Simple extractors via constructions of cryptographic pseudo-random generators

Marius Zimand [*]

Abstract. Trevisan has shown that constructions of pseudo-random generators from hard functions (the Nisan-Wigderson approach) also produce extractors. We show that constructions of pseudo-random generators from one-way permutations (the Blum-Micali-Yao approach) can be used for building extractors as well. Using this new technique we build extractors that do not use designs or polynomial-based error-correcting codes and that are very simple and efficient. For example, one extractor produces each output bit separately in $O(\log^2 n)$ time. These extractors work for weak sources with min entropy λn, for arbitrary constant $\lambda > 0$, have seed length $O(\log^2 n)$, and their output length is $\approx n^{\lambda/3}$.

1 Introduction

Extractors are procedures that remedy an imperfect source of random strings. They have been the object of intense research in the last years and several relevant techniques have been developed. This paper puts forward a new framework for constructing extractors based on a new connection between extractors and pseudo-random generators. Surely, in some regards, there are obvious similarities between the two concepts. A pseudo-random generator takes as input a short random string called the seed and outputs a long string that cannot be distinguished from a truly random string by any test that is computable by circuits of bounded size. An extractor has two inputs: (a) The first one comes from an imperfect (i.e., with biased bits and correlations among bits) distribution on binary strings of some length and it is called the weakly-random string; (b) the second one is a short random seed. The output is a long string that cannot be distinguished from a truly random string by *any* test. One difference between pseudo-random generators and extractors is the number of inputs (one versus two). From a technical point of view this difference is minor because the known constructions of pseudo-random generators implicitly do use an extra input which is a function that in some sense is computationally hard. The fundamental difference is in the randomness requirement for the output. Thus, while the output of a pseudo-random generator looks random in a complexity-theoretic way, the output of an extractor is random (or very close to random) in an absolute information-theoretic way. Consequently pseudo-random generators and extractors appear to belong to two very different worlds, and, for many years, the developments in the construction of pseudo-random generators and extractors went along distinct research lines.

Trevisan [Tre01] has made a breakthrough contribution in this area by observing that the (apparently superficial) similarity between extractors and pseudo-random generators extends to some of the methods to build the two kind of objects. For the reasons mentioned above, Trevisan's result has been extremely surprising. It has also been an isolated example of a transfer from the complexity theory standard arsenal of techniques to the information theoretical area. In this paper we extend Trevisan's observation and establish that, as far as construction methods are concerned, there is a truly close relationship between pseudo-random generators and extractors. Specifically, we show that the other major route (than the

---

[*]Department of Computer and Information Sciences, Towson University, Baltimore, MD. http://triton.towson.edu/~mzimand

one followed by Trevisan) that leads to pseudo-random generators (of a somewhat different kind) can also be used to construct extractors. Some explanations are in order at this point.

There are two known approaches for constructing pseudo-random generators. One approach uses as a building block a hard function f and, in one typical setting of parameters, for any given $k \in \mathbf{N}$, builds a pseudo-random generator g with outputs of length n that is secure against adversary tests computable in time n^k. The running time to compute $g(x)$ is $n^{k'}$, for some $k' > k$. This kind of pseudo-random generators can be used for derandomizing BPP computations. They cannot be used in cryptography, because in this setting, it is unwise to assume that the adversary is endowed with less computational power (n^k) than the legitimate users ($n^{k'}$). Henceforth we will call this type of pseudo-random generator a "derandomization pseudo-random generator" (also known as a Nisan-Wigderson pseudo-random generator).

The second approach uses as a building block a hard object of a more sophisticated type, namely a one-way function (the hardness of such a function f consists in the difficulty to invert it, but f must satisfy an additional property, namely, it should be easy to calculate $f(x)$ given x). It is known that given a one-way function, one can construct a pseudo-random generator [HILL99]. An easier construction produces a pseudo-random generator from any one-way length-preserving permutation. This second approach has the disadvantage that is using as a building block a more demanding type of object. The advantage of the method is that a pseudo-random generator g constructed in this way can be used in cryptography because $g(x)$ can be calculated in time significantly shorter than the time an adversary must spend to distinguish $g(x)$ from a truly random string. Henceforth we will call this type of pseudo-random generator a "crypto pseudo-random generator" (also known as a Blum-Micali-Yao pseudo-random generator).

Trevisan has shown that the known methods for constructing derandomization pseudo-random generators also produce extractors. More precisely, he has shown that the constructions of pseudo-random generators from hard functions given by Nisan and Wigderson [NW94] and Impagliazzo and Wigderson [IW97] can be used almost directly to produce extractors. His method has been extended in a number of papers to build extractors with increasingly better parameters (see the survey paper by Shaltiel [Sha02]). In the paper [Tre99], the conference version of [Tre01], Trevisan has suggested that the methods to construct crypto pseudo-random generator cannot be used to build extractors. We show that in fact they can, at least for a combination of parameters that, even though not optimal, is not trivial. Moreover, we show that the extractors constructed in this way are very simple and efficient.

An extractor can be viewed as a bipartite graph and is therefore a static finite object that can be constructed trivially by exhaustive search. We are looking however for efficient constructions. Typically "efficient" means "polynomial time," but one can envision different levels of efficiency and one remarkable such level would certainly be "linear time." The first extractor built in this paper follows almost directly the classical construction of a pseudo-random generator from a one-way permutation, and comes very close to this level of efficiency: Viewed as a procedure, it runs in $O(n \log n)$ time (in the standard RAM model). In addition it is very simple. The following is a complete description of it. The input consists of the weakly-random string X, of length $n = \tilde{n} 2^{\tilde{n}}$ for some integer $\tilde{n}$, and of the seed $((x_1, \ldots, x_\ell), r)$, with $|x_i| = \tilde{n}$, $\ell = O(\tilde{n})$, and $|r| = \ell \tilde{n}$. We view X as a function $X : \{0,1\}^{\tilde{n}} \to \{0,1\}^{\tilde{n}}$, and, using the standard procedure, we transform X into a circular permutation $R : \{0,1\}^{\tilde{n}} \to \{0,1\}^{\tilde{n}}$. For $i = 0$ to $m - 1 = n^{\Omega(1)}$, we calculate b_i as the inner product modulo 2 of r and $(R^i(x_1) \ldots R^i(x_\ell))$. The output is $b_0 \ldots b_{m-1}$.

Another remarkable level of efficiency which has received a lot of attention recently is that of sublinear time. It may be the case that in some applications we only need the i-th bit from the sequence of random bits that are extracted from the weakly-random string. We would like to obtain this bit in time polynomial in the length of the index i, which typically means polylog time in the input length (under the assumption that each input bit can be accessed in one time unit). By analogy with the case

of list-decodable codes, we call an extractor with this property, a *bitwise locally computable extractor*.[1] The second extractor that we build is of this type. The algorithm deviates from the direct construction of a pseudo-random generator from a one-way function. However it relies on a basic idea used in the construction of the first extractor, combined with the idea of taking consecutive inputs of the hard function as in the extractor of Ta-Shma, Zuckerman and Safra [TSZS01]. This second extractor is even simpler and its complete description is as follows. The input consists of the weakly-random string X of length $n = \tilde{n} \cdot 2^{\tilde{n}}$, for some natural number $\tilde{n}$, and of the seed $((x_1, \ldots, x_\ell), r)$, with $|x_i| = \tilde{n}$, for all i, $\ell = O(\tilde{n})$, and $|r| = \ell \tilde{n}$. We view X as the truth-table of a function $X : \{0,1\}^{\tilde{n}} \to \{0,1\}^{\tilde{n}}$. For $i = 0$ to $m - 1 = n^{\Omega(1)}$, we calculate b_i as the inner product modulo 2 of r and $(X(x_1 + i), \ldots, X(x_\ell + i))$, where the addition is done modulo $2^{\tilde{n}}$. The output is $b_0 \ldots b_{m-1}$.

The parameters of the extractors constructed in this paper are not optimal. Both extractors that have been described above work for weak sources having min-entropy λn, for arbitrary constant $\lambda > 0$, use a random seed of length $O(\log^2 n)$, and the output length is approximately $n^{\lambda/3}$. A variant of the second extractor has seed length $O(\log n)$ (here, for simplicity, we assume that the extractor's error parameter ϵ is a constant), but the output length reduces to $2^{O(\sqrt{\log n})}$.

Lu's extractor [Lu04] coupled with the constructions of designs from the paper of Hartman and Raz [HR03] can be seen to be also a bitwise locally computable extractor with parameters similar to those of our second extractor (note that the designs in [HR03] appear to imply extractors with seed length $\Omega(\log^2 n)$). Lu's extractor is using expander graphs and the designs from [HR03] need somewhat unwieldy algebraic objects. It seems to us that the extractors presented in this paper are simpler than all the extractors from the literature.[2] At the highest level of abstraction, our extractors follow the "reconstruction paradigm" (see [Sha02]) typical to Trevisan's extractor and to its improvements [RRV99, TSZS01, SU01]. The major differences are that our extractors avoid (1) the use of designs (in this respect they are similar to the extractors in [TSZS01] and [SU01]), and, perhaps more strikingly, (2) the encoding of the weakly-random string with an error-correcting code having a good list-decoding property. Our extractors can be implemented very easily and are thus suitable for practical applications. For example, they can be utilized to generate one-time pad keys in cryptosystems based on the bounded-storage model (see the papers of Lu [Lu04] and Vadhan [Vad04]), or for constructions of error-correcting codes using the scheme in [TSZ01] (the second extractor built in this paper is actually a strong extractor— for definition see, for example [Sha02]—as required by this scheme). They may also have theoretical applications in situations where the kind of efficiency achieved by our extractors is essential.

2 Definitions

Notations: $x \odot y$ denotes the concatenation of the strings x and y, $|x|$ denotes the length of the string x, and $\|A\|$ denotes the cardinality of the set A. We remind the standard definition of an extractor. Let $n \in \mathbf{N}$. Let X_n, Y_n be two distributions on Σ^n. The statistical distance between X_n and Y_n is denoted $\Delta_{\mathrm{stat}}(X_n, Y_n)$ and is defined by $\Delta_{\mathrm{stat}}(X_n, Y_n) = \max_{A \subseteq \{0,1\}^n} |\mathrm{Prob}(X_n \in A) - \mathrm{Prob}(Y_n \in A)|$.

If we view the sets $A \subseteq \{0,1\}^n$ as *statistical tests*, then, by the above expression, $\Delta_{\mathrm{stat}}(X_n, Y_n) \le \epsilon$ signifies that no test can distinguish between the distributions X_n and Y_n except with a small bias ϵ. If we restrict to tests that can be calculated by bounded circuits, we obtain the notion of computational distance between distributions. Namely, the computational distance between X_n and Y_n relative to size S is denoted $\Delta_{\mathrm{comp},S}(X_n, Y_n)$ and is defined by $\Delta_{\mathrm{comp},S}(X_n, Y_n) =$

---

[1]The simpler name *locally computable extractor* is already taken by a different kind of efficient extractors, namely by extractors whose outputs depend only on a small number of the bits of the weakly random string, see [Vad04], [Lu04].

[2]We note that Dziembowski and Maurer [DM04] give a similarly simple construction of an object that is related to extractors.

$\max |\mathrm{Prob}(C(X_n) = 1) - \mathrm{Prob}(C(Y_n) = 1)|$, where the maximum is taken over all circuits C of size $\leq S$. Abusing notation, we identify a circuit C with the set of strings x for which $C(x) = 1$. Thus, $x \in C$ is equivalent to $C(x) = 1$.

The min-entropy of a distribution is a good indicator of the degree of randomness of the distribution. The min-entropy of a random variable taking values in $\{0,1\}^n$ is given by $\min \left\{ \log \frac{1}{\mathrm{Prob}(X=a)} \,\middle|\, a \in \{0,1\}^n, \mathrm{Prob}(X = a) \neq 0 \right\}$.

Thus if X has min-entropy $\geq k$, then for all a in the range of X, $\mathrm{Prob}(X = a) \leq 1/2^k$. For each $n \in \mathbf{N}$, let U_n denote the uniform distribution over $\{0,1\}^n$. We are now ready to define an extractor formally.

Definition 2.1 (Extractor) *The values n, k, d, m are integer parameters, and $\epsilon > 0$ is a real number parameter. A function $E \colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a (k, ϵ)-extractor if for every distribution X on $\{0,1\}^n$ with min-entropy at least k, the distribution $E(X, U_d)$ is ϵ-close to the uniform distribution U_m in the statistical sense, i.e., $\Delta_{\mathrm{stat}}(E(X, U_d), U_m) \leq \epsilon$.*

Thus, an extractor has as input (a) a string x produced by an imperfect source with distribution X, where the defect of the distribution is measured by $k = \mathrm{min\text{-}entropy}(X)$, and (b) a random seed y of length d. The output is $E(x, y)$, a string of length m. The key property is that, for every subset $W \subseteq \Sigma^m$,

$$|\mathrm{Prob}_{x \in_X \{0,1\}^n, y \in \{0,1\}^d}(E(x,y) \in W) - \mathrm{Prob}_{z \in \Sigma^m}(z \in W)| \leq \epsilon. \tag{1}$$

If we consider n and k as given (these are the parameters of the source), it is desirable that d is small, m is large, and ϵ is small. It can be shown nonconstructively that for every $k \leq n$ and $\epsilon > 0$, there exist extractors with $d = \log(n - k) + 2\log(1/\epsilon) + O(1)$ and $m = k + d - 2\log(1/\epsilon) - O(1)$. It has been shown [RTS00] that these parameters are optimal. Furthermore, we want the family of extractors to be efficiently computable. For simplicity, we have defined individual extractors. However, implicitely we think of a family of extractors indexed by n and with the other parameters being uniform functions of n. In this way we can talk about efficient constructions of extractors by looking at the time and space required to calculate $E(x, y)$ as functions of n.

An extractor $E \colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ can also be viewed as a regular bipartite graph where the set of "left" nodes is $V_{\mathrm{left}} = \{0,1\}^n$ and the set of "right" nodes is $V_{\mathrm{right}} = \{0,1\}^m$. The degree of each node in V_{left} is 2^d, and two nodes $x \in V_{\mathrm{left}}$ and $z \in V_{\mathrm{right}}$ are connected if there is $y \in \{0,1\}^d$ such that $E(x, y) = z$. We can imagine that each $x \in V_{\mathrm{left}} = \{0,1\}^n$ is throwing 2^d arrows at $V_{\mathrm{right}} = \{0,1\}^m$.

To understand better Equation (1), let us look deeper into the structure of an extractor. We fix parameters n, d, m and ϵ and a function $E \colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$. Let us consider an arbitrary set $W \subseteq \{0,1\}^m$ and a string $x \in \{0,1\}^n$. We say that x hits W ϵ-correctly via E if the fraction of outgoing edges from x that land in W is ϵ-close to the fraction $\|W\|/\|\{0,1\}^m\|$, i.e.,

$$\left| \frac{\|\{E(x,y) \mid y \in \{0,1\}^d\} \cap W\|}{\|\{0,1\}^d\|} - \frac{\|W\|}{\|\{0,1\}^m\|} \right| \leq \epsilon.$$

If we look at a fixed x, it cannot hold that for every $W \subseteq \{0,1\}^m$, x hits W ϵ-correctly (for example, take $W = \{E(x,y) \mid y \in \{0,1\}^d\}$). Fortunately, for E to be an extractor, all we need is that any $W \subseteq \{0,1\}^m$ is hit ϵ-correctly by most $x \in \{0,1\}^n$. The following lemma has appeared more or less explicitly in the literature (see, for example, [Sha02]).

Lemma 2.2 *Let $E \colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ and $\epsilon > 0$. Suppose that for every $W \subseteq \{0,1\}^m$, the number of $x \in \{0,1\}^n$ that do not hit W ϵ-correctly via E is at most 2^t, for some t. Then E is a $(t + \log(1/\epsilon), 2\epsilon)$-extractor.*

4

Proof Let X be a distribution on $\{0,1\}^n$ with min-entropy at least $t + \log(1/\epsilon)$ and let W be a subset of $\{0,1\}^m$. There are at most 2^t x's that do not hit W ϵ-correctly and the distribution X allocates to these x's a mass probability of at most $2^t \cdot 2^{-(t+\log(1/\epsilon))} = \epsilon$. We have,

$$\text{Prob}_{x \in_X \{0,1\}^n, y \in \{0,1\}^d}(E(x,y) \in W)$$
$$= \quad \text{Prob}_{x \in_X \{0,1\}^n, y \in \{0,1\}^d}(E(x,y) \in W \text{ and } x \text{ hits } W \ \epsilon\text{-correctly})$$
$$+ \quad \text{Prob}_{x \in_X \{0,1\}^n, y \in \{0,1\}^d}(E(x,y) \in W \text{ and } x \text{ does not hit } W \ \epsilon\text{-correctly}).$$

The first term in the right hand side is between $\frac{\|W\|}{\|\{0,1\}^m\|} - \epsilon$ and $\frac{\|W\|}{\|\{0,1\}^m\|} + \epsilon$, because for each x that hits W ϵ-correctly,

$$\text{Prob}_{y \in \{0,1\}^d}(E(x,y) \in W) \in \left[\frac{\|W\|}{\|\{0,1\}^m\|} - \epsilon, \frac{\|W\|}{\|\{0,1\}^m\|} + \epsilon \right].$$

The second term is bounded by

$$\text{Prob}_{x \in_X \{0,1\}^n, y \in \{0,1\}^d}(x \text{ does not hit } W \ \epsilon\text{-correctly}),$$

which is, as we have seen, between 0 and ϵ. Plugging these estimates in the above equation, we obtain that

$$\left| \text{Prob}_{x \in_X \{0,1\}^n, y \in \{0,1\}^d}(E(x,y) \in W) - \frac{\|W\|}{\|\{0,1\}^m\|} \right| \leq 2\epsilon.$$

Thus, E is a $(t + \log \frac{1}{\epsilon}, 2\epsilon)$-extractor. ∎

We recall the definition of a pseudo-random generator.

Definition 2.3 (Pseudo-random generator) *Let $\ell, L, S \in \mathbf{N}$ and $\epsilon > 0$ be parameters. A function $g \colon \Sigma^l \to \Sigma^L$ is a pseudo-random generator with security (ϵ, S) if $\Delta_{\text{comp},S}(g(U_\ell), U_L) \leq \epsilon$.*

3 Overview and comparison with Trevisan's approach

Trevisan's method is based on the constructions of pseudo-random generators from hard functions given in [NW94] and in [IW97]. These constructions use a function f as a block-box and construct from it a function g_f that stretches the input (i.e., $|g_f(x)| \gg |x|$) and which has the following property. If there exists a circuit D that distinguishes $g_f(x)$, when x is randomly chosen in the domain of g_f, from the uniform distribution, then there is a small circuit A, which uses D as a subroutine, such that A calculates f (or an approximation of f, depending on whether we are using the method in [IW97] or the one in [NW94]). Therefore if f is a hard function, there can be no circuit D as above of small size and thus g_f is a pseudo-random generator. Trevisan has observed that (1) the truth-table of f can be viewed as a string produced by a weak source that can serve as an extra input of the pseudo-random generator, and (2) the circuit A invoking D can be considered as a special type of a circuit that is endowed with D-gates. By a standard counting argument, it can be shown that, for any circuit D, regardless of its size, the set of functions that can be calculated by small circuits with D-gates is small. A circuit D can be viewed statically as a statistical test (more exactly, the statistical test associated to the circuit D is the set of strings accepted by D). In the new terminology, the fact that D distinguishes the distribution of $g_f(x)$ from the uniform distribution with ϵ bias can be restated as "f does not hit D ϵ-correctly via g." The main property mentioned above can be restated as saying that the set of functions f that do not hit D ϵ-correctly is included in the set of functions computable by small circuits with D-gates. Since the latter set is small, the former set is small as well, and thus, by Lemma 2.2, the construction yields an extractor. In a nutshell, Trevisan's method replaces hard functions (a complexity-theoretic

concept) with random functions (an information-theoretic concept) and takes advantage of the fact that a random function is hard and thus the construction carries over in the new setting.

We would like to follow a similar approach for the construction of crypto pseudo-random generators from one-way permutations. Those constructions do use a one-way permutation R as a black box to construct a pseudo-random generator g_R, and thus a truth-table of R can be considered as an extra input of the pseudo-random generator. Also, the proof is a reduction that shows that if a circuit D distinguishes $g_R(x)$ from the uniform distribution, then there is a small circuit A, invoking the circuit D, that inverts R on a large fraction of inputs. To close the proof in a similar way to Trevisan's approach, we would need to argue that the vast majority of permutations are one-way. It seems that we hit a major obstacle because, unlike the case of hard functions, it is not currently known if even a single one-way function exists (and we are seeking an unconditional proof for the extractors that we build). We go around this obstacle by allowing algorithms to have oracle access to the function they compute. Thus, in the above analysis, the circuit A, in addition to invoking the circuit D, will also have oracle access to the permutation R. In this setting all permutations are easy to compute because, obviously, there is a trivial constant-time algorithm that, for any permutation $R : \{0,1\}^n \to \{0,1\}^n$, given the possibility to query R, calculates $R(x)$. We need to argue that only few permutations R are invertible by algorithms that can query R in a bounded fashion. More precisely we need to estimate the size of the set of permutations $R : \{0,1\}^n \to \{0,1\}^n$ that can be inverted on a set of T elements in $\{0,1\}^n$ by circuits that can pose Q queries to R. This problem has been considered by Impagliazzo [Imp96] and by Gennaro and Trevisan [GT00]. Their techniques seem to work for the case $T \cdot Q < 2^n$ and lead to extractors that work only for sources with high min-entropy.[3]

We obtain better parameters by restricting the type of one-way permutations and the type of circuits that attempt to invert them. A second look at the standard construction of Blum-Micali-Yao pseudo-random generators reveals that the circuit A with D-gates manages to determine x using only the values $R(x), R^2(x), \ldots, R^m(x)$ (where m is the generator's output length). It is thus enough to consider only circuits that use this pattern of queries to the permutation R. Intuitively, for a random permutation R, the value of x should be almost independent of the values of $R(x), R^2(x), \ldots, R^m(x)$, and thus, a circuit A restricted as above cannot invert but a very small fraction of permutations. If we take R to be a random circular permutation, the above intuition can be easily turned into a proof based on a Kolmogorov-complexity counting argument. A circular permutation $R : \{0,1\}^n \to \{0,1\}^n$ is fully specified by the sequence $(R(1), R^2(1), \ldots, R^{N-1}(1))$, where $N = 2^n$. If a circuit A restricted as above inverts $R(x)$ for all x, then the permutation R is determined by the last m values in the above sequence, namely $R^{N-m}(1), R^{N-(m-1)}(1), \ldots, R^{N-1}(1)$. Indeed, given the above values, the circuit A can determine $R^{N-m-1}(1)$, which is $R^{-1}(R^{N-m}(1))$, and then $R^{N-m-2}(1)$, and so on till $R(1)$ is determined. Therefore such a permutation R, given the circuit A, can be described concisely using only $m \cdot n$ bits (for specifying, as discussed, the last m elements in the above sequence). In fact, in our case, the circuit A does not invert $R(x)$ for all $x \in \{0,1\}^n$, and, therefore, the values of R at the points where the inversion fails have to be included in the description. A further complication is that even for the successful cases, the circuit A only list-inverts $R(x)$, which means that A on input $R(x)$ produces a relatively short list of elements, one of which is x. Thus, one also has to include in the description of R the rank of x in the list produced by A. The quantitative analysis of the standard construction of a crypto pseudo-random generator shows that if the permutation R does not hit D ϵ-correctly, then the circuit A with D-gates is only able to produce for an ϵ/m fraction of $R(x), x \in \{0,1\}^n$, a list with m^2/ϵ^2 elements one of which is x. For interesting values of m (the pseudo generator's output length), the ϵ/m fraction is too small and needs to be amplified to a value of the form $(1 - \delta)$, for a small constant δ. This

---

[3]On the other hand, these extractors have the interesting property that their output looks random even to statistical tests that have some type of access to the weakly-random string. These results will be reported in a separate paper.

can be done by employing another technique that is well-known in the context of one-way functions. Namely, we use Yao's method of converting a weak one-way function into a strong one-way function by taking the direct product. In other words, we start with a circular permutation R, define (the direct product) $\overline{R}(x_1, \ldots, x_\ell) = R(x_1) \odot \ldots \odot R(x_\ell)$ (where $\odot$ denotes concatenation), for some appropriate value of ℓ, and use $\overline{R}$ in the definition of the extractor (instead of R in our tentative plan sketched above). It can be shown that, for $\ell = O((1/\delta)\log(1/\gamma))$, if a circuit A list-inverts $(y_1, \ldots, y_\ell)$, with list size $T = m^2/\epsilon^2$, for a $\gamma = \epsilon/m$ fraction of ℓ-tuples $(y_1, \ldots, y_\ell) \in (\{0,1\}^n)^\ell$, then there is a probabilistic algorithm A' that list-inverts $R(x)$ with list size $O(n \cdot T \cdot (1/\delta) \cdot (1/\gamma) \cdot \log(1/\gamma))$ for a $(1-\delta)$ fraction of $x \in \{0,1\}^n$. By fixing the random bits and the queries that depend on these random bits, we can obtain a brief description of R as in our first tentative plan. It follows that only few permutations R can hit D ϵ-incorrectly and, therefore, by Lemma 2.2, we have almost obtained an extractor (we also need to convert an arbitrary function $X : \{0,1\}^n \to \{0,1\}^n$ into a circular permutation $R : \{0,1\}^n \to \{0,1\}^n$, which is an easy task).

Briefly, the proof relies on the fact that if a permutation R does not hit D ϵ-correctly, then there must be a very strong dependency between the "consecutive" values $x, R(x), R^2(x), \ldots, R^m(x)$, for many $x \in \{0,1\}^n$, and only few permutations R exhibit such dependencies.

The second extractor starts from this idea and the observation that, for the sake of building an extractor, we can work with a function $\overline{X}$ (i.e., not necessarily a permutation) and consider consecutive values $\overline{X}(\overline{x}), \overline{X}(\overline{x}+1), \ldots, \overline{X}(\overline{x}+m)$, as in the extractor of Ta-Shma, Zuckerman, and Safra [TSZS01]. That extractor (as well as all the extractors using the "reconstruction paradigm") takes $\overline{X}$ to be the encoding of an arbitrary function X with a good list-decoding property and some other special algebraic properties. This is necessary, among other things, for the same type of amplification as in our discussion above. We use instead a direct-product construction that is much simpler to implement (however, the cost is a longer seed length).

4   Restricted permutations, restricted circuits

The space from where we randomly choose permutations consists of permutations of a special form. First we consider the set CIRC of all circular permutations $R : \{0,1\}^n \to \{0,1\}^n$. Next, for some parameter $\ell \in \mathbf{N}$, we take the ℓ-direct product of CIRC. This means that for any $R \in$ CIRC, we define $\overline{R}_\ell : \{0,1\}^{\ell n} \to \{0,1\}^{\ell n}$ by $\overline{R}_\ell(x_1 \odot x_2 \odot \ldots \odot x_\ell) = R(x_1) \odot R(x_2) \odot \ldots \odot R(x_\ell)$. We let PERM_ℓ be the set $\{\overline{R}_\ell \mid R \in \mathrm{CIRC}\}$. We will drop the subscript ℓ when its value is clear from the context or when it is not relevant in the discussion.

We want to argue that no circuit that queries $\overline{R}$ in a restricted way can invert a "large" fraction of $\overline{R}(\overline{x})$ except for a "small" fraction of permutations $\overline{R}$ in PERM. In order to obtain adequate values for "large" and "small" we will impose the following restriction on the pattern of queries that the circuit can make.

Definition 4.1 *An oracle circuit C on inputs of length at least $\ell \cdot n$ is L-restricted if on any input x and for all oracles $\overline{R} \in \mathrm{PERM}_\ell$, C only queries $x_{\mathrm{first}}, \overline{R}(x_{\mathrm{first}}), \overline{R}^2(x_{\mathrm{first}}), \ldots, \overline{R}^{L-1}(x_{\mathrm{first}})$, where x_{first} is the string consisting of the first $\ell \cdot n$ bits of x.*

We will allow the circuits to attempt to invert $\overline{R}$ in a weaker form: On input $\overline{R}(\overline{x})$, $C^{\overline{R}}$ outputs a small list of strings one of which (in case C succeeds) is $\overline{x}$. When this event happens, we say that $C^{\overline{R}}$ *list-inverts* $\overline{x}$. We are interested in estimating the number of permutations $\overline{R} \in$ PERM so that $C^{\overline{R}}$ list-inverts $\overline{R}(\overline{x})$ for a large fraction of $\overline{x}$.

Definition 4.2 *Let C be an oracle circuit. A permutation $\overline{R}$ is (γ, T)-good for C if for at least a γ fraction of $\overline{x} \in \{0,1\}^{\ell n}$, $C^{\overline{R}}$ on input $\overline{R}(\overline{x})$ outputs a list of T elements that contains $\overline{x}$.*

We will show that a permutation that is (γ, T)-good for a restricted circuit C admits a short description conditioned by C being given. This leads immediately to an estimation of the number of permutations $\overline{R}$ that are (γ, T)-good for a given restricted circuit C.

Lemma 4.3 *Let $\gamma > 0$, $n \in \mathbf{N}, L \in \mathbf{N}$, and $T \in \mathbf{N}$. Let $N = 2^n$. Let $\delta > 0$ and let $\ell = \left\lceil \frac{3}{\delta} \cdot \log\left(\frac{2}{\gamma}\right) \right\rceil$. Assume $\delta \geq 2e^{-n}$ and $\ell < L + 1$. Let C be an L-restricted circuit, having inputs of length ℓn, and let $\overline{R} \in \mathrm{PERM}_\ell$ be a permutation that is (γ, T)-good for C. Then, given C and ℓ, $\overline{R}$ can be described using a number of bits that is bounded by $2\delta N n + L n + N \log n + (\log 6)N + N \log(1/\delta) + N \log\log(2/\gamma) + N \log(1/\gamma) + N \log T + 18n^2 \cdot L \cdot \frac{1}{\gamma} \cdot \left(\frac{1}{\delta}\right)^2 \left(\log\frac{2}{\gamma}\right)^2$.*

Proof Since $\overline{R}$ is the ℓ-direct product of R, it is enough to present a short description of R. We will first show that the assumption that $C^{\overline{R}}$ list-inverts a γ-fraction of $\overline{R}(\overline{x})$ with $\overline{x} \in \{0,1\}^{\ell n}$ implies that there exists an oracle circuit B so that B^R list-inverts a $(1 - \delta)$ fraction of $R(x)$ with $x \in \{0,1\}^n$. The circuit B is not L-restricted but it has a similar property. Namely the circuit B makes two categories of queries to the oracle R. The first category consists of a set of queries that do not depend on the input. The second category depends on the input y and it consists of the queries $y, R(y), \ldots, R^{L-1}(y)$. The circuit B is helpful in producing the concise description of R that we are seeking. Note that the permutation $R \in \mathrm{CIRC}$ is determined by the vector $(R(1), R^2(1), \ldots, R^{(N-1)}(1))$. This vector will be described in the following way. The last L entries are described by themselves. Then we describe each of the other entries y one at a time going backwards in the vector. Suppose that $R(y), R^2(y), \ldots, R^L(y)$ are already described. We describe now the preceding term in the sequence, which is y. There are two cases.

Case 1: B^R list-inverts $R(y)$. In this case y is determined by its rank in the T-list produced by B^R on input $R(y)$. The computation of B^R on input $R(y)$ depends on the strings $R(y), \ldots, R^L(y)$ (which are already described) and on the value of R on the fixed queries (these values have to be given in the description, but they are common to all the entries in the vector).

Case 2: If B^R fails to list-invert y (this will happen only for a small fraction δ of y's), then y is described by itself.

We will show that this description policy needs the asserted number of bits.

We proceed with the technical details. The amplification of the fraction of inverted inputs from γ to $(1 - \delta)$ is done using the well-known technique of producing strong one-way functions from weak one-way functions (Yao [Yao82]). Let $w = 6 \cdot \frac{1}{\delta} \cdot \log\left(2/\gamma\right) \cdot \frac{1}{\gamma}$. Recall that $\ell = \left\lceil \frac{3}{\delta} \cdot \log\left(\frac{2}{\gamma}\right) \right\rceil$. It holds that, for $n \geq \ln(2/\delta)$ and $\delta < 1/3$, $\frac{\ell}{w} < \gamma - (1 - \delta + e^{-n})^\ell$. Let INV be the set of strings $\overline{R}(\overline{x})$ on which $C^{\overline{R}}$ outputs a T-list that contains $\overline{x}$. From the hypothesis, we know that $\|\mathrm{INV}\| \geq \gamma \cdot 2^{\ell n}$. We define the following probabilistic algorithm D.

Input: $y = R(x)$, for some $x \in \Sigma^n$. Goal: Find a short list that contains x.
LIST $= \emptyset$.
Repeat the following $n \cdot w$ times.
      Pick random $i \in \{1, \ldots, \ell\}$.
      Pick $\ell - 1$ random strings in $\{0,1\}^n$ denoted $y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_\ell$.
      Calculate $Y = y_1 \odot \ldots \odot y_{i-1} \odot R(x) \odot y_{i+1} \odot \ldots \odot y_\ell$.
      Call the circuit $C^{\overline{R}}$ to invert Y. $C^{\overline{R}}$ returns a T-list of ℓ-tuples in $(\{0,1\}^n)^\ell$.
      (Note: In case of success one of these ℓ-tuples is
      $(R^{-1}(y_1), \ldots, R^{-1}(y_{i-1}), x, R^{-1}(y_{i-1}), \ldots, R^{-1}(y_\ell))$.)
      Add to LIST the i-th component of every ℓ-tuple in the list produced by $C^{\overline{R}}$.
End Repeat

We say that the above algorithm is successful on input $y = R(x)$ if, at the conclusion of the algorithm, LIST contains x. We estimate the success probability of the above circuit on input $y = R(x)$.

Let $N(y)$ be the multiset of ℓ-tuples having y as one component where the multiplicity of a tuple is the number of occurrences of y in the tuple. For a set $A \subseteq \{0,1\}^n$, we define $N(A) = \bigcup_{y \in A} N(y)$. It can be seen that, for all $y \in \{0,1\}^n$, $\|N(y)\| = \ell \cdot 2^{n(\ell-1)}$. We define

$$V_w = \left\{ y \in \{0,1\}^n \mid \frac{\|N(y) \cap \mathrm{INV}\|}{\|N(y)\|} \geq \frac{1}{w} \right\}.$$

Let $\overline{V_w}$ be the complement of V_w. We have

$$
\begin{aligned}
\|N(\overline{V_w}) \cap \mathrm{INV}\| & \leq \sum_{y \in \overline{V_w}} \|N(y) \cap \mathrm{INV}\| \\
& < 2^n \cdot \frac{1}{w} \cdot (\ell \cdot 2^{n(\ell-1)}) \\
& = \frac{\ell}{w} \cdot \|(\Sigma^n)^\ell\|.
\end{aligned}
$$

We show that this is possible only if $\|\overline{V_w}\| < (\delta - e^{-n}) \cdot \|\Sigma^n\|$. Let $A \subseteq \Sigma^n$ be a set with $\|A\| \geq (\delta - e^n) \cdot \|\Sigma^n\|$. We observe that $N(A)$ covers an overwhelming fraction of $(\Sigma^n)^\ell$. Indeed, note that the probability that a tuple $(y_1, \ldots, y_\ell)$ is not in $N(A)$ is equal to the probability of the event "$y_1 \notin A \ \wedge \ldots \wedge \ y_\ell \notin A$" which is bounded by $(1 - \delta + e^{-n})^\ell$. Therefore, the complementary set of $N(A)$, denoted $\overline{N(A)}$, satisfies

$$\|\overline{N(A)}\| < (1 - \delta + e^{-n})^\ell \cdot \|(\Sigma^n)^\ell\|.$$

Then,

$$
\begin{aligned}
\|N(A) \cap \mathrm{INV}\| & = \|\mathrm{INV}\| - \|\mathrm{INV} \cap \overline{N(A)}\| \\
& \geq \|\mathrm{INV}\| - \|\overline{N(A)}\| \\
& > \left[\gamma - (1 - \delta + e^{-n})^\ell \right] \|(\Sigma^n)^\ell\|.
\end{aligned}
$$

Recall that $\ell/w < \left[\gamma - (1 - \delta + e^{-n})^\ell \right]$. Thus necessarily $\|\overline{V_w}\| < (\delta - e^{-n}) \cdot 2^n$.

On input $y = R(x)$, at each iteration, the algorithm chooses uniformly at random $\overline{y}$ in $N(y)$. The circuit C is invoked next to invert $\overline{R}(\overline{y})$. The algorithm succeeds if and only if $\overline{y} \in \mathrm{INV}$. For all $y \in V_w$, $\frac{\|N(y) \cap \mathrm{INV}\|}{\|N(y)\|} \geq \frac{1}{w}$, and thus the probability that one iteration fails conditioned by $y \in V_w$ is $\leq (1 - (1/w))$. Since the procedure does $n \cdot w$ iterations, the probability over $y \in \{0,1\}^n$ and over the random bits used by the algorithm D, conditioned by $y \in V_w$, that y is not list-inverted is $\leq (1 - (1/w))^{n \cdot w} < e^{-n}$. Therefore the probability that y is not list-inverted is bounded by the probability that $y \notin V_w$ plus the above conditional probability of failure-to-list-invert. Thus, it is bounded by $\delta - e^{-n} + e^{-n} = \delta$.

Note that the algorithm D is using at each iteration the random strings $y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_\ell$ and there are $n \cdot w$ iterations. There is a way to fix these random strings used by D so that the circuit B that is obtained from D by using the fixed bits instead of random bits list-inverts a fraction of at least $(1 - \delta)$ of the strings $x \in \{0,1\}^n$. There are $n \cdot w \cdot (\ell - 1)$ fixed strings.

Assuming that the circuit C and ℓ are given, the permutation R can be described, using the previously-discussed procedure, from

- $2\delta \cdot N \cdot n$ bits that encode the δN elements that B fails to list-invert and the value of R at these points.

- The last L positions in the circular permutation R. This requires $L \cdot n$ bits.

- For each of the $(1-\delta)N$ strings x that are list-inverted by C, the rank of x in the generated LIST. This requires $(1-\delta) \cdot N \cdot (\log n + \log w + \log T)$ bits.

- The set of $n \cdot w \cdot (\ell - 1)$ fixed strings y and the value of R on $y, R(y), \ldots, R^{L-1}(y)$ for every fixed y. This requires $n \cdot w \cdot (\ell - 1) \cdot n + n \cdot w \cdot (\ell - 1) \cdot L \cdot n \leq n^2 w \ell L$ bits (for $\ell - 1 < L$).

The total number of bits needed for the description (given B) is bounded by

$$2\delta N n + L n + (1-\delta)N \log n + (1-\delta)N \log w + (1-\delta)N \log T + n^2 w \ell L.$$

Plugging the values of ℓ and w, we obtain that the description of R is bounded by $2\delta N n + L n + N \log n + (\log 6)N + N \log(1/\delta) + N \log \log(2/\gamma) + N \log(1/\gamma) + N \log T + 18n^2 \cdot L \cdot \frac{1}{\gamma} \cdot \left(\frac{1}{\delta}\right)^2 \left(\log \frac{2}{\gamma}\right)^2$. ∎

We want to estimate the number of permutations that are (γ, T)-good for some L-restricted circuit C. We state the result for a particular combination of parameters that will be of interest in our application. The extractor construction will involve the parameters $m \in \mathbf{N}$ and $\epsilon > 0$. We will have $\gamma = \epsilon/m$, $T = m^2 \cdot (1/\epsilon^2)$, and $L = m$.

Lemma 4.4 *Let $n \in \mathbf{N}, m \in \mathbf{N}, \epsilon > 0, \delta > 0$. Let $N = 2^n$. Consider $\gamma = \epsilon/m$ and $T = m^2 \cdot (1/\epsilon^2)$. Let $\ell = \lceil (3/\delta) \log(2/\gamma) \rceil$. Assume that $\delta = O(1)$ and $m^2 \cdot (1/\epsilon) = o(N/n^4)$. Let C be an m-restricted circuit, with inputs of length ℓn. Then the number of permutations $\overline{R}$ in PERM_ℓ that are (γ, T)-good for C is bounded by 2^h, where $h = 3\delta \cdot N \cdot n + 3N \log m + 3N \log(1/\epsilon)$.*

Proof Under the assumptions in the hypothesis, Lemma 4.3 implies that any permutation that is (γ, T)-good for C can be described with h bits. The conclusion follows immediately. ∎

5 Analysis of the construction of pseudo-random generators from one-way permutations

We recall the classic construction (Blum and Micali [BM84] and Yao [Yao82]) of a pseudo-random generator from a one-way permutation. The construction starts with a one-way permutation $\overline{R} : \{0,1\}^{\ell n} \rightarrow \{0,1\}^{\ell n}$. In the classical setting, we work under the assumption that no circuit of some bounded size inverts $\overline{R}(\overline{x})$ except for a small fraction of $\overline{x}$ in the domain of $\overline{R}$.

Step 1. We consider the predicate $b : \{0,1\}^{\ell n} \times \{0,1\}^{\ell n} \rightarrow \{0,1\}$ defined by $b(\overline{x}, r) = \overline{x} \cdot r$ (the inner product modulo 2). By the well-known Goldreich-Levin Theorem [GL89], $b(\overline{x}, r)$ is a hard-core predicate for $\overline{R}(\overline{x}) \odot r$, i.e., no circuit of an appropriate bounded size can calculate $b(\overline{x}, r)$ from $\overline{R}(\overline{x}) \odot r$ except with a probability very close to $1/2$. More precisely, it holds that if a probabilistic circuit C_1 on input $\overline{R}(\overline{x}) \odot r$ calculates $b(\overline{x}, r)$ with probability $1/2 + \epsilon$ (the probability is over $\overline{x}$, r, and the random bits used by C_1) then there is a circuit C_2 not much larger than C_1 which for a $3\epsilon/4$ fraction of $\overline{x}$ list-inverts $\overline{x}$. (In the classical setting this is in conflict with the above assumption, because one can check the elements from the list one by one till $\overline{x}$ is determined.) Lemma 5.3 proves this fact adapted to an information-theoretic context (actually, in our setting, the fact holds with stronger parameters).

Step 2. The function $H_{\overline{R}} : \{0,1\}^{2\ell n} \rightarrow \{0,1\}^{2\ell n + 1}$, given by $H_{\overline{R}}(\overline{x}, r) = \overline{R}(\overline{x}) \odot r \odot b(\overline{x}, r)$, can be shown to be a pseudo-random generator with extension 1. More precisely, it holds that if C_2 is a circuit that distinguishes $H_{\overline{R}}(\overline{x}, r)$ from $U_{2\ell n + 1}$ with bias ϵ, one can build a circuit C_3, not much larger than C_2, that on input $\overline{R}(\overline{x}) \odot r$ calculates $b(\overline{x}, r)$ correctly with probability at least $1/2 + \epsilon$. Lemma 5.2 proves this fact adapted to an information-theoretic context.

Step 3. We define $G_{\overline{R}}(\overline{x}, r)$ by the following algorithm.

Input: $\overline{R}$ a permutation of $\{0,1\}^{\ell n}$, $\overline{x} \in \{0,1\}^{\ell n}$, $r \in \{0,1\}^{\ell n}$.

For $i = 0$ to $m - 1$, $b_i = r \cdot (\overline{R}^i(\overline{x}))$.

Output $b_0 \odot b_1 \odot \ldots \odot b_{m-1}$.

It can be shown that under the given assumption, $G_{\overline{R}}$ is a pseudo-random generator. More precisely, it holds that if a circuit C_4 distinguishes $G_{\overline{R}}$ from U_m with bias ϵ, then there is a circuit C_3, not much larger than C_4, so that C_3 distinguishes $H_{\overline{R}}(\overline{x}, r)$ from $U_{2\ell n+1}$ with bias at least ϵ/m. Lemma 5.1 proves this fact adapted to an information-theoretic context.

We need to establish the properties of the above transformations (Steps 1, 2, and 3) in an information-theoretic context because they will be used for the construction of an extractor. In our setting $\overline{R} : \{0,1\}^{\ell n} \to \{0,1\}^{\ell n}$ is a random permutation and C_4 is a statistical test. We will show that there are some circuits $C_{1,1}, \ldots, C_{1,2^{m+1}-4}$ such that if $\overline{R}$ does not hit C_4 ϵ-correctly via G, then $\overline{R}$ is $(\epsilon/m, m^2/\epsilon^2)$-good for some $C_{1,i}$, and thus, by the results in the previous section, $\overline{R}$ has a short description. In our context, the size of the different circuits appearing in Steps 1, 2, and 3 will be considered to be unbounded. What matters is the number and the pattern of queries, i.e., the fact that C_3, C_2 and C_1 are restricted circuits. This is an informatic-theoretic feature. The following lemmas follow closely the standard proofs, only that, in addition, they analyze the pattern of queries made by the circuits involved.

Lemma 5.1 (Analysis of Step 3.) *For any circuit C_4 there are $2^{m-1}-1$ circuits $C_{3,1}, C_{3,2}, \ldots, C_{3,2^{m-1}-1}$ such that:*

(1) If $\overline{R}$ is a permutation with

$$|\mathrm{Prob}_{\overline{x},r}(G_{\overline{R}}(\overline{x}, r) \in C_4) - \mathrm{Prob}(U_m \in C_4)| > \epsilon,$$

(i.e., $\overline{R}$ does not hit C_4 ϵ-correctly via G), then there is $i \in \{1, \ldots, 2^{m-1} - 1\}$ such that

$$|\mathrm{Prob}(H_{\overline{R}}(U_{\ell n}, U'_{\ell n}) \in C^{\overline{R}}_{3,i}) - \mathrm{Prob}(U_{2\ell n+1} \in C^{\overline{R}}_{3,i})| > \frac{\epsilon}{m}.$$

(2) All the circuits $C_{3,i}$ are $(m - 2)$-restricted.

Proof For $k \in \{0, \ldots, m - 1\}$, we define the distributions

$$d_k = U_k \odot (U_{\ell n} \cdot U'_{\ell n}) \odot (\overline{R}(U_{\ell n}) \cdot U'_{\ell n}) \odot \ldots \odot (\overline{R}^{m-k-1}(U_{\ell n}) \cdot U'_{\ell n}),$$

where U_k, $U_{\ell n}$, and $U'_{\ell n}$ are distinct instances of the uniform distributions on $\{0,1\}^k$, $\{0,1\}^{\ell n}$, and $\{0,1\}^{\ell n}$, respectively. Suppose that a permutation $\overline{R}$ satisfies

$$\mathrm{Prob}_{\overline{x},r}(G_{\overline{R}}(\overline{x}, r) \in C_4) - \mathrm{Prob}(U_m \in C_4) > \epsilon. \tag{2}$$

In the new notation, the above reads $\mathrm{Prob}(d_0 \in C_4) - \mathrm{Prob}(d_{m-1} \in C_4) > \epsilon$. This implies that there is some $k \in \{0, \ldots, m - 2\}$ such that $\mathrm{Prob}(d_k \in C_4) - \mathrm{Prob}(d_{k+1} \in C_4) > \epsilon/m$. For $z_1 \in \{0,1\}^{\ell n}$, $z_2 \in \{0,1\}^{\ell n}$, $z_3 \in \{0,1\}$, we define

$$f(z_1 \odot z_2 \odot z_3) = z_3 \odot (z_1 \cdot z_2) \odot (\overline{R}(z_1) \cdot z_2) \odot \ldots \odot (\overline{R}^{m-k-2}(z_1) \cdot z_2).$$

Note that

$$d_k = U_k \odot f(R(U_{\ell n}) \odot U'_{\ell n} \odot (U_{\ell n} \cdot U'_{\ell n}))$$

11

and

$$d_{k+1} = U_k \odot f(U_{\ell n} \odot U'_{\ell n} \odot U_1).$$

We define the following circuit D that is able to distinguish $H_{\overline{R}}(U_{\ell n}, U'_{\ell n})$ from $U_{2\ell n+1}$. The input of D is a string $y \in \{0,1\}^{2\ell n+1}$, which we break into $y = y_1 \odot y_2 \odot y_3$, with y_1 and y_2 in $\{0,1\}^{\ell n}$, and $y_3 \in \{0,1\}$. The circuit D on input $y \in \{0,1\}^{2\ell n+1}$, chooses a k-bits long string u, calculates $f(y_1, y_2, y_3)$ using the oracle $\overline{R}$ and simulates C_4 on input $u \odot f(y_1, y_2, y_3)$. Note that the calculation of $f(y_1, y_2, y_3)$ requires at most the query of the strings $y_1, R(y_1), \ldots, R^{m-3}(y_1)$. Thus, D is an $(m-2)$-restricted circuit. Clearly, $\mathrm{Prob}_{u,y}(y \in D^{\overline{R}}) = \mathrm{Prob}(d_{k+1} \in C_4)$ and $\mathrm{Prob}_{u,\overline{x},r}(H_{\overline{R}}(\overline{x}, r) \in D^{\overline{R}}) = \mathrm{Prob}(d_k \in C_4)$. Therefore, $\mathrm{Prob}_{u,\overline{x},r}(H_{\overline{R}}(\overline{x}, r) \in D^{\overline{R}}) - \mathrm{Prob}_{u,y}(y \in D^{\overline{R}}) > \epsilon/m$. By fixing in all possible ways $k \in \{0, \ldots, m-2\}$ and then the k-bits long string u, we obtain $2^{m-1}-1$ circuits, denoted $C_{3,1}, \ldots, C_{3,2^{m-1}-1}$, that act like D except that the random bits are replaced by the fixed bits. The argument above shows that if R satisfies Equation (2), then there is one circuit $C_{3,i}$ in the above set of circuits such that $\mathrm{Prob}_{\overline{x},r}(H_{\overline{R}}(\overline{x}, r) \in C_{3,i}^{\overline{R}}) - \mathrm{Prob}_y(y \in C_{3,i}^{\overline{R}}) > \epsilon/m$. The circuits $C_{3,i}$ are $(m-2)$-restricted circuits. With a similar proof, one can see that if

$$\mathrm{Prob}(U_m \in C_4) - \mathrm{Prob}_{\overline{x},r}(G_{\overline{R}}(\overline{x}, r) \in C_4) > \epsilon, \tag{3}$$

then there is one circuit $C_{3,i}$ such that $\mathrm{Prob}_y(y \in C_{3,i}^{\overline{R}}) - \mathrm{Prob}_{\overline{x},r}(H_{\overline{R}}(\overline{x}, r) \in C_{3,i}^{\overline{R}}) > \epsilon/m$. ∎

Lemma 5.2 (Analysis of Step 2.) *Let C_3 be an oracle circuit that is L-restricted, for some parameter L. There are four oracle circuits $C_{2,1}, C_{2,2}, C_{2,3}, C_{2,4}$ such that*

(1) If a permutation $\overline{R}$ satisfies

$$|\mathrm{Prob}_{\overline{x},r}(H_{\overline{R}}(\overline{x} \odot r) \in C_3^{\overline{R}}) - \mathrm{Prob}(U_{2\ell n+1} \in C_3^{\overline{R}})| > \epsilon,$$

then there is $i \in \{1, 2, 3, 4\}$ such that

$$\mathrm{Prob}_{\overline{x},r}(C_{2,i}^R(\overline{R}(\overline{x}) \odot r) = b(\overline{x}, r)) > \frac{1}{2} + \epsilon.$$

(2) The four circuits are L-restricted.

Proof We define the oracle circuit B that on input $\overline{R}(\overline{x}) \odot r$ runs as follows. It chooses a random bit u and then it simulates the circuit $C_3^{\overline{R}}$ to determine if $\overline{R}(\overline{x}) \odot r \odot u$ belongs to C_3^R or not. If the answer is YES, the output is u, and if the answer is NO, the output is $1 - u$. We also define the circuit D in a similar way, with the only change that the YES/NO branches are permuted. Note that B and D are both circuits that are L-restricted.

Recall that $H_{\overline{R}}(\overline{x}, r) = \overline{R}(\overline{x}) \odot r \odot b(\overline{x}, r)$. Let us suppose that for some permutation $\overline{R}$, $\mathrm{Prob}_{\overline{x},r}(\overline{R}(\overline{x}) \odot r \odot b(\overline{x}, r)) \in C_3^{\overline{R}}) - \mathrm{Prob}(U_{2\ell n+1} \in C_3^{\overline{R}}) > \epsilon$. Note that $\mathrm{Prob}_{\overline{x},r}(\overline{R}(\overline{x}) \odot r \odot b(\overline{x}, r)) \in C_3^{\overline{R}}) - \mathrm{Prob}(U_{2\ell n+1} \in C_3^{\overline{R}}) = (\mathrm{Prob}_{\overline{x},r}(\overline{R}(\overline{x}) \odot r \odot b(\overline{x}, r)) \in C_3^{\overline{R}}) - \mathrm{Prob}_{U_1,\overline{x},r}(\overline{R}(\overline{x}) \odot r \odot U_1) \in C_3^{\overline{R}})) + (\mathrm{Prob}_{U_1,\overline{x},r}(\overline{R}(\overline{x}) \odot r \odot U_1) \in C_3^{\overline{R}}) - \mathrm{Prob}(U_{2\ell n+1} \in C_3^{\overline{R}}))$. The second term is equal to zero, because R is a permutation and, thus, $U_1 \odot \overline{R}(\overline{x}) \odot r$ is actually the uniform distribution on $\{0,1\}^{2\ell n+1}$. Therefore, $\mathrm{Prob}_{\overline{x},r}(\overline{R}(\overline{x}) \odot r \odot b(\overline{x}, r)) \in C_3^{\overline{R}}) - \mathrm{Prob}_{U_1,\overline{x},r}(\overline{R}(\overline{x}) \odot r \odot U_1) \in C_3^{\overline{R}}) > \epsilon$. According to Yao's lemma that connects predictors to distinguishers (for a proof see, for example, [Zim04, pp. 162]), it follows that $\mathrm{Prob}_{u,\overline{x},r}(B^R(\overline{R}(\overline{x}) \odot r) = b(\overline{x}, r)) > \frac{1}{2} + \epsilon$. Let B_0 (B_1) be the circuit that is obtained from B by fixing bit u to 0 (respectively, to 1). Then at least one of the events "$B_0(\overline{R}(\overline{x}) \odot R) = b(\overline{x}, r)$" or "$B_1(\overline{R}(\overline{x}) \odot R) = b(\overline{x}, r)$" has probability $> \frac{1}{2} + \epsilon$.

If $\mathrm{Prob}_z(z \in C_3^{\overline{R}}) - \mathrm{Prob}_{\overline{x},r}(\overline{R}(\overline{x}) \odot r \odot b(\overline{x}, r)) \in C_3^{\overline{R}}) > \epsilon$, then the same argument works for the circuit D, and we obtain two deterministic circuits D_0 and D_1. The four circuits B_0, B_1, D_0 and D_1 satisfy the requirements. ∎

Lemma 5.3 (Analysis of Step 1.) *Let C_2 be an oracle circuit that is L-restricted for some parameter L. Then there is a circuit C_1 such that*

(1) If $\overline{R}$ is a permutation such that $\text{Prob}_{\overline{x},r}(C_2^{\overline{R}}(\overline{R}(\overline{x}) \odot r) = b(\overline{x},r)) > \frac{1}{2} + \epsilon$, then for at least a fraction ϵ of $\overline{x} \in \{0,1\}^{\ell n}$, $C_1^{\overline{R}}$ on input $\overline{y} = \overline{R}(\overline{x})$ outputs a list of $1/\epsilon^2$ strings that contains $\overline{x}$ (i.e., $\overline{R}$ is $(\epsilon, 1/\epsilon^2)$-good for C_1).

(2) The circuit C_1 is L-restricted.

Proof Suppose permutation $\overline{R} : \{0,1\}^n \to \{0,1\}^{\ell n}$ satisfies $\text{Prob}_{\overline{x},r}(C_2^{\overline{R}}(\overline{R}(\overline{x}) \odot r) = b(\overline{x},r)) > (1/2) + \epsilon$. Then, by a standard averaging argument, for a fraction ϵ of $\overline{x}$ in $\{0,1\}^{\ell n}$, $\text{Prob}_r(C_2^R(\overline{R}(\overline{x}) \odot r) = b(\overline{x},r)) > (1/2) + (\epsilon/2)$. Consider such an $\overline{x}$ and let $\text{Had}(\overline{x})$ denote the encoding of $\overline{x}$ via the Hadamard error-correcting code (see [Tre04]). By the definition of the Hadamard code, $b(\overline{x},r)$ is just the r-th bit of $\text{Had}(\overline{x})$. Thus the string $u = C_2^R(\overline{R}(\overline{x}) \odot (0\ldots0)) \odot \ldots \odot C_2^R(\overline{R}(\overline{x}) \odot (1\ldots1))$ agrees with $\text{Had}(\overline{x})$ on at least a fraction $(1/2) + (\epsilon/2)$ of positions. Since the circuit C_2 is L-restricted, the string u can be calculated by querying only $\overline{y}, \overline{R}(\overline{y}), \ldots, \overline{R}^{L-1}(\overline{y})$, where $\overline{y} = \overline{R}(\overline{x})$. By brute force we can determine the list of all strings $\overline{z}$ so that $\text{Had}(\overline{z})$ agrees with u in at least $\frac{1}{2} + \frac{\epsilon}{2}$ positions. It is known (see, for example, [Zim04, pp. 218]) that there are at most $\frac{1}{4} \cdot \left(\frac{2}{\epsilon}\right)^2 = \left(\frac{1}{\epsilon}\right)^2$ such strings $\overline{z}$ and one of them is $\overline{x}$. ∎

By combining Lemma 5.1, Lemma 5.2, and Lemma 5.3, we obtain the following fact.

Lemma 5.4 *Let C_4 be a circuit. Then there are $2^{m+1} - 4$ circuits $C_{1,1}, \ldots, C_{1,2^{m+1}-4}$ such that*

(1) If $\overline{R}$ is a permutation with

$$|\text{Prob}_{\overline{x},r}(G_{\overline{R}}(\overline{x},r) \in C_4) - \text{Prob}(U_m \in C_4)| > \epsilon,$$

(i.e., $\overline{R}$ does not hit C_4 ϵ-correctly via G), then there is some circuit $C_{1,i}$ such that for at least a fraction $\frac{\epsilon}{m}$ of $\overline{x}$, $C_{1,i}^{\overline{R}}$ on input $\overline{R}(\overline{x})$ outputs a list of $m^2 \cdot \left(\frac{1}{\epsilon}\right)^2$ strings that contains $\overline{x}$ (i.e., $\overline{R}$ is $(\epsilon/m, m^2/\epsilon^2)$-good for $C_{1,i}$).

(2) All the circuits $C_{1,i}$ are $(m-2)$-restricted.

6 An extractor from a crypto pseudo-random generator

We first build a special type of extractor in which the weakly-random string is the truth-table of a permutation in PERM.

The following parameters will be used throughout this section. Let $\epsilon > 0, \delta > 0$, and $n, m \in \mathbf{N}$ be parameters. Let $N = 2^n$. Let $\ell = \lceil (3/\delta) \log(2m \cdot (1/\epsilon)) \rceil$. We consider the set of permutations PERM_ℓ. We assume that $\delta = O(1)$ and $m^2 \cdot (1/\epsilon) = o(N/n^4)$.

Let $G : \text{PERM}_\ell \times (\{0,1\}^{\ell n} \times \{0,1\}^{\ell n}) \to \{0,1\}^m$ be the function defined by the following algorithm (the same as the algorithm for $G_{\overline{R}}$ from the previous section).

Parameters: $\ell \in \mathbf{N}, m \in \mathbf{N}$.
Input: $\overline{R} \in \text{PERM}_\ell, (\overline{x}, r) \in \{0,1\}^{\ell n} \times \{0,1\}^{\ell n}$.
For $i = 0$ to $m-1$, $b_i = r \cdot \overline{R}^i(\overline{x})$.
Output $b_0 \odot b_1 \odot \ldots \odot b_{m-1}$.

The following lemma, in view of Lemma 2.2, shows that G is an extractor for the special case of weakly-random strings that are truth-tables of permutations in PERM_ℓ.

Lemma 6.1 *Let C_4 be a test for strings of length m (i.e., $C_4 \subseteq \{0,1\}^m$). Let $\mathrm{GOOD}(C_4) = \{\overline{R} \in \mathrm{PERM}_\ell \mid \overline{R}$ does not hit C_4 ϵ-correctly via $G\}$. Then $\|\mathrm{GOOD}(C_4)\| < 2^{m+h+1}$, where $h = 3\delta Nn + 3N\log m + 3N\log(1/\epsilon)$.*

Proof Let $C_{1,1}, \ldots, C_{1,2^{m+1}-4}$ be the $2^{m+1} - 4$ circuits implied by Lemma 5.4 to exist (corresponding to the test C_4). Let $\overline{R}$ be in $\mathrm{GOOD}(C_4)$. Then Lemma 5.4 shows that there is a circuit $C_{1,i}$ from the above list having the following property: For at least a fraction $\gamma = \epsilon/m$ of strings $\overline{x} \in \{0,1\}^{\ell n}$, $C_{1,i}^{\overline{R}}$ on input $\overline{R}(\overline{x})$ returns a list having $T = m^2 \cdot (1/\epsilon^2)$ strings, one of which is $\overline{x}$. Thus, $\overline{R}$ is (γ, T)-good for $C_{1,i}$ (recall Definition 4.2). It follows that the set of permutations $\overline{R} \in \mathrm{PERM}_\ell$ that do not hit C_4 ϵ-correctly via G is included in $\bigcup_1^{2^{m+1}-4} \{\overline{R} \in \mathrm{PERM}_\ell \mid \overline{R}$ is (γ, T)-good for $C_{1,i}\}$. Lemma 4.4 shows, that, for each $i \in \{1, \ldots, 2^{m+1} - 4\}$, $\|\{\overline{R} \in \mathrm{PERM}_\ell \mid \overline{R}$ is (γ, T)-good for $C_{1,i}\}\| \leq 2^h$, where $h = 3\delta \cdot N \cdot n + 3N\log m + 3N\log(1/\epsilon)$. The conclusion follows. ∎

In order to obtain a standard extractor (rather than the special type given by Lemma 6.1), the only thing that remains to be done is to transform a random binary string X into a permutation $R \in \mathrm{CIRC}$, which determines $\overline{R} \in \mathrm{PERM}_\ell$ that is used in the function G given above.

Note that a permutation $R \in \mathrm{CIRC}$ is specified by $(R(1), R^2(1), \ldots, R^{N-1}(1))$, which is an arbitrary permutation of the set $\{2, 3, \ldots, N\}$. Consequently, we need to generate permutations of the set $\{1, 2, \ldots, N-1\}$ (which can be viewed as permutations of $\{2, 3, \ldots, N\}$ in the obvious way). We can use the standard procedure that transforms a function mapping $[N-1]$ to $[N-1]$ into a permutation of the same type. To avoid some minor truncation nuisances, we actually use a function $X : [N] \to [N]$.

Input: $X : [N] \to [N]$.
    for $i = 1$ to $N - 1$, $R(i) = i$ (initially R is the identity permutation).
    Loop 2:
    for $i = 1$ to $N - 1$
        $Y(i) = 1 + (X(i) \mod i)$.
    Loop 3:
    for $i = 1$ to $N - 1$
        Swap $R(i)$ with $R(Y(i))$.
Output: permutation $R : [N-1] \to [N-1]$.

We want to estimate the number of functions $X : [N] \to [N]$ that map via the above procedure to a given permutation $R : [N-1] \to [N-1]$. We call a sequence $(Y(1), \ldots, Y(N))$ a **-sequence* if, for all i, $Y(i) \in \{1, \ldots, i\}$. Observe that, using Loop 3, a *-sequence $(Y(1), \ldots, Y(N-1))$ defines a unique permutation $(R(1), \ldots, R(N-1))$, and thus it is enough to estimate the maximum number of functions $X : [N] \to [N]$ that map via Loop 2 in the above procedure in a given *-sequence $(Y(1), \ldots, Y(N-1))$ (the maximum is taken over all *-sequences of length $N - 1$). We denote this number by $A(N)$. A (rough) upper bound can be established as follows.

$$
\begin{aligned}
A(N) &\leq \left\lceil \frac{N}{1} \right\rceil \cdot \left\lceil \frac{N}{2} \right\rceil \cdot \ldots \cdot \left\lceil \frac{N}{N-1} \right\rceil \\
&\leq \left(\frac{N}{1} + 1 \right) \cdot \left(\frac{N}{2} + 1 \right) \cdot \ldots \cdot \left(\frac{N}{N-1} + 1 \right) \\
&= \frac{(N+1)(N+2) \cdot \ldots \cdot (2N-1)}{1 \cdot 2 \cdot \ldots \cdot (N-1)} \\
&= \binom{2N-1}{N-1} \leq 2^{2N}.
\end{aligned}
$$

14

We can now present the (standard) extractor. We choose the parameters as follows. Fix $n \in \mathbf{N}$ and let $N = 2^n$ and $\overline{N} = n \cdot 2^n$. Let $\lambda \in (0,1)$ be a constant. Let $\alpha > 0, \beta > 0$ be constants such that $\alpha < \lambda/3, \beta < (\lambda - 3\alpha)/4$. Let $\epsilon \geq N^{-\beta}$ and $m \leq N^\alpha$. Take $\delta = (\lambda - 4\beta - 3\alpha)/4$ and $\ell = \lceil (3/\delta) \log(2m \cdot (1/\epsilon)) \rceil$. The weakly-random string X has length $\overline{N}$ and is viewed as the truth-table of a function mapping $[N]$ to $[N]$. The seed is of the form $y = (\overline{x}, r) \in \{0,1\}^{\ell n} \times \{0,1\}^{\ell n}$. We first transform X into a permutation $\overline{R}(X) \in \mathrm{PERM}_\ell$ using the above algorithm and then taking the ℓ-product. We define the extractor $E : \{0,1\}^{\overline{N}} \times \{0,1\}^{2\ell n} \to \{0,1\}^m$ by $E(X, (\overline{x}, r)) = G(\overline{R}(X), (\overline{x}, r))$. More explicitly, the extractor is defined by the following procedure.

Parameters: $n \in \mathbf{N}, \overline{N} \in \mathbf{N}, \lambda > 0, \epsilon > 0, \ell \in \mathbf{N}, m \in \mathbf{N}$, satisfying the above requirements.

Inputs: The weakly-random string $X \in \{0,1\}^{\overline{N}}$, viewed as the truth-table of a function $X : [N] \to [N]$; the seed $y = (\overline{x}, r) \in \{0,1\}^{\ell n} \times \{0,1\}^{\ell n}$.

Step 1. Transform X into a permutation $\overline{R}_X \in \mathrm{PERM}_\ell$. The transformation is performed by the above procedure which yields a permutation $R \in \mathrm{CIRC}$, and, next, $\overline{R}_X$ is the ℓ-direct product of R.

Step 2. For $i = 0$ to $m - 1$, $b_i = r \cdot \overline{R}_X^i(\overline{x})$.
Output $b_0 \odot b_1 \odot \ldots \odot b_{m-1}$, which is denoted $E(X, y)$.

We have defined a function $E : \{0,1\}^{\overline{N}} \times \{0,1\}^{2\ell n} \to \{0,1\}^m$. Note that the seed length $2\ell n$ is $O(\log^2 \overline{N})$ and the output length m is $\overline{N}^\alpha$, for an arbitrary $\alpha < \lambda/3$.

Theorem 6.2 *The function E is a $(\lambda \overline{N}, 2\epsilon)$-extractor.*

Proof Let C_4 be a subset of $\{0,1\}^m$. Taking into account Lemma 2.2, it is enough to show that the number of strings $X \in \{0,1\}^{\overline{N}}$ that do not hit C_4 ϵ-correctly via E is at most $2^{\lambda \overline{N} - \log(1/\epsilon)}$. Let $X \in \{0,1\}^{\overline{N}}$ be a string that does not hit C_4 ϵ-correctly via E. By the definition of E, it follows that $\overline{R}_X$ does not hit C_4 ϵ-correctly via G. By Lemma 6.1, there are at most 2^{m+h+1} permutations $\overline{R} \in \mathrm{PERM}_\ell$ that do not hit C_4 ϵ-correctly via G, where $h = 3\delta N n + 3N \log m + 3N \log(1/\epsilon)$. Since the number of functions $X : [N] \to [N]$ that map into a given permutation $\overline{R} \in \mathrm{PERM}_\ell$ is at most $A(N) < 2^{2N}$, it follows that

$$\|\{X \in \{0,1\}^{\overline{N}} \mid X \text{ does not hit } C_4 \text{ } \epsilon\text{-correctly}\}\| < 2^{2N} \cdot 2^{m+h+1} < 2^{\lambda \overline{N} - \log(1/\epsilon)},$$

where the last inequality follows from the choice of parameters. ∎

7 A bitwise locally-computable extractor

We present a bitwise locally-computable extractor: Each bit of the output string can be calculated separately in $O(\log^2 \overline{N})$, where $\overline{N}$ is the length of the weakly-random string. The proof uses the same plan as for the extractor in Section 6, except that the weakly-random string X is viewed as the truth-table of an arbitrary function (not necessarily a permutation) and the "consecutive"' values that are used in the extractor are $\overline{X}(\overline{x}), \overline{X}(\overline{x} + 1), \ldots, \overline{X}(\overline{x} + m - 1)$ (instead of $\overline{R}(\overline{x}), \overline{R}^2(\overline{x}), \ldots, \overline{R}^{m-1}(\overline{x})$ used in Section 6).

The parameter $n \in \mathbf{N}$ will be considered fixed throughout this section. We denote $N = 2^n$ and $\overline{N} = n \cdot N$. The parameter $m \in \mathbf{N}$ will be specified later (it will be a subunitary power of N). For

two binary strings x and r of the same length, $b(x, r)$ denotes the inner product of x and r viewed as vectors over the field GF(2).

The weakly-random string X has length $\overline{N}$, and is viewed as the truth-table of a function $X :$ $\{0, 1\}^n \rightarrow \{0, 1\}^n$. For some $\ell \in \mathbf{N}$ that will be specified later we define $\overline{X} : \{0, 1\}^{\ell n} \rightarrow \{0, 1\}^{\ell n}$ by $\overline{X}(x_1 \odot \ldots \odot x_\ell) = X(x_1) \odot \ldots \odot X(x_\ell)$, i.e., $\overline{X}$ is the ℓ-direct product of X. We also denote $\overline{x} = x_1 \odot \ldots \odot x_\ell$. The seed of the extractor will be $(\overline{x}, r) \in \{0, 1\}^{\ell n} \times \{0, 1\}^{\ell n}$. We define $\overline{x} + 1 = (x_1 + 1) \odot \ldots \odot (x_\ell + 1)$ (where the addition is done modulo 2^n) and inductively, for any $k \in \mathbf{N}$, $\overline{x} + k + 1 = (\overline{x} + k) + 1$. The extractor is defined by

$$E(X, (\overline{x}, r)) = b(\overline{X}(\overline{x}), r) \odot b(\overline{X}(\overline{x} + 1), r) \odot \ldots \odot b(\overline{X}(\overline{x} + m - 1), r). \tag{4}$$

A set $D \subseteq \{0, 1\}^m$ is called a *test*. We say that X hits a test D ϵ-correctly via E if $|\text{Prob}_{\overline{x}, r}(E(X, (\overline{x}, r)) \in D) - \frac{\|D\|}{\|\{0,1\}^m\|}| \leq \epsilon$. We want to show that the number of functions X that do not hit D ϵ-correctly via E is small and then use Lemma 2.2. To this aim we investigate the properties of a function X that does not hit a test $D \subseteq \{0, 1\}^m$ ϵ-correctly via E.

Lemma 7.1 *Let $D \subseteq \{0, 1\}^m$ be a fixed set. Then there are $2^{m+2} - 4$ circuits $C_1, \ldots C_{2^{m+2}-4}$ such that if X does not hit D ϵ-correctly, then there is some circuit C_i, $i \in \{1, \ldots, 2^{m+2} - 4\}$, such that*

$$\text{Prob}_{\overline{x}, r}(C_i \text{ on input } b(\overline{X}(\overline{x} - m + 1), r) \odot \ldots \odot b(\overline{X}(\overline{x} - 1), r) \text{ outputs } b(\overline{X}(\overline{x}), r)) \geq 1/2 + \epsilon/m.$$

Proof The proof is similar to the proofs of lemmas 5.1 and 5.2. Let $X : \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a function that does not hit D ϵ-correctly via E. This means that $|\text{Prob}_{\overline{x}, r}(E(X, (\overline{x}, r)) \in D) - \text{Prob}(U_m \in D)| > \epsilon$. Let us first suppose that

$$\text{Prob}_{\overline{x}, r}(E(X, (\overline{x}, r)) \in D) - \text{Prob}(U_m \in D) > \epsilon. \tag{5}$$

For each $k \in \{0, \ldots, m - 1\}$, we define the hybrid distribution d_k given by

$$d_k = b(\overline{X}(\overline{x}), r) \odot b(\overline{X}(\overline{x} + 1), r) \odot \ldots \odot b(\overline{X}(\overline{x} + m - k - 1), r) \odot U_k.$$

and

$$d_m = U_m.$$

Equation (5) states that $\text{Prob}(d_0 \in D) - \text{Prob}(d_m \in D) > \epsilon$. Using the standard argument, it follows that there exists $k \in \{0, \ldots, m - 1\}$ such that $\text{Prob}(d_k \in D) - \text{Prob}(d_{k+1} \in D) > \epsilon/m$. We build a probabilistic circuit C that on input $b(\overline{X}(\overline{x}), r) \odot b(\overline{X}(\overline{x} + 1), r) \odot \ldots \odot b(\overline{X}(\overline{x} + m - k - 2), r)$ attempts to calculate $b(\overline{X}(\overline{x} + m - k - 1), r)$.

Circuit C.

Input: $v_0 \odot v_1 \odot \ldots \odot v_{m-k-2}$, each $v_i \in \{0, 1\}$. (In case $k = m - 1$, there is no input.)

Choose randomly $u \in \{0, 1\}, t \in \{0, 1\}^k$.

If $v_0 \odot v_1 \odot \ldots \odot v_{m-k-2} \odot u \odot t \in D$, return u.
Else return $1 - u$.

By Yao's lemma on predictors versus distinguishers, it holds that

$\text{Prob}_{\overline{x}, r}(C \text{ on input } b(\overline{X}(\overline{x}), r) \odot \ldots \odot b(\overline{X}(\overline{x} + m - k - 2), r) \text{ outputs } b(\overline{X}(\overline{x} + m - k - 1), r)) \geq 1/2 + \epsilon/m,$

where the probability is taken over $\overline{x} \in \{0,1\}^{\ell n}, r \in \{0,1\}^{\ell n}$ and the random bits used by C. The procedure C uses $k+1$ random bits (for u and t), and $k \in \{0, \ldots, m-1\}$. By considering all possibilities for k and by fixing the $(k+1)$ bits in all possible ways, we obtain $2^1 + \ldots + 2^m = 2^{m+1} - 2$ circuits $C_1, \ldots, C_{2^{m+1}-2}$ with the desired property.

In the alternative (to Equation (5)) case

$$\text{Prob}(U_m \in D) - \text{Prob}(E(X, (\overline{x}, r)) \in D) > \epsilon,$$

we obtain in a similar way another set of $2^{m+1} - 2$ circuits. ∎

The next lemma is an analogue of Lemma 5.3. It states that if there exists a circuit C with the property indicated in Lemma 7.1, then, from $\overline{X}(\overline{x} - m + 1), \ldots, \overline{X}(\overline{x} - 1)$, one can compute, in a weak but non-trivial way, $\overline{X}(\overline{x})$.

Lemma 7.2 *Let C be a circuit. Then there is a circuit B such that the following holds. Suppose $\overline{X} : \{0,1\}^{\ell n} \to \{0,1\}^{\ell n}$ is a function such that*

$$\text{Prob}_{\overline{x}, r}(C(b(\overline{X}(\overline{x} - m + 1), r) \odot \ldots \odot b(\overline{X}(\overline{x} - 1), r)) = b(\overline{X}(\overline{x}), r)) \geq (1/2) + (\epsilon/m).$$

Then, for at least a fraction ϵ/m of $\overline{x}$ in $\{0,1\}^{\ell n}$, B on input $\overline{X}(\overline{x} - m + 1) \odot \ldots \odot \overline{X}(\overline{x} - 1)$ outputs a list of $m^2 \cdot (1/\epsilon^2)$ elements, one of which is $\overline{X}(\overline{x})$.

Proof The proof is similar to the proof of Lemma 5.3. Let C and $\overline{X}$ be as in the hypothesis. By an averaging argument, it follows that for a fraction (ϵ/m) of $\overline{x}$ in $\{0,1\}^{\ell n}$,

$$\text{Prob}_r(C(b(\overline{X}(\overline{x} - m + 1), r) \odot \ldots \odot b(\overline{X}(\overline{x} - 1), r) = b(\overline{X}(\overline{x}), r)) \geq (1/2) + \epsilon/(2m). \tag{6}$$

Let $\text{Had}(x)$ denote the encoding of a string x via the Hadamard error-correcting code. By the definition of the Hadamard code, $b(x, r)$ is just the r-th bit of $\text{Had}(x)$. Consider the binary string $u(\overline{x}) \in \{0,1\}^{2^{\ell n}}$ whose r-th bit is $C(b(\overline{X}(\overline{x} - m + 1), r) \odot \ldots \odot b(\overline{X}(\overline{x} - 1), r)$ (here $r \in \{0, \ldots, 2^{\ell n} - 1\}$ is written in base 2 on ℓn bits for the sake of the definition of b). Clearly, the string $u(\overline{x})$ can be calculated from $\overline{X}(\overline{x} - m + 1), \ldots, \overline{X}(\overline{x} - 1)$. The equation (6) implies that, for a fraction (ϵ/m) of $\overline{x} \in \{0,1\}^{\ell n}$, $u(\overline{x})$ agrees with $\text{Had}(\overline{X}(\overline{x}))$ on at least $1/2 + \epsilon/(2m)$ positions. By brute force, we can determine all the strings z so that $\text{Had}(z)$ agrees with $u(\overline{x})$ in at least $\frac{1}{2} + \epsilon/(2m)$ positions. It is known that there are at most $\frac{1}{4} \cdot \left(\frac{2m}{\epsilon}\right)^2 = \left(\frac{m}{\epsilon}\right)^2$ such strings z and, by the above discussion, one of them is $\overline{X}(\overline{x})$. ∎

The key property of the circuit B in the above lemma is captured in the following definition (which is analogous to Definition 4.2).

Definition 7.3 *Let B be a circuit. A function $\overline{X} : \{0,1\}^{\ell n} \to \{0,1\}^{\ell n}$ is (γ, T)-good for B if for at least a γ fraction of $\overline{x} \in \{0,1\}^{\ell n}$, B on input $\overline{X}(\overline{x} - m + 1) \odot \ldots \odot \overline{X}(\overline{x} - 1)$ outputs a T- list of strings, one of which is $\overline{X}(\overline{x})$.*

We choose the parameters in the same way as in Section 4. The parameters ϵ and m will be specified later. We take $\delta > 0$, $\gamma = \epsilon/m$, $T = m^2/\epsilon^2$, $\ell = \lceil(3/\delta)\log(2/\gamma)\rceil$ and $w = \lceil 6 \cdot (1/\delta) \cdot \log(2/\gamma) \cdot (1/\gamma)\rceil$.

The next two lemmas are the analogues of Lemma 4.3. The first lemma shows the amplification effect obtained by taking the ℓ-direct product.

Lemma 7.4 *The parameters are as specified above. Let B be a circuit. Then there is an oracle circuit A such that:*

(1) If $\overline{X}$ is (γ, T)-good for B, then, for a fraction $(1 - \delta)$ of x in $\{0,1\}^n$, the circuit A, on input x and $X(x - m + 1) \odot \ldots \odot X(x - 1)$ and with access to oracle X restricted as shown in (2), outputs a list containing $n \cdot w \cdot T$ elements, one of which is $X(x)$.

(2) The oracle circuit A queries a set of $n \cdot w \cdot (\ell - 1) \cdot (m - 1)$ strings that do not depend on the input.

Proof The proof is very similar to the first part of the proof of Lemma 4.3. Let GOOD be the set of strings $\overline{x}$ in $\{0,1\}^{\ell n}$ such that the circuit B, on input $\overline{X}(\overline{x} - m + 1) \odot \ldots \odot \overline{X}(\overline{x} - 1)$, calculates a T-list that contains $\overline{X}(\overline{x})$. By hypothesis, $\|\text{GOOD}\| \geq \gamma \cdot 2^{\ell n}$. We consider the following algorithm A' that can query the oracle $X : \{0,1\}^n \to \{0,1\}^n$ in several random positions.

Input: $x \in \{0,1\}^n$, and $X(x - m + 1), \ldots, X(x - 1) \in (\{0,1\}^n)^{m-1}$. The algorithm can pose random queries to the oracle $X : \{0,1\}^n \to \{0,1\}^n$. The goal is to calculate a list of strings that contains $X(x)$.

LIST $= \emptyset$.

Repeat the following $n \cdot w$ times.

Pick random $i \in \{1, \ldots, \ell\}$.

Pick $\ell - 1$ random strings in $\{0,1\}^n$ denoted $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_\ell$.

By querying the oracle X, find, for each x_j, the strings $X(x_j - m + 1), X(x_j - m + 2), \ldots, X(x_j - 1)$.

Let $\overline{x} = (x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_\ell)$. Build the string $\overline{X}(\overline{x} - m + 1) \odot \overline{X}(\overline{x} - m + 2) \odot \ldots \odot \overline{X}(\overline{x} - 1)$.

Run the circuit B on input $\overline{X}(\overline{x} - m + 1) \odot \overline{X}(\overline{x} - m + 2) \odot \ldots \odot \overline{X}(\overline{x} - 1)$.

The circuit B returns a T-list of ℓ-tuples in $(\{0,1\}^n)^\ell$.

(Note: In case of success, one of these ℓ-tuples is

$\overline{X}(\overline{x}) = X(x_1), \ldots, X(x_{i-1}), X(x), X(x_{i+1}) \ldots, X(x_\ell))$

Add to LIST the i-th component of every ℓ-tuple in the list produced by B.

End Repeat

We say that the above algorithm is successful on input x if, at the conclusion of the algorithm, LIST contains $X(x)$. We estimate the success probability of the above circuit on input x. Let $N(x)$ be the multiset of ℓ-tuples having x as one component where the multiplicity of a tuple is the number of occurrences of x in the tuple. On input x, at each iteration, the algorithm chooses uniformly at random $\overline{x}$ in $N(x)$. The algorithm succeeds at that iteration if and only if $\overline{x} \in \text{GOOD}$. By following the same arguments and the same calculations as in Lemma 4.3, we conclude that the probability that algorithm A' succeeds on x is at least $(1 - \delta)$, where the probability is taken over x and the random strings used by A'. Note that the algorithm A' is using at each iteration the random strings $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_\ell$, and there are $n \cdot w$ iterations. For each such random string x_j, A' needs the $(m - 1)$ values $X(x_j - m + 1), X(x_j - m + 2), \ldots, X(x_j - 1)$. There is a way to fix the above random strings so that the circuit A, which results from A' by using the fixed strings instead of the random strings, succeeds on at least a $(1 - \delta)$ fraction of the strings $x \in \{0,1\}^n$. Therefore, the circuit A has the desired properties. ∎

Lemma 7.5 *The parameters are as specified above. Let A be an oracle circuit and $X : \{0,1\}^n \to \{0,1\}^n$ be a function such that A and X satisfy the conditions (1) and (2) in Lemma 7.4. More precisely, we assume that:*

(1) For a fraction $(1 - \delta)$ of x in $\{0,1\}^n$, the circuit A, on input x and $X(x - m + 1) \odot \ldots \odot X(x - 1)$ and with access to oracle $\overline{X}$ restricted as shown in (2), outputs a list containing $n \cdot w \cdot T$ elements, one of which is $X(x)$.

(2) The oracle circuit A queries a set of $n \cdot w \cdot (\ell - 1) \cdot (m - 1)$ strings that do not depend on the input.

Then, given A, X can be described using a number of bits bounded by $2\delta N n + mn + N \log n + (\log 6)N + N \log(1/\delta) + N \log \log(2/\gamma) + N \log(1/\gamma) + N \log T + 36n^2 \cdot m \cdot \frac{1}{\gamma} \cdot \left(\frac{1}{\delta}\right)^2 \left(\log \frac{2}{\gamma}\right)^2$.

Proof The oracle circuit A allows a short description of the strings $X(x)$ for the fraction of $(1 - \delta)$ of the strings $x \in \{0,1\}^n$ given in assumption (1). Namely, such a string $X(x)$ is completely determined by the circuit A, by the value of X for the fixed set of queries given in assumption (2), by the previous

$m - 1$ values $X(x - m + 1), \ldots, X(x - 1)$, and by the rank of $X(x)$ in the list returned by A on input $x, X(x - m + 1), \ldots, X(x - 1)$. Thus, the truth-table of the function $X : \{0,1\}^n \to \{0,1\}^n$ can be described (given the circuit A) using the following information.

- $2\delta \cdot N \cdot n$ bits that encode the set of δN elements on which A fails and the value of X at these points.

- The "first" $m - 1$ values $X(0), \ldots, X(m - 1)$. This information requires $(m - 1) \cdot n$ bits. (Here, $X(i)$ represents the value of X at the i-th string in $\{0,1\}^n$, lexicographically ordered.)

- For each of the $(1 - \delta)N$ strings y on which A succeeds, the rank of $X(x)$ in the list returned by A. This requires $(1 - \delta) \cdot N \cdot (\log n + \log w + \log T)$ bits.

- The set of $n \cdot w \cdot (\ell - 1) \cdot (m - 1)$ fixed strings that are queried by A and the value of X at these strings. This information requires $2n^2 \cdot w \cdot (\ell - 1) \cdot (m - 1)$ bits.

The total number of bits needed for the description of X (given A) is bounded by

$$2\delta Nn + m \cdot n + (1 - \delta)N \log n + (1 - \delta)N \log w + (1 - \delta)N \log T + 2n^2 \cdot w \cdot (\ell - 1) \cdot (m - 1).$$

Keeping into account that $\ell = \lceil (3/\delta) \log(2/\gamma) \rceil$ and $w = \lceil 6 \cdot (1/\delta) \cdot \log(2/\gamma) \cdot (1/\gamma) \rceil$, the conclusion follows. ∎

We make the final choice of parameters. Let $n \in \mathbf{N}$ and the constant $\lambda \in (0,1)$. Recall that $N = 2^n$ and $\overline{N} = n \cdot N$. We take the constants $\alpha < \lambda/3, \beta < (\lambda - 3\alpha)/4$ and $\delta = (\lambda - 3\alpha - 4\beta)/4$. We also take the output length $m \leq N^\alpha$ and the extractor bias $\epsilon \geq N^{-\beta}$. Note that $\ell = \lceil (3/\delta) \log(2m/\epsilon) \rceil = O(n)$.

Theorem 7.6 *Assume that the parameters $\overline{N}, \lambda, m, \ell$ and ϵ satisfy the above requirements. Then the function $E : \{0,1\}^{\overline{N}} \times \{0,1\}^{2\ell n} \to \{0,1\}^m$, given in Equation 4, is a $(\lambda\overline{N}, 2\epsilon)$-extractor.*

Proof Assume $X \in \{0,1\}^{\overline{N}}$ does not hit a test $D \subseteq \{0,1\}^m$ ϵ-correctly via E. Then X can be described by one of the circuits $C_1, \ldots, C_{2^{m+2}-4}$, given by Lemma 7.1, and, according to Lemma 7.5, by a string of length h, where $h \leq 2\delta Nn + mn + N \log n + (\log 6)N + N \log(1/\delta) + N \log\log(2/\gamma) + N \log(1/\gamma) + N \log T + 36n^2 \cdot m \cdot \frac{1}{\gamma} \cdot \left(\frac{1}{\delta}\right)^2 \left(\log \frac{2}{\gamma}\right)^2$. Thus, the number of strings X that do not hit D ϵ-correctly via E is bounded by 2^{m+2+h}. For our choice of parameters, it holds that $m + 2 + h \leq \lambda\overline{N} - \log(1/\epsilon)$. Therefore, by Lemma 2.2, E is a $(\lambda\overline{N}, 2\epsilon)$-extractor. ∎

The construction scheme of the last extractor (given in Equation (4)) allows some flexibility in the choice of parameters and, in particular, we can obtain an extractor with seed length logarithmic in the length of the weakly random string. Namely, we can consider the weakly random string X to be the truth-table of a function of type $X : \{0,1\}^n \to \{0,1\}^{N_1}$, where $N_1 >> n$. We use the same value of ℓ, and we take the ℓ-direct product of X and obtain $\overline{X} : \{0,1\}^{\ell n} \to \{0,1\}^{\ell N_1}$. Clearly, $|\overline{X}(\overline{x})| = \ell N_1$. To get a short seed we need to replace the Hadamard code (recall that the function $b(x,r)$ gives the r-th bit of $\mathrm{Had}(x)$) by an error-correcting code with a good list decoding property that has a better rate. For example the code given in [GHSZ02], which we denote Code, is of the type Code $: \{0,1\}^{\tilde{n}} \to \{0,1\}^{\overline{n}}$, with $\overline{n} = O(\tilde{n} \cdot (1/\epsilon)^4)$, is computable in polynomial time, and it has the property that any ball of radius $(1/2) + \epsilon$ has at most $O((1/\epsilon)^2)$ codewords. Similarly to function b, we define the function $c(\overline{x}, r) =$ the r-th bit of $\mathrm{Code}(\overline{x})$, for $\overline{x} \in \{0,1\}^{\ell n}$ and any binary string r with length $|r| = \log(\mathrm{Code}(\overline{x})) = \log(\ell \cdot N_1 \cdot (1/\epsilon)^4) + O(1)$. We define the extractor E' by

$$E'(X, (\overline{x}, r)) = c(\overline{X}(\overline{x}), r) \odot c(\overline{X}(\overline{x} + 1), r) \odot \ldots \odot c(\overline{X}(\overline{x} + m - 1), r). \tag{7}$$

The analysis is very similar to that done for the previous extractor given in Equation (4). For example, if we assume that $\epsilon \leq 2^{(1/4)n}$, take $N_1 = 2^{n^2}$ and $m = 2^{(1/3)n}$, and we denote the length of X by $\overline{N}$ (i.e.,

$\overline{N} = 2^{n^2+n}$), we obtain a quite simple extractor that has seed length $O(\log(\overline{N}))$, is capable to extract from sources with min-entropy $\lambda\overline{N}$, for arbitrary constant $\lambda > 0$, and has output length $\approx 2^{(1/3)\sqrt{\log(\overline{N})}}$. This extractor has a good seed length, however the output length is much smaller than the min-entropy of the source.

8   Acknowledgments

References

[BM84]   Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13(4):850–864, November 1984.

[DM04]   Stefan Dziembowski and Ueli Maurer. Optimal randomizer efficiency in the bounded-storage model. *Journal of Cryptology*, 17(1):5–26, January 2004. (Conference version appeared in Proc. of STOC'02.).

[GHSZ02] V. Guruswamy, J. Håstad, M. Sudan, and D. Zuckerman. Combinatorial bounds for list decoding. *IEEE Transactions on Information Theory*, 48(5):1021–1035, May 2002.

[GL89]   O. Goldreich and L. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the 21st ACM Symposium on Theory of Computing*, pages 25–32, 1989.

[GT00]   R. Gennaro and L. Trevisan. Lower bounds on the efficiency of generic cryptographic constructions. In *Proceedings of the 41st IEEE Symposium on Foundations of Computer Science*, 2000.

[HILL99]   J. Håstad, R. Impagliazzo, L. Levin, and M. Luby. Construction of a pseudo-random generator from any one-way function. *SIAM Journal on Computing*, 28(4), 1999.

[HR03]   T. Hartman and R. Raz. On the distribution of the number of roots of polynomials and explicit logspace extractors. *Random Structures & Algorithms*, 23(3):235–263, Oct. 2003.

[Imp96]   R. Impagliazzo. Very strong one-way functions and pseudo-random generators exist relative to a random oracle. (manuscript), January 1996.

[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 the Theory of Computing (STOC '97)*, pages 220–229, New York, May 1997. Association for Computing Machinery.

[Lu04]   C.J. Lu. Encryption against storage-bounded adversaries from on-line strong extractors. *Journal of Cryptology*, 17(1):27–42, January 2004.

[NW94]   N. Nisan and A. Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49:149–167, 1994.

[RRV99]   R. Raz, O. Reingold, and S. Vadhan. Extracting all the randomness and reducing the error in trevisan's extractor. In *Proceedings of the 30th ACM Symposium on Theory of Computing*, pages 149–158. ACM Press, May 1999.

[RTS00]   J. Radhakrishnan and A. Ta-Shma. Tight bounds for dispersers, extractors, and depth-two superconcentrators. *SIAM Journal on Discrete Mathematics*, 13(1):2–24, February 2000.

[Sha02]   R. Shaltiel. Recent developments in explicit constructions of extractors. *Bulletin EATCS*, 77:67–95, June 2002.

[SU01]   R. Shaltiel and C. Umans. Simple extractors for all min-entropies and a new pseudo-random generator. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, 2001.

[Tre99]   L. Trevisan. Constructions of near-optimal extractors using pseudo-random generators. In *Proceedings of the 30th ACM Symposium on Theory of Computing*, pages 141–148. ACM Press, May 1999.

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

[Tre04]   L. Trevisan. Some applications of coding theory in computational complexity. Technical Report Report No. 43, Electronic Colloquium on Computational Complexity, September 2004. Available at `http://www.eccc.uni-trier.de/eccc-local/Lists/TR-2004.html`.

[TSZ01]   A. Ta-Shma and D. Zuckerman. Extractor codes. In *Proceedings of the 33rd ACM Symposium on Theory of Computing*, pages 193–199, 2001.

[TSZS01]   A. Ta-Shma, D. Zuckerman, and S. Safra. Extractors from Reed-Muller codes. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, 2001.

[Vad04]   S. Vadhan. On constructing locally computable extractors and cryptosystems in the bounded-storage model. *J. of Cryptology*, 17(1):43–77, January 2004.

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

[Zim04]   Marius Zimand. *Computational Complexity: A Quantitative Perspective*. North-Holland Mathematics Studies. vol. 196. Elsevier, 2004.