Unbalanced expanders and randomness extractors from parvaresh–vardy codes. *Journal of the ACM (JACM)*, 56(4):20, 2009.

[GW97] Oded Goldreich and Avi Wigderson. Tiny families of functions with random properties: A quality-size trade-off for hashing. *Random Structures & Algorithms*, 11(4):315–343, 1997.

[GW02] Oded Goldreich and Avi Wigderson. Derandomization that is rarely wrong from short advice that is typically good. In *International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 209–223. Springer, 2002.

[GW14] Oded Goldreich and Avi Widgerson. On derandomizing algorithms that err extremely rarely. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC 2014)*, pages 109–118. ACM, 2014.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[HIOS15] Iftach Haitner, Yuval Ishai, Eran Omri, and Ronen Shaltiel. Parallel hashing via list recoverability. In *Annual Cryptology Conference*, pages 173–190. Springer, 2015.

[HLR07] Chun-Yuan Hsiao, Chi-Jen Lu, and Leonid Reyzin. Conditional computational entropy, or toward separating pseudoentropy from compressibility. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 169–186. Springer, 2007.

[HRZW17] Brett Hemenway, Noga Ron-Zewi, and Mary Wootters. Local list recovery of high-rate tensor codes & applications. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2017)*, pages 204–215. IEEE, 2017.

[HW18] Brett Hemenway and Mary Wootters. Linear-time list recovery of high-rate expander codes. *Information and Computation*, 261:202–218, 2018.

[IW97] Russell Impagliazzo and Avi Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC 1997)*, pages 220–229. ACM, 1997.

[Jus72] Jørn Justesen. Class of constructive asymptotically good algebraic codes. *IEEE Transactions on Information Theory*, 18(5):652–656, 1972.

[KMRZS17] Swastik Kopparty, Or Meir, Noga Ron-Zewi, and Shubhangi Saraf. High-rate locally correctable and locally testable codes with sub-polynomial query complexity. *Journal of the ACM (JACM)*, 64(2):11, 2017.

[KRZSW18] Swastik Kopparty, Noga Ron-Zewi, Shubhangi Saraf, and Mary Wootters. Improved decoding of folded reed-solomon and multiplicity codes. In *Proceedings of the 59th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2018)*, pages 212–223. IEEE, 2018.

[KU11] Kiran S. Kedlaya and Christopher Umans. Fast polynomial factorization and modular composition. *SIAM Journal on Computing*, 40(6):1767–1802, 2011.

[KvM02] Adam R. Klivans and Dieter van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM Journal on Computing*, 31(5):1501–1526, 2002.

[MRSV19] Jack Murtagh, Omer Reingold, Aaron Sidford, and Salil Vadhan. Deterministic approximation of random walks in small space. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2019)*, pages 42:1–42:22. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019.

[MV05] Peter B. Miltersen and N. Vinodchandran Variyam. Derandomizing Arthur–Merlin games using hitting sets. *Computational Complexity*, 14(3):256–279, 2005.

[NN93] Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM Journal on Computing*, 22(4):838–856, 1993.

[NW94] Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994.

[PF79] Nicholas Pippenger and Michael J Fischer. Relations among complexity measures. *Journal of the ACM (JACM)*, 26(2):361–381, 1979.

[Rei03] Omer Reingold. Personal Communication, 2003.

[RRV02] Ran Raz, Omer Reingold, and Salil Vadhan. Extracting all the randomness and reducing the error in trevisan's extractors. *Journal of Computer and System Sciences*, 65(1):97–128, 2002.

[RSW06] Omer Reingold, Ronen Shaltiel, and Avi Wigderson. Extracting randomness via repeated condensing. *SIAM Journal on Computing*, 35(5):1185–1209, 2006.

[RVW02] Omer Reingold, Salil Vadhan, and Avi Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders. *Annals of Mathematics*, pages 157–187, 2002.

[RW18] Atri Rudra and Mary Wootters. Average-radius list-recoverability of random linear codes. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2018)*, pages 644–662. SIAM, 2018.

[SGP15] Maciej Skorski, Alexander Golovnev, and Krzysztof Pietrzak. Condensed unpredictability. In *International Colloquium on Automata, Languages, and Programming*, pages 1046–1057. Springer, 2015.

[Sha04] Ronen Shaltiel. Recent developments in explicit constructions of extractors. In *Current Trends in Theoretical Computer Science: The Challenge of the New Century Vol 1: Algorithms and Complexity Vol 2: Formal Models and Semantics*, pages 189–228. World Scientific, 2004.

[Sko15] Maciej Skorski. Metric pseudoentropy: Characterizations, transformations and applications. In *International Conference on Information Theoretic Security*, pages 105–122. Springer, 2015.

[STV01] Madhu Sudan, Luca Trevisan, and Salil Vadhan. Pseudorandom generators without the XOR lemma. *Journal of Computer and System Sciences*, 62(2):236–266, 2001.

[SU05] Ronen Shaltiel and Christopher Umans. Simple extractors for all min-entropies and a new pseudorandom generator. *Journal of the ACM (JACM)*, 52(2):172–216, 2005.

[SU06] Ronen Shaltiel and Christopher Umans. Pseudorandomness for approximate counting and sampling. *Computational Complexity*, 15(4):298–341, 2006.

[SU07] Ronen Shaltiel and Christopher Umans. Low-end uniform hardness vs. randomness tradeoffs for AM. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC 2007)*, pages 430–439. ACM, 2007.

[Sud97] Madhu Sudan. Decoding of Reed–Solomon codes beyond the error-correction bound. *Journal of Complexity*, 13(1):180–193, 1997.

[Ta-17] Amnon Ta-Shma. Explicit, almost optimal, epsilon-balanced codes. In *Proceedings of the 49th Annual ACM Symposium on Theory of Computing (STOC 2017)*, pages 238–251. ACM, 2017.

[Tel18] Roei Tell. Quantified derandomization of linear threshold circuits. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing (STOC 2018)*, pages 855–865. ACM, 2018.

[Tel19] Roei Tell. Improved bounds for quantified derandomization of constant-depth circuits and polynomials. *Computational Complexity*, 28(2):259–343, 2019.

[Tre01] Luca Trevisan. Extractors and pseudorandom generators. *Journal of the ACM*, 48(4):860–879, 2001.

[TV00] Luca Trevisan and Salil Vadhan. Extracting randomness from samplable distributions. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS 2000)*, pages 32–42. IEEE, 2000.

[TZ04] Amnon Ta-Shma and David Zuckerman. Extractor codes. *IEEE Transactions on Information Theory*, 50(12):3015–3025, 2004.

[TZS06] Amnon Ta-Shma, David Zuckerman, and Shmuel Safra. Extractors from Reed–Muller codes. *Journal of Computer and System Sciences*, 72:786–812, 2006.

[Uma03] Christopher Umans. Pseudo-random generators for all hardnesses. *Journal of Computer and System Sciences*, 67(2):419–440, 2003.

[Vad12] Salil Vadhan. Pseudorandomness. *Foundations and Trends® in Theoretical Computer Science*, 7(1–3):1–336, 2012.

[VDHS13] Joris Van Der Hoeven and Éric Schost. Multi-point evaluation in higher dimensions. *Applicable Algebra in Engineering, Communication and Computing*, 24(1):37–52, 2013.

[vzGG13] Joachim von zur Gathen and Jürgen Gerhard. *Modern computer algebra*. Cambridge university press, 2013.

[Wee04] Hoeteck Wee. On pseudoentropy versus compressibility. In *Proceedings of the 19th Annual IEEE Conference on Computational Complexity (CCC 2004)*, pages 29–41. IEEE, 2004.

[Wic13] Daniel Wichs. Barriers in cryptography with weak, correlated and leaky sources. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, pages 111–126. ACM, 2013.

[Wil16] Ryan Williams. Strong ETH breaks with Merlin and Arthur: Short non-interactive proofs of batch evaluation. In *Proceedings of the 31st Annual Conference on Computational Complexity (CCC 2016)*, pages 2:1–2:17, 2016.

[Yao82] Andrew C. Yao. Theory and application of trapdoor functions. In *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 1982)*, pages 80–91. IEEE, 1982.

[Yek12] Sergey Yekhanin. Locally decodable codes. *Foundations and Trends in Theoretical Computer Science*, 6(3):139–255, 2012.

[Zuc97] David Zuckerman. Randomness-optimal oblivious sampling. *Random Structures and Algorithms*, 11(4):345–367, 1997.

[Zuc07] David Zuckerman. Linear degree extractors and the inapproximability of Max Clique and Chromatic Number. *Theory of Computing*, 3:103–128, 2007.

A Condensers and List Recoverable Codes

In this section we continue the discussion of Section 1.3, proving the equivalence between strong condensers and list recoverable codes.

Definition A.1. *Given a function* $E\colon \{0,1\}^a \times [n] \to \Sigma$*, we denote* $\mathcal{C}_E\colon \{0,1\}^a \to \Sigma^n$ *as the code mapping* $x \in \{0,1\}^a$ *to*

$$\mathcal{C}_E(x) = E(x,1) \circ \ldots \circ E(x,n).$$

Note that the rate of $\mathcal{C}_E$ *is* $\frac{a}{n \log |\Sigma|}$.

Theorem A.2. *Let* $\mathsf{Cond}\colon \{0,1\}^a \times [n] \to \Sigma$ *be some function so that* $\mathcal{C}_{\mathsf{Cond}}$ *is* (ε, ℓ, L) *list recoverable. Then,* Cond *is a strong*

$$\left(k = \log \frac{L}{\varepsilon}, k' = \log \frac{\ell}{n}, 2\varepsilon\right)$$

condenser. Recall that $\frac{\ell}{n} = 2^{k'}$ *is the average size of a list* $\mathcal{C}_{\mathsf{Cond}}$ *can handle.*

Proof: Assume towards a contradiction that Cond is not such a condenser, so there exists an (a,k) source X for which $\mathsf{Cond}(X,Y) \circ Y$ is not ε-close to having min-entropy $k'+d$, where Y is uniformly distributed over $[n]$. By Claim 2.26 there exist sets $S_1, \ldots, S_n \subseteq \Sigma$ satisfying $\sum_{i=1}^n |S_i| \leq n \cdot 2^{k'} = \ell$ such that

$$\Pr_{x \sim X, i \sim [n]}[\mathsf{Cond}(X,i) \in S_i] = \Pr_{x \sim X, i \sim [n]}[\mathcal{C}_{\mathsf{Cond}}(x)_i \in S_i] > 2\varepsilon.$$

By an averaging argument, there exists a set $G \subseteq \{0,1\}^a$ of density larger than ε such that for every $x \in G$,

$$\Pr_{i \sim [n]}[C(x)_i \in S_i] \geq \varepsilon.$$

By the list recovery properties of $\mathcal{C}_{\mathsf{Cond}}$ it must hold that $|G| \leq L$, however $|G| > \varepsilon \cdot |\mathrm{Supp}(X)| \geq \varepsilon \cdot 2^k \geq L$, in contradiction. ■

Theorem A.3. *Let* $\mathsf{Cond}\colon \{0,1\}^a \times [n] \to \Sigma$ *be a strong* (k, k', ε) *condenser. Then,* $\mathcal{C}_{\mathsf{Cond}}$ *is*

$$\left(4\varepsilon, \ell = 2\varepsilon n \cdot 2^{k'}, L = 2^k\right)$$

list recoverable.

Proof: Let $S_1, \ldots, S_n \subseteq \Sigma$ satisfy $\sum_{i=1}^n |S_i| \leq \ell$. Define the test $T \subseteq \Sigma \times [n]$ so that $(z,i) \in T$ if and only if $z \in S_i$. Let

$$\mathcal{L} = \left\{u \in \mathcal{C}_{\mathsf{Cond}} : \Pr_{i \sim [n]}[u_i \in S_i] \geq 4\varepsilon\right\}$$

and assume towards a contradiction that $|\mathcal{L}| \geq 2^k = L$. Note that the set $\mathcal{L}$ is in one-to-one correspondence with the set

$$\mathcal{A} = \left\{x \in \{0,1\}^a : \Pr_{i \sim [n]}[\mathsf{Cond}(x,i) \in S_i] \geq \frac{\ell}{n \cdot 2^{k'}} + 2\varepsilon\right\}.$$

Let Y be uniformly distributed over $[n]$. Then, on the one hand, $H^{\varepsilon}_{\infty}(\mathsf{Cond}(U_{\mathcal{A}}, Y) \circ Y) \geq k' + \log n$ so by Claim 2.26,

$$\Pr[\mathsf{Cond}(U_{\mathcal{A}}, Y) \circ Y \in T] \leq \varepsilon + |T| \cdot 2^{-k' - \log n} \leq \varepsilon + \frac{\ell}{n \cdot 2^{k'}}.$$

On the other hand,

$$\Pr[\mathsf{Cond}(U_{\mathcal{A}}, Y) \circ Y \in T] = \frac{1}{|\mathcal{A}|} \sum_{x \in \mathcal{A}} \Pr_{i \sim [n]}[\mathsf{Cond}(x, i) \in S_i] \geq \frac{\ell}{n \cdot 2^{k'}} + 2\varepsilon > \frac{\ell}{n \cdot 2^{k'}} + \varepsilon,$$

in contradiction. ∎

B The Goldreich-Wigderson Sampler

In this section we show that the Goldreich-Wigderson sampler given in Section 10 is indeed efficiently computable. For convenience we use the extractor terminology (see Lemma 2.21).

Theorem B.1 (following [GW97]). *For any positive integers n and $c < n$, and every $\varepsilon > 0$ and $m \leq n - O(c + \log \frac{1}{\varepsilon})$ there exists a $(k = n - c, \varepsilon)$ extractor $\mathsf{Ext}\colon \{0,1\}^n \times \{0,1\}^{\ell} \to \{0,1\}^m$ where $\ell = O(c + \log \frac{1}{\varepsilon})$.*

Furthermore, there exists a circuit $C\colon \{0,1\}^n \to (\{0,1\}^m)^{2^{\ell}}$ of size polynomial in n and $\log \frac{2^c}{\varepsilon}$ that on input $x \in \{0,1\}^n$ outputs the evaluation of Ext on x and all possible seeds.

To implement Ext, we need efficient constructions of expanders of arbitrary degree.

Claim B.2. *For every positive integer n and $0 < \lambda < 1$ there exists a connected d-regular undirected graph with n vertices which is a λ-expander, for $d = \mathrm{poly}(\frac{1}{\lambda})$. Given a vertex $x \in [n]$ and an edge label $i \in [d]$, the i-th neighbor of x can be computed in time $\log \frac{1}{\lambda} \cdot \mathrm{polylog(n)}$.*

Proof: Let G be a $\lambda_0 = \frac{1}{4}$ expander over $[n]$ with degree $d_0 = O(1)$ guaranteed to us by Theorem 2.23. For $t = \left\lceil \frac{1}{2} \log \frac{1}{\lambda} \right\rceil$, consider the graph G^t wherein each edge corresponds to taking a walk of length t on G. The graph G^t has the same vertex set, has degree $d_0^t = \mathrm{poly}(\frac{1}{\lambda})$, and it is known that $\lambda(G^t) \leq \lambda_0^t \leq \lambda$.

Given $x \in [n]$ and an edge label $i = (i_1, \ldots, i_t) \in [d_0]^t$, computing the i-th neighbor of x amounts to taking a walk on G_0 of length t from x according to the labels $i_1, \ldots, i_t$, which can be done in time $t \cdot \mathrm{polylog(n)}$. ∎

Proof of Theorem B.1: Ext is constructed as follows. Given positive integers n, c, and $\varepsilon > 0$, we use the following primitives.

- Let G be a λ-expander over the vertex set $\{0,1\}^m$, where $\lambda \leq \frac{\varepsilon^2}{4 \cdot 2^{c/2}}$. By Claim B.2, such a graph exists with degree $d = \left(\frac{2^c}{\varepsilon}\right)^a$ for some constant $a > 1$. Furthermore, the neighbourhood function of G, $\Gamma_G\colon \{0,1\}^m \times [d] \to \{0,1\}^m$, can be computed in time $\mathrm{poly}(m, \log \frac{1}{\varepsilon}, c)$.
- Let $\mathcal{H} \subseteq \{0,1\}^{n-m} \to [d]$ be a two-universal family of hash functions. In Lemma 2.31 we took $\mathcal{H}$ to be the set of all affine functions, and here, to get a smaller family, we use affine transformations with Toeplitz matrices (see, e.g., [Gol97, Section 3]). This gives $\log |\mathcal{H}| = n - m + 2 \log d - 1$. Computing the linear transformation can be done in time $\widetilde{O}(n - m)$.

Given $x \in \{0,1\}^n$ and $h \in \mathcal{H}$, we output

$$\mathsf{Ext}(x,h) = \Gamma_G\left(x_{[1,m]}, h(x_{[m+1,n]})\right).$$

The correctness follows from [GW97, Theorem 4.2]. For it to hold, we need to verify that

$$2^{-(n-m-c)} \leq \frac{\varepsilon^8}{64d},$$

and indeed choosing $n - m = (a+8)\log\frac{1}{\varepsilon} + 6ac$ satisfies the above. This choice of m puts a bound on $\log|\mathcal{H}|$:

$$\log|\mathcal{H}| = n - m + 2\log d - 1 \leq 8a\log\frac{1}{\varepsilon} + 8ac,$$

and so our seed length does satisfy $\ell \geq \log|\mathcal{H}|$. To output less than m bits, we can simply truncate the output.

Given $x \in \{0,1\}^n$ and $h \in \mathcal{H}$, computing $\mathsf{Ext}(x,h)$ takes time

$$O(n + \log|\mathcal{H}|) + \widetilde{O}(n-m) + \mathrm{poly}(m, \log(1/\varepsilon), c) = \mathrm{poly}(n, \log(1/\varepsilon))$$

and so can be done by a circuit of size $\mathrm{poly}(n, \log(1/\varepsilon))$ as well. Thus, iterating over all seeds can be done by a circuit of size

$$|\mathcal{H}| \cdot \mathrm{poly}(n, \log(1/\varepsilon)) + O(1) = \mathrm{poly}\left(n, \frac{1}{\varepsilon}, 2^c\right).$$

■

We instantiate Theorem B.1 as a sampler in the slightly unusual setting, where $\delta = 2^{-c}$ is constant and $\delta \gg \varepsilon$. We then get the following corollary.

Corollary B.3. *There exists a constant $b > 1$ such that the following holds for every positive integer N and any constants $\delta, \beta \in (0,1)$ and $0 < \gamma < \frac{1-\beta}{b}$. There exists an explicit bi-regular $(\delta, \varepsilon = N^{-\gamma})$ sampler*

$$\Gamma\colon [N] \times [D] \to [M = N^{\beta}]$$

for $D = N^{b\gamma}$, and a circuit $C\colon [N] \to [M]^D$ of size $\widetilde{O}(D)$ that on input $x \in [N]$ outputs $\Gamma(x,1), \ldots, \Gamma(x,D)$.

Moreover, Γ is bi-regular and the function $\Gamma^{-1}\colon [M] \times [a] \to [N]$ for $a = \frac{ND}{M} = N^{1+b\gamma-\beta}$ is fully-explicit, i.e., given $v \in [M]$ and $i \in [a]$, $\Gamma^{-1}(v,i)$ can be computed in $\mathrm{polylog}(N)$ time.

Proof: The first part of the corollary follows from applying Lemma 2.21 to Theorem B.1. Proving that Γ is bi-regular and establishing its running time can be done together. Assume Γ_G outputs m bits, i.e., before we truncate it to $\log M$ bits as the truncation changes neither the bi-regularity nor the running time. We use the notation of the proof of Theorem B.1. Given $v \in \{0,1\}^m$ and $i \in [a]$, interpret $i = (i_1, i_2)$, where $i_1 \in [d]$ and $i_2 \in \{0,1\}^{n-m}$. Then, outputting $\Gamma^{-1}(v,i)$ is computed as follows.

- Let $u \in \{0,1\}^m$ be such that $\Gamma_G(u, i_1) = v$. Working with expanders suitable edge labelings, this can be done in $\mathrm{poly}(m)$ time.
- Return the $x \in \{0,1\}^n$ for which $x_{[1,m]} = u$ and $x_{[m+1,n]} = i_2$.

For correctness, note that $\mathcal{H}$ is also one-universal, so in particular there exists $h \in \mathcal{H}$ such that $h(i_2) = i_1$.[22] Thus, indeed, $\Gamma_G(u, h(i_2)) = v$ and also distinct i-s give rise to distinct x-s. ■

[22] Here, we need to slightly modify $\mathcal{H}$ so that the linear transformations act only on nonzero vectors.

ECCC ISSN 1433-8092
https://eccc.weizmann.ac.il