Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 113 (2022)

Leakage-Resilient Hardness v.s. Randomness

Yanyi Liu
Cornell Tech
yl2866@cornell.edu

Rafael Pass*
Tel-Aviv University & Cornell Tech
rafaelpass@tau.ac.il

February 10, 2023

Abstract

A central open problem in complexity theory concerns the question of whether all efficient randomized algorithms can be simulated by efficient deterministic algorithms. The celebrated "hardness v.s. randomness" paradigm pioneered by Blum-Micali (SIAM JoC'84), Yao (FOCS'84) and Nisan-Wigderson (JCSS'94) presents hardness assumptions under which e.g., $\mathsf{prBPP} = \mathsf{prP}$ (so-called "high-end derandomization), or $\mathsf{prBPP} \subseteq \mathsf{prSUBEXP}$ (so-called "low-end derandomization), and more generally, under which $\mathsf{prBPP} \subseteq \mathsf{prDTIME}(\mathcal{C})$ where $\mathcal{C}$ is a "nice" class (closed under composition with a polynomial), but these hardness assumptions are not known to also be necessary for such derandomization.

In this work, following the recent work by Chen and Tell (FOCS'21) that considers "almost-all-input" hardness of a function f (i.e., hardness of computing f on more than a finite number of inputs), we consider "almost-all-input" *leakage-resilient hardness* of a function f—that is, hardness of computing $f(x)$ even given, say, $\sqrt{|x|}$ bits of leakage of $f(x)$. We show that leakage-resilient hardness characterizes derandomization of prBPP (i.e., gives a both *necessary* and *sufficient* condition for derandomization), both in the high-end and in the low-end setting.

In more detail, we show that there exists a constant c such that for every function T, the following are equivalent:

- $\mathsf{prBPP} \subseteq \mathsf{prDTIME}(\mathsf{poly}(T(\mathsf{poly}(n))))$;
- Existence of a $\mathsf{poly}(T(\mathsf{poly}(n)))$-time computable function $f : \{0,1\}^n \to \{0,1\}^n$ that is almost-all-input leakage-resilient hard with respect to n^c-time probabilistic algorithms.

As far as we know, this is the first assumption that characterizes derandomization in both the low-end and the high-end regime.

Additionally, our characterization naturally extends also to derandomization of prMA, and also to average-case derandomization, by appropriately weakening the requirements on the function f. In particular, for the case of average-case (a.k.a. "effective") derandomization, we longer require the function to be almost-all-input hard, but simply satisfy the more standard notion of average-case leakage-resilient hardness (w.r.t., every samplable distribution), whereas for derandomization of prMA, we instead consider leakage-resilience for relations.

*Supported in part by NSF Award CNS 2149305, NSF Award SATC-1704788, NSF Award RI-1703846, AFOSR Award FA9550-18-1-0267, a JP Morgan Faculty Award, and an Algorand Foundation grant (MEGA-ACE). This material is based upon work supported by DARPA under Agreement No. HR00110C0086. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA.

ISSN 1433-8092

1 Introduction

Randomness is an ubiquitous tool in algorithm design. A central open problem in complexity theory concerns the question of whether all randomized algorithms can be derandomized; that is, can every randomized polynomial-time algorithm be simulated by a deterministic polynomial-time, or perhaps even just sub-exponential one? In this work, we consider this question with respect to promise problems; as usual, we refer to prBPP as the class of promise problems (as opposed to languages) that can be solved in probabilistic polynomial time (with 2-sided error), and prP to the class of promise problems than can be solved in deterministic polynomial time.

We here focus on the questions of whether $\mathsf{prBPP} = \mathsf{prP}$ (the, so-called, "high-end regime"), $\mathsf{prBPP} \subseteq \mathsf{prSUBEXP}$ (the, so-called, "low-end regime") and more generally whether $\mathsf{prBPP} \subseteq \mathsf{prDTIME}(\mathcal{C})$, where $\mathcal{C}$ is a "nice" class of time bounds (where by "nice" we here mean that $\mathcal{C}$ is closed under composition with a polynomial).

A long sequence of works originating with the works of Blum-Micali [BM84], Yao [Yao82], Nisan [Nis91], Nisan-Wigderson [NW94], Babai-Fortnow-Nisan-Wigderson [BFNW93], Impagliazzo-Wigderson [IW97] have presented beautiful connections between this problem and the problem of proving computational-complexity lower bounds—the so-called *hardness v.s. randomness* paradigm. For instance, the results of [NW94, IW97] show that $\mathsf{prBPP} = \mathsf{prP}$ under the assumption that $\mathsf{E} = \mathsf{DTIME}(2^{O(n)})$ contains a language that requires Boolean circuits of size $2^{\Omega(n)}$ for almost all input lengths (i.e., E is not contained in $\mathsf{ioSIZE}(2^{\Omega(n)})$). Additionally, results by Impagliazzo, Kabanets and Wigderson [IKW02] show a *partial* converse: if $\mathsf{prBPP} = \mathsf{prP}$, then some non-trivial circuit lower bound must also hold. In more detail, if $\mathsf{prBPP} = \mathsf{prP}$ (or even just $\mathsf{MA} = \mathsf{NP}$), then $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$; very recent works [Tel19, MW18] managed to strengthen the conclusion to e.g., $\mathsf{NTIME}[n^{\mathsf{poly}\log n}] \not\subseteq \mathsf{P/poly}$.

But despite over 40 years of research on the topic of derandomization, there has still been a large "gap" between the hardness assumptions required for derandomizing prBPP, and the ones that are known to be necessary for derandomization, leaving open the following question:

> *For "nice" classes $\mathcal{C}$, does there exist some (natural) hardness assumption that is* **equivalent** *to $\mathsf{prBPP} \subseteq \mathsf{prDTIME}[\mathcal{C}]$?*

Most notably, known derandomization results for prBPP require complexity lower-bounds on functions in EXP, whereas it is only known that derandomization of prBPP implies complexity lower bounds for functions in non-deterministic classes.

There as been some recent progress on the above problem:

- An elegant work by Chen, Rothblum, Tell and Yogev show an equivalence between derandomization and circuit lower bound under a conjecture (a weaker version of the non-deterministic exponential-time hypothesis) [CRTY20]. There work applies for both the high-end and the low-end regime, but is only conditional (i.e., relies on a conjecture).

- Chen and Tell [CT21], relying on the work by Goldreich [Gol11a], show that the existence of a multi-output function $f : \{0,1\}^n \to \{0,1\}^n$ computable by polynomial size logspace-uniform circuits with depth bounded by n^2 that cannot be computed in some (a-priori bounded) probabilistic polynomial time on *any* sufficiently large input—this is referred to as *"almost-all-input hardness"*—implies that $\mathsf{prBPP} = \mathsf{prP}$. They also show a partial converse: That a relaxed version of this conjecture, where the depth requirement is dropped, also is necessary.

- Liu and Pass [LP22], following the work of Hirahara [Hir20], demonstrates a *class* of promise problems (related to conditional time-bounded Kolmogorov complexity) such that (worst-case)

hardness with respect to a-prior polynomially bounded probabilistic algorithms of *all* problems in the class is equivalent to $\mathsf{prBPP} = \mathsf{prP}$. (The result of Liu and Pass also shows that a single problem can be used to characterize $\mathsf{prBPP} = \mathsf{prP}$ but this problem is very artificial.) Similar to the results of [CT21], this characterization can be extended slightly beyond the "high-end regime", but fails to capture the "low-end regime": it only works to handle derandomization in time $\mathcal{C}$, where $\mathcal{C}$ is a class closed under composition (i.e., if $T \in \mathcal{C}$, then $T(T(\cdot) \in T)$, and thus already does not apply to e.g., SUBEXP.

- Finally, a very recent elegant work by Korten [Kor22] demonstrates a natural search problem—the R-Lossy Code problem—that is complete for prBPP. As such, the assumption that this problem can be solved in deterministic time $\mathcal{C}$ characterizes when $\mathsf{prBPP} \subseteq \mathsf{DTIME}(\mathcal{C})$. We note, however that this assumption is not a hardness assumption, but rather an "easiness" assumptions.

Thus summing up, it is known how to characterize both the high-end and low-end derandomization through a hardness assumption under a conjecture [CRTY20]; unconditionally, however, it is only known how to characterize the high-end regime (i.e., $\mathsf{prBPP} = \mathsf{prP}$) and even there it is only known under either (a) a class of hardness assumptions (as opposed to one), or (b) a very specific and artificial single hardness assumption.

In this work, we follow the work by Chen and Tell [CT21] and also consider the notion of "almost-all-input" hardness of a multi-output function. In contrast to them, however, we consider such "almost-all-input" hardness in the context of *leakage resilience*, a notion first considered in the cryptographic literature in the 1980s. As we shall see, our main result show that "almost-all-input" leakage resilient hardness can be used to fully characterize derandomization, both in the high-even and in the low-end setting; additionally, our characterization will extend also to derandomization of prMA, and also to average-case (a.k.a. "effective") derandomization. In particular, for the case of average-case derandomization, we longer require the function to be almost-all-input hard, but simply satisfy the standard notion of average-case leakage-resilient hardness (w.r.t., every samplable distribution). And to characterize derandomization of prMA, we instead consider the notion of a *leakage-resilient relation*, which again is a notion considered in the cryptographic literature. Taken together, we believe that our results demonstrate an intriguing connection between derandomization and notions from cryptography.

1.1 Leakage-resilient Hardness

Consider some multi-output function $f : \{0,1\}^n \to \{0,1\}^n$. Roughly speaking, we say that f is T-*hard* if no $T(|x|)$-time algorithm/attacker A can compute $f(x)$ given any input x. Perhaps the most widely known example of a candidate construction of a hard function is *integer factorization*: given a product of two-primes x, compute the factorization $f(x)$ of x. In 1985, Rivest and Shamir [RS85] asked the question of what happens to this candidate hard function if the attacker gets some additional "side-information" about the factorization of x. Namely, the attacker gets not only x, but also some T-time computable side-information (a.k.a. "leakage") $\mathsf{leak}(x, f(x))$. Of course, if $|\mathsf{leak}(x, f(x))| \geq |f(x)|$, then the problem becomes trivial since the side-information can simply reveal the whole factorization; in fact, for the particular factorization problem, it trivially suffices to leak $n/2$ bits to reveal just one of the primes. Rivest and Shamir [RS85] show that, in fact, it suffices to get $n/3$ bits of leakage for the function to becomes easy; this result was improved by Coppersmith [Cop97] to $n/4$ bits, and a heuristic (with a conjectured polynomial running-time bound) by Maurer [Mau92] shows an attack given just ϵn bits of leakage for any constant $\epsilon > 0$. As far as we know, it is unknown if this problem can be solved using just n^ϵ bits of leakage, for any $\epsilon < 1$.

The notion of leakage-resilient hardness captures the notion that a function is hard even given some *bounded-length* leakage [RS85, Mau92, AGV09]: We say that a function f is (T, ℓ)-*leakage resilient hard* if no $T(|x|)$-time attacker A can compute $f(x)$ given x and $\mathsf{leak}(x, f(x))$ for any $T(|x|)$-time computable leakage function leak that outputs at most $\ell(|x|)$ bits. In recent years, *leakage-resilient cryptography* [ISW03, MR04, DP08, AGV09]—the design of cryptographic protocols resilient to some forms of leakage of honest players' secrets—has received significant attention and has become a subfield in cryptography; the bounded-length leakage model is the most common way of formalizing the class of leakage functions.

In this paper, we will consider this notion of leakage-resilient hardness, except that following Chen and Tell, we will also consider it in the context of almost-all-input hardness: that is, for any pair (A, leak) there can be at most finitely many x for which A can compute $f(x)$ given x and $\mathsf{leak}(x, f(x))$.

1.2 Characterizing Derandomization

We are now ready to state our main theorem, which shows that almost-all-input leakage-resilient hardness characterizes both high-end and low-end derandomization. We say that a class of time-bounds $\mathcal{C}$ is *nice* if for all polynomials p, q it holds that if $T \in \mathcal{C}$, then $p(n)T(q(n)) \in \mathcal{C}$. For instance, the sets $\mathsf{poly}(n)$ and $2^{n^{o(1)}}$ are nice and recall that $\mathsf{P} = \mathsf{DTIME}(\mathsf{poly})$ and $\mathsf{SUBEXP} = \cap_{\epsilon>0}\mathsf{DTIME}(2^{n^\epsilon})$.

Derandomizing prBPP Our main theorem gives a characterization of derandomization of prBPP:

Theorem 1.1. *There exists a constant c such that for every "nice" class of running-time bounds $\mathcal{C}$, and for every $0 < \epsilon < 1$, the following are equivalent:*

- $\mathsf{prBPP} \subseteq \cup_{T\in\mathcal{C}}\mathsf{prDTIME}[T(n)]$.
- *The existence of a multi-output function $f : \{0,1\}^n \to \{0,1\}^n$ computable in deterministic time $T \in \mathcal{C}$ such that f is almost-all-input (n^c, n^ϵ)-leakage-resilient hard.*

In particular, $\mathsf{prBPP} = \mathsf{prP}$ (resp. $\mathsf{prBPP} \subseteq \mathsf{prSUBEXP}$) iff there exists a polynomial-time (resp. subexponential-time) computable multi-output function f that is almost-all-input $(n^c, \sqrt{n})$-leakage-resilient hard.

Some Corollaries to Leakage-resilient Hardness For the proof of Theorem 1.1 , in one direction, we only require leakage-resilience with a "small" amount of leakage, whereas in the other direction, we obtain a function that is leakage-resilient hard with a "large" amount of leakage. Consequently, our proof actually also yields an amplification theorem for leakage-resilient hardness: For any $\epsilon > 0, d \geq 1$, the existence of an efficient (n^c, n^ϵ)-leakage-resilient hard function implies an efficient $(n^d, n - \Omega(\log n))$-leakage-resilient hard function.

Theorem 1.2. *There exists a constant c such that for every nice class $\mathcal{C}$, all constants $\epsilon > 0, d \geq 1$, the following holds. If there exists a $\mathcal{C}$-computable almost-all-input (n^c, n^ϵ)-leakage-resilient hard function. Then there exists a $\mathcal{C}$-computable almost-all-input $(n^d, n - 3\log n)$-leakage-resilient hard function.*

Let us highlight that as far as we know, this is the first type of leakage-resilience amplification result in the literature that we are aware of. (Brakerski and Kalai [BK12] demonstrate a parallel repetition theorem for leakage-resilience but it does not show how to amplify the amount of leakage a function is resilient against, but rather only how to maintain the (relative) amount of leakage.)

Additionally, by combing the result of Chen and Tell [CT21] with our Theorem 1.1, we get an interesting (one-sided) connection between low-depth computable hard functions and leakage-resilience:

Theorem 1.3. *There exists some c such that the following holds. If there exists a function f computable by polynomial-size logspace-uniform circuits with depth bounded by n^2 that is almost-all-input n^c-hard, then for any constant $d \geq 1$, there exists a polynomial-time computable almost-all-input $(n^d, n - 3\log n)$-leakage-resilient hard function.*

Comparison with IW: leakage-resilient *local* hardness Recall that Impagliazzo and Wigderson [IW97] shows that $\mathsf{prBPP} = \mathsf{prP}$ under the assumption that $\mathsf{E} \not\subseteq \mathsf{ioSIZE}(2^{\Omega(n)})$ (i.e., that there exists some exponential-time computable function that does not have $2^{\Omega(n)}$-size circuits.) Since Theorem 1.1 shows that leakage-resilient hardness is both a sufficient and necessary condition for derandomizing prBPP, it directly follows that $\mathsf{E} \not\subseteq \mathsf{ioSIZE}(2^{\Omega(n)})$ implies leakage-resilient hardness (by combining [IW97] with Theorem 1.1), but it gives little insight into whether the type of assumption used by IW is inherent, or to what extent it "overshots". Indeed, understanding to what extent the NW/IW framework is inherent for derandomization is a long standing open problem.

We now show how to use our framework to provide an (in our eyes) crisp answer to this question. We start by noting that by a slight adjustment to the proof of Theorem 1.1, an (a-priori) weaker notion of leakage-resilient *local* hardness actually suffices to derandomize prBPP: Given a function $f : \{0,1\}^n \to \{0,1\}^n$, we say that A *t-locally computes $f(\cdot)$* on input x if for every $i \in [|x|]$, $A(x,i) = f(x)_i$ (i.e., the ith bit of $f(x)$), and $A(x,i)$ runs in time bounded by $t(|x|)$; we analogously say that A *t-locally computes f on input x given (T,ℓ)-leakage* leak if for every $i \in [|x|]$, $A(x, \mathsf{leak}(x, f(x)), i) = f(x)_i$, $A(x,z,i)$ runs in time bounded by $t(|x|)$, $\mathsf{leak}(x, f(x))$ runs in time bounded by $T(|x|)$, and $|\mathsf{leak}(x, f(x))| \leq \ell(|x|)$. We finally say that f is *almost-all-input (T,ℓ)-leakage resilient t-local hard* if there does not exists (A, leak) such that A t-locally computes f on infinitely many x given (T,ℓ)-leakage leak. Note that this notion of leakage-resilient hardness differs from the standard one in two ways: (a) we are decoupling the running time T of the leakage function, and the running time t of the computing machine A, and (b) we require the computing machine A to be able to locally compute each bit of the output of $f(x)$—this will allow us to consider sublinear running times t. Indeed, we will focus our attention on the regime where A is required to reconstruct each bit of $f(x)$ in sublinear time: We say that f is simply *almost-all-input (T,ℓ)-leakage resilient locally hard* if there exists some $0 < \epsilon < 1$ such that f is almost-all-input (T,ℓ)-leakage resilient t-locally hard where $t(n) = n^\epsilon$. Note that (T,ℓ)-leakage resilient hardness is trivially an (a-priori) stronger condition than (T,ℓ)-leakage resilient local hardness when $T \in \Omega(n^2)$. We now have the following generalization of Theorem 1.1:

Theorem 1.4. *There exists a constant $c \geq 2$ such that for every "nice" class of running-time bounds $\mathcal{C}$, and for every $0 < \epsilon < 1$, the following are equivalent:*

- $\mathsf{prBPP} \subseteq \cup_{T \in \mathcal{C}} \mathsf{prDTIME}[T(n)]$.
- *The existence of a multi-output function $f : \{0,1\}^n \to \{0,1\}^n$ computable in deterministic time $T \in \mathcal{C}$ such that f is almost-all-input (n^c, n^ϵ)-leakage-resilient* locally *hard.*
- *The existence of a multi-output function $f : \{0,1\}^n \to \{0,1\}^n$ computable in deterministic time $T \in \mathcal{C}$ such that f is almost-all-input (n^c, n^ϵ)-leakage-resilient hard.*

Leakage-resilient local hardness is useful as it allows to capture the assumption that $\mathsf{E} \not\subseteq \mathsf{ioSIZE}(2^{\Omega(n)})$. In fact, we observe that $\mathsf{E} \not\subseteq \mathsf{ioSIZE}(2^{\Omega(n)})$ directly implies the existence of a (n^c, n^ϵ)-leakage-resilient locally hard function for every c. To see this, consider some E-computable function $g_n : \{0,1\}^n \to$

$\{0,1\}$ that does not have circuits of size $2^{\Omega(n)}$; in other words, g_n cannot be computed by a $2^{\Omega(n)}$ time algorithm even when given any (potentially non computable) advice string. Define the multi-output function f that is *constant* on each input length n and simply outputs the truthtable of $g_{\log n}$; that is,

$$f(x) = g_m(1)||g_m(2)||\dots||g_m(|x|)$$

where $m = \log |x|$. Note that $f(x)$ is polynomial-time computable (since we are only evaluating g_m on input of logarithmic length in $|x|$), and additionally by the hardness of g, it directly follows that f is almost-all-input leakage-resilient *locally* hard (since locally computing f on some input x in sublinear time with *efficiently computable* leakage implies computing $g_{\log |x|}(y)$ on every input $y \in \{0,1\}^{\log |x|}$ in subexponential time with advice).[1] In fact, it directly follows that this construction is almost-all-input leakage-resilient locally hard even with respect to *uncomputable* leakage.

In other words, the IW assumption "overshoots" the minimal assumption needed for derandomizing BPP in two ways: (a) it considers leakage-resilient hardness of a "degenerated" multi-output function that is *constant* on each input length, and (b) it requires leakage resilient hardness also with respect to *uncomputable* leakage, whereas the minimal assumption only requires it w.r.t. *polynomial-time* computable leakage.

Effective Derandomization Goldreich [Gol11a] (see also [Gol11b, Kab01]) considers a notion of "effective" derandomization of prBPP where the derandomizer does not need to work on all inputs—rather, it can fail sometimes, but only on inputs that are "hard to find"—in more detail, no PPT finder/refuter can find instances on which the derandomization fails except with negligible probability. In essence, effective derandomization is good enough for all efficient applications of derandomization.

For technical reasons, however, Goldreich, is not able to characterize such effective derandomization, but rather only a notion of $p(\cdot)$-effective derandomization where the finder/refuter running time is bounded by $p(n)$ for some fixed polynomial p and it success probability is bounded by $\frac{1}{p(n)}$. (In more details, for every a-priori fixed polynoomial bound $p(\cdot)$ on the running-time/success probability of a refuter, we require the existence of derandomization that works for that particular bound. In contrast, effective derandomization (as we consider it here) requires the existence of a single derandomization procedure that works for *any* polynomial-time refuter, and with only negligible failure probability.)

Using our leakage-resilient framework, we can get a clean characterization also of effective derandomization through average-case leakage-resilient hardness, where average-case leakage-resilient hardness with respect to some distribution $\mathcal{D}$ is defined just like before except we now consider instances x sampled from $\mathcal{D}$ and we allow the attacker A to succeed on at most a negligible fraction of instances.

More precisely, we say that Π is *effectively contained* in Π' (denoted $\Pi \subseteq_{poly} \Pi'$) if for every PPT A there exists a negligible μ such that the probability that $A(1^n)$ is able to output an n-bit element in the symmetric difference between Π and Π' is bounded by $\mu(n)$; we may extends this notion of classes of problems in the usual way: $\mathcal{D} \subseteq_{poly} \mathcal{D}'$ iff for every $\Pi \in \mathcal{D}$, there exists some $\Pi' \in \mathcal{D}'$ such that $\Pi \subseteq_{\mathsf{poly}} \Pi'$.

Theorem 1.5. *There exists a constant c such that for every $0 < \epsilon < 1$, the following are equivalent:*

- prBPP $\subseteq_{\mathsf{poly}}$ prP.

[1] There is a minor subtlety here. If we have an attacker A that locally computes $f(x)$ on some input x in sublinear time, then A can compute $g_{\log |x|}(y)$ for every $y \in \{0,1\}^{\log |x|}$ in sublinear time *given access to* x which can be of exponential length compared to $|y|$. But since A runs in sublinear time, it can access at most a sublinear number of bits of x, and thus we can compute g_m by a circuit of subexponential size.

- *The existence of a multi-output function $f : \{0,1\}^n \to \{0,1\}^n$ computable in deterministic polynomial time such that f is average-case (n^c, n^ϵ)-leakage-resilient hard for every efficiently samplable distribution $\mathcal{D}$.*

The result also extends to the low-end regime but only if we consider a stronger form of effective containment (which allows the refuter to have super polynomial running time).

Derandomizing prMA Finally, we consider the problem of derandomizing prMA (as opposed to just prBPP). Here, we require considering the notion of a leakage-resilient hard *relation* [Pas20], which is identically defined to that of a leakage-resilient hard function, except that any input x can be mapped to multiple values $y \in R(x)$. We show:

Theorem 1.6. *There exists a constant c such that for every "nice" class of running-time bounds $\mathcal{C}$, and for every $0 < \epsilon < 1$, the following are equivalent:*

- $\mathsf{prMA} \subseteq \cup_{T \in \mathcal{C}} \mathsf{prNTIME}[T(n)]$.
- *The existence of a relation $R \subset \{0,1\}^n \times \{0,1\}^n$ computable in non-deterministic time $T \in \mathcal{C}$ such that R is almost-all-input (n^c, n^ϵ)-leakage-resilient hard.*

In particular, prMA = prNP iff there exists a relation $R \in \mathsf{NP}$ that is almost-all-input $(n^c, \sqrt{n})$-leakage-resilient hard.

Proof Overview We focus our attention on the proof of Theorem 1.1 (and Theorem 1.4); afterwards, we briefly discuss how to extend these techniques to prove the remaining results. For simplicity, here focusing only on the high-end setting, but the key point is that the same technique *directly* extends also to the low-end setting.

- **Leakage-resilient Hardness implies prBPP = prP:** Following Goldreich [Gol11a], we consider the notion of a *targeted PRG*—roughly speaking, this is a PRG g that gets an additional *target* z as input, and indistinguishability holds with respect to uniform algorithms that also get the target z as input. In other words, g is just like a normal PRG, but with the exception that both the PRG and the distinguisher get access to the auxiliary "target" string z, and we require security to hold for all strings z. (Since we consider PRGs in the context of derandomization, we allow the running-time of the PRG to be (polynomially) larger than the running-time of the distinguisher.) Using standard techniques, it follows that the existence of such a targeted PRG, with sufficiently large stretch, implies that prBPP = prP (simply let the instance to be decided be the target for the PRG).

  We next show how to use leakage-resilient hardness to construct a targeted PRG. Assume the existence of a leakage-resilient hard multi-output function $f : \{0,1\}^n \to \{0,1\}^n$. Given a target $z \in \{0,1\}^n$, we compute $g_z = \mathsf{ECC}(f(z))$, where ECC is an appropriate list-decodable error-correcting code with good parameters, and interpret g_z as a hard function to use in the Impaglizzo-Wigderson (IW)/Nisan-Wigderson (NW) pseudo-random generator generator [IW97, NW94]. That is, we are relying on the Sudan-Trevisan-Vadhan PRG [STV01]. The IW-NW proof essentially shows that given a distinguisher D for the PRG, and some (bounded-length) advice about the truthtable (and D), we efficiently compute the evaluation of g_z with probability $1/2 + \frac{1}{p(n)}$ for some polynomial p over random n-bit inputs. We observe that this advice in fact can be efficiently computed if we have access to the truthtable of g_z and the distinguisher D, and we can thus view it as efficiently leakage on $(z, f(z))$. We can next list-decode (again efficiently) g_z and recover a polynomial-length list of candidates for $f(z)$;

given $f(z)$, we can efficiently determine which of these candidates is the correct one, and also include the index of this candidate in the leakage (which again will be short). Given both these leakage, we can now re-compute $f(z)$ by simply again running the list-decoding algorithm and outputting the string specified by the index. We note that we here rely on the fact that once this leakage has been fixed, the rest of the NW reconstruction procedure is deterministic, and furthermore, the list-decoding procedure is also deterministic, so the attacker A can recompute the same list of candidates in the same order, and thus we are guaranteed that it also recovers the exact same string.

In other words, if anyone can break the targeted PRG on infinitely many targets z, we can compute $f(z)$ on infinitely many inputs z given short and efficiently computable leakage on $(z, f(z))$; we note that, somewhat curiously, the leakage function actually also needs to access z and not just $f(z)$ in order to simulate the distinguisher D (that gets z as an input).

We finally observe that if the error-correcting code additionally satisfies a *local* list decoding property—e.g., by using the error-correcting code of [STV01]— then we can actually locally compute each bit of $f(z)$ in sublinear time in the length of $f(z)$ which we can use to conclude also the implication in the proof of Theorem 1.4. There is just one small catch; the local list decoding procedure will be randomized, so we may not necessarily recover the same list of candidates, or the same ordering of them. But the list-decoding procedure has a small running time and thus also uses a small amount of randomness, so we can just include this additional randomness as part of the leakage.

In the actual proof, we show the above in a more modular way:

  - We first consider a notion of a *strongly black-box PRG* —roughly speaking a PRG based on f for which there exists (a) an efficient algorithm that given black-box access to f and some distinguisher for the PRG outputs some advice string, and (b) another efficient algorithm that given this advice string and black-box access to the same distinguisher, is able to efficiently compute f. This notion is a strengthening of the notion of a black-box PRG from [Vad12] where the advice string did not need to be efficiently computable. Nevertheless, following [IW98], we note that the advice string needed in the reduction to prove security of the [STV01] PRG construction actually can be efficiently computed.
  - Next, we show that any such strongly black-box PRG construction can be used to get a targeted PRG from leakage-resilient hardness.

- $\mathsf{prBPP} = \mathsf{prP}$ **implies Leakage-resilient Hardness:** Our proof, roughly speaking, proceeds in two steps. First, we show using an information theoretic argument that a random function f is almost-all-input leakage-resilient hard. Next, we show how this function can be "derandomized" assuming $\mathsf{prBPP} = \mathsf{prP}$—the crucial aspect that makes this derandomization possible is that it is possible to *efficiently* verify whether there exists some attacker that efficiently computes the function on some input given efficient leakage (by enumerating the $\log n$ first Turing machines and evaluating them).

  For the first step, we use a simple compression argument to show that for every attacker, leakage-function pair (A, leak), for any input x, with high probability over the choice of $y = F(x) \in \{0,1\}^n$, it is the case that the attacker A can compute $F(x)$ with probability at most, say, $1/6$. In fact, we will show a slightly stronger statement: for any A, x, with high probability over y, there does not exists any leakage function, leak, such that (A, leak) computes y with probability $1/6$. The advantage of this stronger formulation is that now it is without of generality to restrict attention to *deterministic* leakage functions leak (since for any fixed A, x, y we can always consider the deterministic leakage function that fixes the best randomness).

To prove the (stronger) statement, let us first consider the case when also the attacker A is deterministic. Since the length ℓ of the leakage is significantly shorter than $|y|$, it follows that for any fixed x, most strings y are not in the range of what the attacker can output given x and *any* leakage, and thus for every x, with high probability over the choice of y, the attacker fails to output y no matter what the function leak is.

Next, note that even if A is randomized, there can be at most 6 strings that A outputs with probability $1/6$ given any fixed x and any fixed leakage output, so the above argument actually also extends to randomized A (except that we increase the number of string y that can be hit by a factor 6). This thus concludes a random choice of y will with high probability not be computable by any (A, leak).

Next, we show that for any x, this random choice of y can actually be derandomized assuming $\mathsf{prBPP} = \mathsf{prP}$, and relying on the fact that we only need y to be hard to compute with respect to *uniform* (bounded) polynomial-time computable (A, leak). Towards this, we follow the approach of Goldreich [Gol11a] and show that for any x, we can greedily compute the bits of $y = f(x)$ one at a time, relying on the fact that we know that a random selection will work, and then use the fact that $\mathsf{prBPP} = \mathsf{prP}$ to efficiently find a good selection. In more details, we know that with high probability over the choice of y, the first $\log n$ uniform (A, leak) machines with running time bounded n^c will fail to compute y so we can start by picking bit 1 y_1 of y that leads to a high probability over the continuation of y of all those $\log n$ machines failing to compute y. Estimating this probability requires randomness, but if $\mathsf{prBPP} = \mathsf{prP}$ then it can also be done deterministically. This second step can be done in a modular way by appealing to the elegant BPP-decision-to-search reduction of Goldreich [Gol11a] and by appropriately specifying the above problem of finding a "good" y that fools the $\log n$ first uniform (A, leak) machine with running time n^c (in the sense that their estimated success probability is significantly smaller than $1/6$) as a BPP-search problem.

Effective Derandomization To characterize "effective derandomization", the proof follows a very similar structure, except to perform the derandomization we instead pass through a new notion of an "average-case targeted PRG". The converse direction (showing necessity of the assumption) becomes a bit more complicated than before as we no longer have access to a perfect derandomization, but these additional subtleties can be dealt with. (Roughly speaking, the issue is that since the derandomizer only succeeds on average, we may run into trouble during the decision to search process. On a high-level, the way we get around these issues is by relying on the fact that we only need to derandomize a *single* fixed problem, and that effective derandomization yields a single derandomized algorithm for solving it, and we can next show that if an error occurs during the decision to search process, the location of the first mistake can be efficiently guessed.)

Characterizing Derandomization of prMA To show how to derandomize prMA, we instead pass through a new notion of a non-deterministic targeted PRG, which can be instantiated from our assumption and which implies derandomization of prMA.

To prove the converse direction, we proceed a bit different from the proof of Theorem 1.1—we can no longer passing through the above-mentionned search-to-decision reduction, as no such reduction is known for prMA. Instead, we show how to directly construct a leakage-resilient hard relation from the assumption that prMA can be derandomized using a diagonalization argument. Roughly speaking, we show that the above described BPP-search problem (of given a string x, finding some y that is leakage-resilient hard to compute w.r.t. the first $\log n$ algorithms) actually is a leakage-reslient hard relation assuming that prMA can be derandomized! First note that this problem trivially is in

prMA and thus also in prNP under the assumption that $\mathsf{prMA} = \mathsf{prNP}$. More interestingly, if there exists some attacker A that given an input x and given some efficient leakage on x can compute a y in this relation, then this y cannot be in the relation (since by definition, y cannot be computed by any efficient algorithm with small description), which is a contradiction.

2 Preliminaries

We assume familiarity with basic concepts such as Turing machines, polynomial-time algorithms, and probabilistic algorithms and computational classes such as prBPP and prP. We say that a function f is *time-constructible* if f is increasing and for all $n \in \mathbb{N}$, $f(n)$ can be computed by a Turing machine in time $\mathsf{poly}(f(n))$. We say that a class of functions $\mathcal{C}$ is *nice* if for all $T \in \mathcal{C}$, $t(p(n))q(n) \in \mathcal{C}$ for all polynomials p, q.

We say that $\mathcal{D} = \{D_n\}_{n\in\mathbb{N}}$ is an *ensemble* if for all $n \in \mathbb{N}$, D_n is a probability distribution over $\{0,1\}^n$. We say that an ensemble $\mathcal{D} = \{D_n\}_{n\in\mathbb{N}}$ is *samplable* if there exists a probabilistic polynomial time Turing machine S such that $S(1^n)$ samples D_n.

2.1 Leakage Resilient Hardness of (Multi-output) Functions

We consider multi-output functions $f : \{0,1\}^n \to \{0,1\}^n$ (as opposed to binary functions, traditionally considered in the derandomization literature). We will focus on a leakage-resilient notion of hardness. Roughly speaking, we say that f is (T, ℓ)-*leakage resilient hard* if no T-time attacker can compute $f(x)$ given x and $\mathsf{leak}(x, f(x))$, where leak is any T-time computable leakage function such that $\mathsf{leak}(x, f(x)) \leq \ell(|x|)$. We will consider this notion of in the context of *almost-all-input hardness* [CT21] which requires all potential attackers to succeed only on finitely many inputs.

Definition 2.1 (Almost-all-input leakage-resilient hardness)**.** *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a (multi-output) function. We say that f* is almost-all-input (T, ℓ)-leakage resilient hard *if for all T-time[2] probabilistic algorithms leak, A satisfying $\mathsf{leak}(x, f(x)) \leq \ell(|x|)$, for all sufficiently long strings x, $A(x, \mathsf{leak}(x, f(x))) \neq f(x)$ with probability $\geq 2/3$ (over their internal randomness).*

The notion of leakage-resilient *local* hardness will be useful for us. In the local hardness condition, we require no attacker A can produce each bit of $f(x)$ given the input x together with the coordinate. This allows us to consider attackers A that run in $|x|^\varepsilon$ time on input x.

Definition 2.2 (Almost-all-input leakage-resilient local hardness)**.** *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a (multi-output) function. We say that f* is almost-all-input (T, ℓ)-leakage resilient t-local hard *for all T-time probabilistic algorithms leak satisfying $\mathsf{leak}(x, f(x)) \leq \ell(|x|)$, for all t-time probabilistic algorithms A, for all sufficiently long strings x, $A(x, \mathsf{leak}(x, f(x)))$ $\mathsf{locally\ computes}$ $f(x)$ with probability at most $\geq 1/3$ (over their internal randomness), where we say that $A(x, \mathsf{leak}(x, f(x)))$ $\mathsf{locally\ computes}$ $f(x)$ if for all $i \in \{0,1\}^{\log |x|}$, $i \leq |x|$, $A(x, \mathsf{leak}(x, f(x)), i) = f(x)_i$.*

We simply say that f is almost-all-input (T, ℓ)-leakage resilient local hard *if there exists $\varepsilon > 0$ such that f is almost-all-input (T, ℓ)-leakage resilient n^ε-local hard.*

We remark that we are decoupling the running time T of the leakage function and the running time t of the computing machine A. In addition, we only require hardness with respect to $|x|^\varepsilon$-time attackers A which can only read the first $|x|^\varepsilon$ bits of the string x.[3] As mentioned in the

[2] To simplify notation, we say that an algorithm $A(\cdot,\cdot)$ runs in time T if A runs in $T(n)$ time where n is the size of the first input.

[3] We assume that A is a standard Turing machine with each input on a separate tape, and we do not assume that A has oracle access to its inputs. This is a *weaker* hardness assumption than letting A have oracle access to the inputs.

introduction, the notion of leakage-resilient local hardness enables us to capture the assumption that $\mathsf{E} \not\subseteq \mathsf{ioSIZE}(2^{\Omega(n)})$.

Lemma 2.3. *If there exists a constant $\varepsilon > 0$ such that $\mathsf{E} \not\subseteq \mathsf{ioSIZE}(2^{\varepsilon n})$, then there exists a function $f : \{0,1\}^n \to \{0,1\}^n$ that is $(n^c, n^{\varepsilon/2})$-leakage resilient $n^{\varepsilon/2}$-locally hard for every $c \geq 1$.*

Proof: Let $g_n : \{0,1\}^n \to \{0,1\}$ be a E-computable function that requires circuits of size $> 2^{\varepsilon n}$ to compute (which is guaranteed to exist by the assumption). Consider the following multi-output function f such that for each $x \in \{0,1\}^*$,

$$f(x) = g_k(1)||\dots||g_k(2^k)||0||\dots||0$$

where $k = \lfloor \log |x| \rfloor$. Note that $f(x)$ can be computed in time $2^{O(k)} = 2^{O(\log |x|)} = |x|^{O(1)}$ so f is poly-time computable. Assume for contradiction that f is not $(n^c, n^{\varepsilon/2})$-leakage resilient $n^{\varepsilon/2}$-locally hard. Then there exist attackers (A, leak) such that $\mathsf{leak}(x, f(x))$ outputs at most $|x|^{\varepsilon/2}$ bits and $A(x, \mathsf{leak}(x, f(x)))$ computes $f(x)$ locally in time $|x|^{\varepsilon/2}$ for infinitely many x. For each such x and input length $k = \lfloor \log |x| \rfloor$, we will construct a $2^{\varepsilon k}$-size circuit C_k to compute g on input length k, which is a contradiction. Since $A(x, \mathsf{leak}(x, f(x)))$ locally computes $f(x)$ in time $|x|^{\varepsilon/2}$, there exists a leakage string $w \in \{0,1\}^{|x|^{\varepsilon/2}}$ and a random tape $r \in \{0,1\}^{|x|^{\varepsilon/2}}$ such that for each $i \in \{0,1\}^{\log |x|}$, $A(x, w, i; r)$ will compute $f(x)_i$ within time $|x|^{\varepsilon/2}$, and it follows that $A(x, w, i; r)$ will only read the first $|x|^{\varepsilon/2}$ bits of the string x; let x' denote the $|x|^{\varepsilon/2}$-bit prefix of x. Consider a circuit C_k having the strings x', w, r hardwired in it, and on input $i \in \{0,1\}^k$, it will compute $A(x, w, i; r)$. C_k is of size $O(|x|^{\varepsilon/2} \log |x|) \leq 2^{\varepsilon k}$ that computes g on input length k, which concludes the proof. ■

In addition, we can also consider just "plain" (as opposed to leakage-resilient) hardness. As above, we also require the hardness condition holds on almost all inputs.

Definition 2.4 (Almost-all-input hardness)**.** *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a (multi-output) function. We say that f* is almost-all-input T-hard *if for all T-time probabilistic algorithms A, for all sufficiently long strings x, $A(x) \neq f(x)$ with probability $\geq 2/3$ (over their internal randomness).*

2.2 Targeted Pseudorandom Generator

We consider the notion of a *targeted pseudorandom generator (targeted PRG)* [Gol11a]. Roughly speaking, a targeted pseudorandom generator G takes a seed along with a "target" string x as input, and we require that its output is indistinguishable from uniform by (computationally-bounded) distinguishers that additionally get the target x as input. In other words, G is just like a normal PRG, but with the exception that both the PRG and the distinguisher get access to the auxiliary "target" string x, and we require security to hold for all strings x. Since we consider PRGs in the context of derandomization, we allow the running-time of the PRG to be larger than the running-time of the distinguisher.

Definition 2.5 (Targeted pseudorandom generator [Gol11a])**.** *Let $G : 1^n \times \{0,1\}^{\ell(n)} \times \{0,1\}^{m(n)} \to \{0,1\}^n$ be a computable function. We say that G is an $T(n)$-secure $(\ell(n), m(n))$-targeted pseudorandom generator (T-secure $(\ell(n), m(n))$-targeted PRG) if for all deterministic attackers D that run in $T(n)$ time (where n is the length of its first input), for all sufficiently large $n \in \mathbb{N}$ and all strings $x \in \{0,1\}^{\ell(n)}$, it holds that*

$$|\Pr[s \leftarrow \{0,1\}^{m(n)} : D(1^n, x, G(1^n, x, s)) = 1] - \Pr[y \leftarrow \{0,1\}^n : D(1^n, x, y) = 1]| < \frac{1}{6}.$$

For any targeted PRG G, we say that G is $O(T(n))$-secure if for all constant $c > 0$, G is $(cT(n))$-secure. Note that the length of the target string $\ell(n)$ can be potentially larger than the running time bound of the distinguisher $T(n)$. In such cases, we only require security with respect to distinguishers D which can only read the first $T(n)$ bits of the target string. Note that this is a weaker security requirement than allowing D to have oracle access to the target string.

It is well-known that a (non-uniformly) secure PRG can derandomize prBPP. When considering a targeted uniformly-secure PRG, the same derandomization result still holds. This in essence follows by the standard proof (that non-uniformly secure PRG derandomize prBPP), but with an additional padding argument to deal with the "target"/auxiliary input.

Lemma 2.6 ([Gol11a])**.** *Assume that there exist constants $\sigma \geq 1, \theta \geq 1$ and a $O(n)$-secure $(n^\theta, \sigma \log n)$-targeted PRG G that runs in time $t(n) \geq n$. Then, $\mathsf{prBPP} \subseteq \cup_{p,q \in \mathsf{poly}} \mathsf{prDTIME}[t(p(n))q(n)]$.*

For the sake of completeness, we refer the reader to Appendix A for a detailed proof.

3 Derandomization and Leakage Resilient Hardness

In this section, we show a characterization between derandomizing prBPP and the existence of almost-all-input leakage resilient hard functions. Our result can be adapted to both the high-end and the low-end setting.

Theorem 3.1. *There exists a constant $c \geq 1$ such that for all nice classes of functions $\mathcal{C}$, the following are equivalent.*

1. $\mathsf{prBPP} \subseteq \cup_{T \in \mathcal{C}} \mathsf{prDTIME}[T]$.
2. *The existence of a constant $\varepsilon > 0$, a function $T \in \mathcal{C}$, and an almost-all-input (n^c, n^ε)-leakage resilient locally hard function $f : \{0,1\}^n \to \{0,1\}^n$ computable in deterministic time T.*
3. *For all $d \geq 1$, there exist $T \in \mathcal{C}$ and an almost-all-input $(n^d, n - 3\log n)$-leakage resilient hard function $f : \{0,1\}^n \to \{0,1\}^n$ computable in deterministic time T.*

Proof: The implication (1) $\Rightarrow$ (3) follows from Theorem 3.8 (stated and proved in Section 3.1). To show (2) $\Rightarrow$ (1), we apply Lemma 3.9 (stated and proved in Section 3.2) to obtain a targeted PRG and (1) follows from Lemma 2.6. (3) trivially implies (2). ■

We then state corollaries of Theorem 3.1 in both the high-end regime and the low end regime. To characterize derandomizing prBPP in polynomial time, we take the class $\mathcal{C}$ in Theorem 3.1 to be the class of polynomials $\mathsf{poly}(\cdot)$.

Corollary 3.2. *There exists a constant $c \geq 1$ such that the following holds. $\mathsf{prBPP} = \mathsf{prP}$ if and only if there exists an efficiently computable multi-output function f that is almost-all-input $(n^c, \sqrt{n})$-leakage resilient hard.*

To characterize derandomizing prBPP in subexponential time $\mathsf{prSUBEXP} = \cap_{\varepsilon > 0} \mathsf{prDTIME}[2^{n^\varepsilon}]$, we consider the class $\mathcal{C}$ consisting of (all) time-constructible functions T such that $T(n)$ is smaller than 2^{n^ε} for all $\varepsilon > 0$[4], and we refer to a function f as being computable in subexponential time if f runs in time 2^{n^ε} for all $\varepsilon > 0$.

[4]Note that this class is indeed nice since if $T(n) < 2^{n^\varepsilon}$ for all $\varepsilon > 0$, it holds that $T(p(n))q(n) < 2^{n^\varepsilon}$ for all $\varepsilon > 0$ and all polynomials p, q.

Corollary 3.3. *There exists a constant $c \geq 1$ such that the following holds.* $\mathsf{prBPP} \subseteq \mathsf{prSUBEXP}$ *if and only if there exists an subexponential time computable multi-output function f that is almost-all-input $(n^c, \sqrt{n})$-leakage resilient hard.*

In addition, we note that the proof of Theorem 3.1 also yields the following amplification result for leakage resilient hardness.

Theorem 3.4. *There exists a constant c such that if there exist a constant $\varepsilon > 0$ and an almost-all-input (n^c, n^ε)-leakage-resilient hard function computable in time $t(n)$, then for all $d \geq 1$, there exist polynomials p, q and an almost-all-input $(n^d, n - 3\log n)$-leakage-resilient hard function computable in time $t(p(n))q(n)$.*

Proof: The theorem follows from Lemma 3.9, Lemma 2.6, and Theorem 3.8. ■

Besides amplifying leakage resilience, we observe that by combining the result of Chen and Tell with Theorem 3.1, we obtain a leakage-resilient hard function from a low-depth function with just plain hardness.

Theorem 3.5. *There exists some c such that the following holds. If there exists a function f computable by polynomial-size logspace-uniform circuits with depth bounded by n^2 that is almost-all-input n^c-hard, then for any constant $d \geq 1$, there exists a polynomial-time computable almost-all-input $(n^d, n - 3\log n)$-leakage-resilient hard function.*

Proof: Chen and Tell [CT21] showed that the existence of such f implies $\mathsf{prBPP} = \mathsf{prP}$. The existence of a leakage-resilient hard function then follows from Theorem 3.1. ■

3.1 Leakage Resilient Hardness from Derandomization

We proceed to constructing a multi-output function that is almost-all-input leakage resilient hard. Towards this, it is instructive to recall some ingredients from [Gol11a]. We first recall the definition of a prBPP search problem.

Definition 3.6 (prBPP search problem)**.** *Let R_{YES} and R_{NO} be two disjoint binary relations $\subseteq \{0,1\}^* \times \{0,1\}^*$. We say that $(R_{\mathsf{YES}}, R_{\mathsf{NO}})$ is a* prBPP search problem *if the following two conditions hold.*

1. *The decisional problem $(R_{\mathsf{YES}}, R_{\mathsf{NO}}) \in \mathsf{prBPP}$; that is, there exists a* PPT *algorithm V such that for every $(x, y) \in R_{\mathsf{YES}}$ it holds that $\Pr[V(x, y) = 1] \geq 2/3$, and for every $(x, y) \in R_{\mathsf{NO}}$ it holds that $\Pr[V(x, y) = 1] \leq 1/3$.*
2. *There exists a* PPT *algorithm A such that, for every $x \in S_{R_{\mathsf{YES}}}$, it holds that $\Pr[A(x) \in R_{\mathsf{YES}}(x)] \geq 2/3$, where $R_{\mathsf{YES}}(x) = \{y : (x, y) \in R_{\mathsf{YES}}\}$ and $S_{R_{\mathsf{YES}}} = \{x : R_{\mathsf{YES}} \neq \emptyset\}$*

It has also been shown in [Gol11a] that there exists a deterministic search to decision reduction for prBPP by using techniques resembling the Conditional Expectation Method.

Theorem 3.7 (Search to decision reduction [Gol11a])**.** *For every* prBPP *search problem $(R_{\mathsf{YES}}, R_{\mathsf{NO}})$, there exists a binary relation R such that $R_{\mathsf{YES}} \subseteq R \subseteq (\{0,1\}^* \times \{0,1\}^*) \backslash R_{\mathsf{NO}}$ and solving the search problem of R is polynomial-time deterministically reducible to some decisional problem in* prBPP*.*

Now we return to showing the existence of a multi-output function that is hard to compute on almost all inputs with leakage assuming prBPP can be derandomized.

Theorem 3.8. *If* $\mathsf{prBPTIME}[O(n)] \subseteq \mathsf{prDTIME}[t(n)]$*, then for any constant* $c \geq 1$*, there exists a function* $f : \{0,1\}^n \to \{0,1\}^n$ *running in time* $t(p(n))q(n)$ *(for some polynomials* p, q*) such that* f *is almost-all-input* $(n^c, n - 3\log n)$*-leakage resilient hard.*

Proof: We start by defining a prBPP-search problem which the task of constructing a leakage-resilient hard function can be reduced to. Consider any constant $c \geq 1$ and let $\ell(n) = n - 3\log n$. To construct an almost-all-input $(n^c, \ell(n))$-leakage resilient hard function, for each input $x \in \{0,1\}^n$, we need to find (uniformly) a string $f(x) = r$ such that r is hard to compute (for any n^c-time algorithms) given x and any (n^c-time) "side information" leaked from r. We observe that for *any* attacker/leakage functions g, leak—even non-computable g, leak—with high probability over r, r will satisfy the hardness with leakage condition w.r.t. g, leak.

Claim 1. *For any probabilistic algorithms* leak, g*, for all* $n \in \mathbb{N}$*,* $\ell \leq n$*,* $x \in \{0,1\}^n$*, with probability at most* $6 \cdot 2^{-n+\ell+1}$ *over random* $r \in \{0,1\}^n$*, it holds that*

$$\Pr\left[|\mathsf{leak}(x,r)| \leq \ell \wedge g(x, \mathsf{leak}(x,r)) = r\right] \geq \frac{1}{6} \tag{1}$$

Proof: Consider any $n \in \mathbb{N}$, $x \in \{0,1\}^n$, and any probabilistic algorithm g. We will show that with probability at most $6 \cdot 2^{-n+\ell+1}$ over random $r \in \{0,1\}^n$, for any *deterministic* function leak' that outputs $\leq \ell$ bits, it holds that

$$\Pr[g(x, \mathsf{leak}'(x,r)) = r] \geq \frac{1}{6} \tag{2}$$

The proof of Claim 1 will directly follow from Equation 2 by noting that given any probabilistic leak, g, any $n \in \mathbb{N}, \ell \leq n$, $x \in \{0,1\}^n$, we can consider the deterministic leak' obtained by fixing the random tape of leak to be such that it maximize the number r's that satisfy Equation 1.

To show Equation 2, consider the set of "bad" r's: $B = \{r : \exists w, |w| \leq \ell, \Pr[g(x,w) = r] \geq 1/6\}$. For any $r \in \{0,1\}^n$, if there exists a function leak' that outputs $\leq \ell$ bits and $\Pr[g(x, \mathsf{leak}'(x,r)) = r] \geq \frac{1}{6}$, it follows that $r \in B$. We now bound $|B|$. Note that are at most $2^{\ell+1}$ strings w such that $|w| \leq \ell$, and for each such w, there can be at most 6 strings ouput by g with probability $\geq 1/6$; thus we have that $|B| \leq 6 \cdot 2^{\ell+1}$. It follows that the probability that a random r falls into B is at most $6 \cdot 2^{-n+\ell+1}$. ■

Next, we note that if we only consider efficiently (and uniformly) computable g, leak, it suffices to consider attacker/leakage functions of description length no more than $\log n$. We can thus defined an prBPP-search problem that will enable us to find a "hard" r w.r.t. all such efficient attacker/leakage functions.

The BPP search problem: Let R_{YES} be a binary relation such that $(x,r) \in R_{\mathsf{YES}}$ if

1. $|x| = |r|$
2. For all probabilistic machines leak, g such that $|\mathsf{leak}| \leq \log n, |g| \leq \log n$, it holds that

$$\Pr\left[|\mathsf{leak}'(x,r)| \leq \ell(n) \wedge g'(x, \mathsf{leak}'(x,r)) = r\right] < \frac{1}{6} \tag{3}$$

   where n denotes $|x|$, and leak' and g' denote "time-truncated" versions of leak, g that are only executed for n^c steps, where c is the constant in the lemma statement.

Let R_{NO} be a binary relation such that $(x,r) \in R_{\mathsf{NO}}$ if for at least one pair of leak and g with $|\mathsf{leak}|, |g| \leq \log n$, the above equation *with* $\frac{1}{6}$ *replaced by* $\frac{1}{3}$ does *not* hold.

We turn to showing that $(R_{\mathsf{YES}}, R_{\mathsf{NO}})$ is a prBPP search problem by presenting a verifying algorithm V and a solution finding algorithm A.

The search problem verifier: On input (x, r), the verifier V enumerates all probabilistic machines leak, g such that $|\mathsf{leak}|, |g| \leq \log n$. V estimates the value

$$p_{\mathsf{leak},g} = \Pr\left[|\mathsf{leak}'(x, r)| \leq \ell(n) \wedge g'(x, \mathsf{leak}(x, r)) = r\right]$$

by running the following experiment for sufficiently many times and computing the average acceptance probability. In each experiment, V emulates $\mathsf{leak}(x, r)$ for n^c steps, and emulates $g(x, \mathsf{leak}(x, r))$ for n^c steps. V accepts in this experiment if $g(x, \mathsf{leak}(x, r)) = r$. After estimating the average acceptance probability for each pair of leak and g, V outputs 1 if the estimated values of $p_{\mathsf{leak},g}$ are $< \frac{3}{12}$ for all pairs of leak, g. By the Chernoff bound and the Union bound, V will accept if $(x, r) \in R_{\mathsf{YES}}$ (and reject if $(x, r) \in R_{\mathsf{NO}}$) with very high probability.

The solution finder: We next construct a solution finding algorithm A such that $(x, A(x)) \in R_{\mathsf{YES}}$ with high probability for all x. On input x, A simply outputs a random string of the same length. For any fixed $x \in \{0,1\}^n$, by Claim 1 and a Union bound over the choice of leak and g, we conclude that $A(x)$ outputs a valid witness with probability at least $1 - 6n^2 \cdot 2^{-n+\ell(n)+1} \geq \frac{2}{3}$.

Constructing the hard function f: Finally, we show how to construct a function f that is hard to compute in the presence of any leakage, by making use of the prBPP search problem $(R_{\mathsf{YES}}, R_{\mathsf{NO}})$. By Theorem 3.7, there exists a binary relation R such that $R_{\mathsf{YES}} \subseteq R \subseteq (\{0,1\}^* \times \{0,1\}^*) \backslash R_{\mathsf{NO}}$ and solving the search problem of R is polynomial-time deterministic reducible to some decisional prBPP problem. This leads us to our construction of f. On input x, f solves the search problem of R and outputs a R-witness of x. We first show that f is almost-all-input $(n^c, \ell(n))$-leakage resilient hard. Consider any n^c-time algorithms leak and g satisfying $|\mathsf{leak}(x, f(x))| \leq \ell(|x|)$, all sufficiently large inputs $x \in \{0,1\}^n$ such that $|g| \leq \log n$ and $|\mathsf{leak}| \leq \log n$. Since R and R_{NO} are disjoint and f solves the search problem of R, $(x, f(x)) \notin R_{\mathsf{NO}}$ and this implies that

$$\Pr\left[g(x, \mathsf{leak}(x, f(x))) \neq f(x)\right] \geq \frac{2}{3}$$

We turn to proving that f runs in deterministic time $t(p(n))q(n)$ for some polynomials p, q, which will conclude our proof. Recall that f can be polynomial-time deterministically reduced to a decisional problem $\Pi \in \mathsf{prBPP}$. Since $\mathsf{prBPTIME}[O(n)] \subseteq \mathsf{prDTIME}[t(n)]$, by padding instances in Π so that the probabilistic algorithm for Π now runs in linear time in the length of the padded instance, it follows that $\Pi \in \mathsf{prDTIME}[t(p'(n))]$ (for some polynomial p'). This, combined with the fact that the reduction runs in deterministic polynomial time, shows that f can be computed in deterministic time $t(p'(a(n)))b(n)$ for some polynomials a, b and the claim follows. ■

3.2 Derandomization from Leakage Resilient Hardness

We turn our attention to the converse direction and we will show how to obtain a targeted PRG, which is later used to derandomize prBPP, from an almost-all-input leakage resilient hard function. Combining the result from the previous section, we will obtain a characterization between derandomization and leakage resilient hardness.

Lemma 3.9. *There exists a constant $c \geq 1$ such that the following holds. Assume that there exist a constant $\varepsilon > 0$ and an almost-all-input (n^c, n^ε)-leakage resilient n^ε-locally hard function $f : \{0,1\}^n \to \{0,1\}^n$ computable in deterministic time $t(n) \geq n$. Then there exist constants $\sigma, \theta \geq 1$ and a $O(n)$-secure $(n^\theta, \sigma \log n)$-targeted PRG computable in time $t(n^\theta)\mathsf{poly}(n)$.*

The proof of Lemma 3.9 relies on the notion of *black-box PRG construction from a worst-case hard function* f [STV01, Vad12]. Roughly speaking, this notion of black-box PRG from a function f requires the existence of an efficient oracle algorithm that given (a) some fixed advice string, and (b) black-box access to any distinguisher for the PRG, is able to compute function f. Following, [IW98], we will here consider a strengthening of this notion of a black-box construction, simply referred to as *strongly black-box*, where also the advice string can be efficiently computed using black-box access to f.

Definition 3.10. *Let $g : 1^n \times 1^m \times \{0,1\}^d \to \{0,1\}^m$ be a (deterministic) oracle algorithm, and let $k(\cdot)$ be functions. We say that g is a* k-reconstructive PRG construction *if there exist* PPT *oracle algorithms R, M such that for every $f : [n] \to \{0,1\}$ and $T : \{0,1\}^m \to \{0,1\}$, if*

$$|\Pr[T(g^f(1^n, 1^m, \mathcal{U}_d)) = 1] - \Pr[T(\mathcal{U}_m) = 1]| \geq \frac{1}{6}$$

then $M^{f,T}(1^n, 1^m)$ will output at most $k(n,m)$ bits such that for all $i \in [n]$,

$$R^T(M^{f,T}(1^n, 1^m), i) = f(i)$$

with probability at least $2/3$.

We next observe that the Sudan-Trevisan-Vadhan PRG [STV01] obtain by combining a locally list-decodable error correcting code [STV01] and the Nisan-Wigderson PRG construction [NW94] yields a strongly black-box construction of a PRG. We note that [Vad12] previously argued that this construction is black-box; we here simply observe that the advice string needed can be efficiently computed.

Theorem 3.11 (Extending [STV01]; see also [Vad12, Theorem 7.67])**.** *There exists a k-reconstructive PRG construction $g : 1^n \times 1^m \times \{0,1\}^d \to \{0,1\}^m$ such that for every $m \in \mathbb{N}$, $n \geq m$, $f : [n] \to \{0,1\}$ the following conditions are satisfied:*

1. *Explicitness: g^f is computable in uniform time* $\mathsf{poly}(m,n)$.
2. *Seed length: $d(n,m) = O(\log^2 n / \log m)$.*
3. *Reduction advice length: $k(n,m) = \mathsf{poly}(m, \log n)$.*

Since the proof follows standard techniques, we have deferred it to Appendix B.

Return to proving Lemma 3.9 We are now ready to prove Lemma 3.9 by relying on the above result.

Proof: [of Lemma 3.9] Consider any constant $\varepsilon > 0$.

A padding trick. In this proof, we will use a padding argument to make the leakage we need as small as it is required. Let m denote the output length of the targeted PRG that we hope to construct. Let n denote the input length of the multi-output function f. Let $\theta = O(1/\varepsilon) \in \mathbb{N}$ be a constant such that $\frac{1}{\theta}$ is sufficiently smaller than ε. In this proof, we usually assume that $n = \mathsf{poly}(m)$ and it holds that $n = n(m) = m^\theta$. In some cases depending on the context, m is defined w.r.t. n and it holds that $m(n) = \lfloor n^{1/\theta} \rfloor$ (and we can think of m as being sublinear in n).

Constructing the PRG. Let g be the k-reconstructive PRG obtained from Theorem 3.11, and let R, M be the algorithms associated with g (as in Definition 3.10). We will consider a function $G : 1^m \times \{0,1\}^{m^\theta} \times \{0,1\}^d \to \{0,1\}^m$. On input $(1^m, x, y)$ where $x \in \{0,1\}^{m^\theta}, y \in \{0,1\}^d$, the algorithm G proceeds in the following steps.

- G first computes $z = f(x)$. Let $n = m^\theta = |z|$.
- G outputs

$$G(1^m, x, y) = g^z(1^n, 1^m, y)$$

Note that the seed length of the PRG $d(n, m) = O(\log^2 n / \log m)$ so we can let σ be a constant such that $d = \sigma \log m$ and G is now a function of the form $1^m \times \{0,1\}^{m^\theta} \times \{0,1\}^{\sigma \log m} \to \{0,1\}^m$.

We claim that G is a $O(m)$-secure $(m^\theta, \sigma \log m)$-targeted PRG. Suppose not; then there exists a $O(m)$-time deterministic distinguisher D such that for infinitely many $m \in \mathbb{N}$, $n = m^\theta$, $x \in \{0,1\}^n$,

$$\left|\Pr[v \leftarrow \{0,1\}^{\sigma \log m} : D(1^m, x, G(1^m, x, v)) = 1] - \Pr[w \leftarrow \{0,1\}^m : D(1^m, x, w) = 1]\right| \geq \frac{1}{6} \quad (4)$$

(Note that D runs in time $O(m)$ so it is unable to read the whole string x.) We will prove that f can be computed locally in n^ε time with n^c-time computable leakage, for some constant c which we will fix later.

Computing f with leakage. We will construct a n^c-time algorithm leak, and a n^ε-time (local) algorithm A, where $\mathsf{leak}(x, f(x))$ will produce a $|x|^\varepsilon$-bit leakage and $A(x, \mathsf{leak}(x, f(x))$ will locally compute the function $f(x)$ on input x for infinitely many x (i.e., those inputs x on which Equation 4 holds). The algorithms A and leak will collaboratively proceed as follows. On input x, z, leak computes $n = |x|$ and $m = \lfloor n^{1/\theta} \rfloor$, and leak simply outputs $M^{z, D(1^m, x, \cdot)}(1^n, 1^m)$. We turn to constructing the algorithm A. On input x, the output of leak (denoted by a), and a bit index $i \in [n]$, A simply outputs $R^{D(1^m, x, \cdot)}(a, i)$.

Analyzing the reduction. We turn to analyzing the reduction. We first show that $A(x, \mathsf{leak}(x, f(x)), i)$ will indeed compute $f(x)_i$ for all $i \in [|x|]$ on infinitely many inputs x. This follows from the correctness of the distinguisher D, and the security of the reconstructive PRG g. In more detail, let us fix a (sufficiently long) string $x \in \{0,1\}^n$ w.r.t. which Equation 4 holds. Note that the distinguisher $D(1^m, x, \cdot)$ will also distinguish the reconstructive PRG $g^z(1^n, 1^m, \cdot)$, and therefore $A(x, \mathsf{leak}(x, f(x)), i)$ will output

$$R^{D(1^m, x, \cdot)}(M^{z, D(1^m, x, \cdot)}(1^n, 1^m), i)$$

which equals $z_i = f(x)_i$ with probability at least $2/3$. In addition, $\mathsf{leak}(x, f(x))$ is short and of length at most n^ε (due to our choice of θ). Since $\mathsf{leak}(x, f(x))$ contains the reduction advice for the reconstructive PRG g, and by Theorem 3.11 it is at most of length $\mathsf{poly}(m, \log n) = \mathsf{poly}(n^{1/\theta}, \log n)$, which is at most n^ε (since θ is picked to be much larger than $1/\varepsilon$).

We proceed to showing that leak runs in time n^c (for some sufficiently large universal constant c) and A runs in time n^ε. Note that leak simply invokes the algorithm M on input $1^n, 1^m$ (given z and $D(1^m, x, \cdot)$), M runs in polynomial time, and D runs in time $O(m)$. It follows that leak runs in time $\mathsf{poly}(n)$ and we can pick c to be large enough such that leak runs in time n^c. A will call the algorithm R on input (a, i), which (as argued above) is of length at most $n^{2/\theta}$. Since R runs in polynomial time, A runs in time $\mathsf{poly}(n^{2/\theta})$ which will be at most n^ε if we pick θ much larger than $1/\varepsilon$.

It remains to show that G runs in time $t(m^\theta)\mathsf{poly}(m)$. Note that it takes $t(m^\theta)$ time to compute $z = f(x)$, and the construction g runs in time polynomial in $n = m^\theta$. Thus, it follows that $G(1^m, x, \cdot)$ runs in time $t(m^\theta)\mathsf{poly}(m)$. ■

4 Derandomization and Average Case Leakage-resilient Hardness

We turn to showing that the connection between derandomization of prBPP and the existence of a leakage-resilient hard function established in the previous section also holds in the average case, where

our focus lies on derandomization of prBPP on inputs produced by some probabilistic polynomial-time sampler (or more generally, by probabilistic T-time sampler for some super-polynomial T). We prove that average-case derandomization is characterized by an average-case notion of leakage-resilient hardness. Informally, a function f is said to be average-case leakage-resilient hard with respect to some distribution $\mathcal{D}$ if the hardness condition holds on inputs sampled from $\mathcal{D}$.

Definition 4.1 (Average-case Leakage-resilient Hardness)**.** *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a (multi-output) function. We say that f is α-average-case (T, ℓ)-leakage resilient hard on an ensemble $\mathcal{D} = \{D_n\}_{n\in\mathbb{N}}$ if for all T-time probabilistic algorithms leak, A satisfying $\mathsf{leak}(x, f(x)) \leq \ell(|x|)$, for all sufficiently large $n \in \mathbb{N}$, it holds that*

$$\Pr[x \leftarrow D_n : A(x, \mathsf{leak}(x, f(x))) \text{ computes } f(x)] \leq 1 - \alpha(n)$$

where for any string $x \in \{0,1\}^*$, we say that $A(x, \mathsf{leak}(x, f(x)))$ computes $f(x)$ if $A(x, \mathsf{leak}(x, f(x))) = f(x)$ with probability at least $\frac{1}{3}$ (over internal randomness of A and leak).

We say that f is simply average-case (T, ℓ)-leakage resilient hard on $\mathcal{D}$ *if it is $(1 - 1/p)$-average-case (T, ℓ)-leakage resilient hard on $\mathcal{D}$ for all polynomials p.*

We then discuss what average-case derandomization of some promise problem means. Roughly speaking, we require that no probabilistic poly-time sampler (which is referred to as a *refuter* in [Kab01]) can find an "error" in the derandomization, where an "error" is an instance/input that lies in the *symmetric difference* between the two promise problems (where one is the promise problem we aim to derandomize, the other one is the promise problem we obtain after derandomization). We now recall the notion of symmetric difference. For any two sets S_1, S_2, let $\nabla(S_1, S_2)$ denote $(S_1 \backslash S_2) \cup (S_2 \backslash S_1)$. Let $\Pi_1 = (\Pi_{1,\mathsf{YES}}, \Pi_{1,\mathsf{NO}})$ and $\Pi_2 = (\Pi_{2,\mathsf{YES}}, \Pi_{2,\mathsf{NO}})$ be two promise problems. We let $\nabla(\Pi_1, \Pi_2)$ denote the symmetric difference between Π_1 and Π_2 where $\nabla(\Pi_1, \Pi_2) = \nabla(\Pi_{1,\mathsf{YES}}, \Pi_{2,\mathsf{YES}}) \cup \nabla(\Pi_{1,\mathsf{NO}}, \Pi_{2,\mathsf{NO}})$.

We proceed to introducing the notion of *effective containment*, which will be used to (formally) capture the notion of average-case derandomization of some class of promise problems.

Definition 4.2 (Effective Containment [Gol11a])**.** *Let $\mathcal{D}_1, \mathcal{D}_2$ be two classes of promise problems. We say that $\mathcal{D}_1$* is effectively contained in $\mathcal{D}_2$ (denoted by $\mathcal{D}_1 \subseteq_{\mathsf{poly}} \mathcal{D}_2$) *if for all promise problems $\Pi_1 \in \mathcal{D}_1$, there exists a promise problem $\Pi_2 \in \mathcal{D}_2$, such that for all polynomials p, all polynomial-time probabilistic samplers F, and for all sufficiently large n,*

$$\Pr[x \leftarrow F(1^n) : x \in \nabla(\Pi_1, \Pi_2) \cap \{0,1\}^n] \leq \frac{1}{p(n)}$$

As mentioned before, we remark that the above notion slightly differs from what Goldreich formally defined in [Gol11a], but in our eyes, the notion we consider is more suitable for considering "derandomization" from a practical perspective[5]

We are now ready to present the result formally.

Theorem 4.3. *There exists a constant c such that for every $0 < \epsilon < 1$, the following are equivalent:*

- $\mathsf{prBPP} \subseteq_{\mathsf{poly}} \mathsf{prP}$.

[5]Goldreich defines the notion of effective containment with respect to some polynomial p and he characterizes derandomization of prBPP where for each promise problem $\Pi_1 \in \mathsf{prBPP}$ and each polynomial p, there exists a derandomized promise problem Π_2 such that no p-time algorithm can tell apart Π_1 and Π_2 with probability $1/p$. In other words, the problem Π_2 may be dependent on the running time of the "attacker/sampler". In contrast, the notion we consider has a different order of quantifiers where for each promise problem $\Pi_1 \in \mathsf{prBPP}$, we obtain a single promise problem Π_2 such that no poly-time algorithm can only find a "mistakes" with noticeable probability.

- *The existence of a multi-output function $f : \{0,1\}^n \to \{0,1\}^n$ computable in deterministic polynomial time that is average-case (n^c, n^ϵ)-leakage resilient hard on all samplable distributions.*

Proof: The implication (1) $\Rightarrow$ (2) follows from Theorem 4.5 (stated and proved below). To show (2) $\Rightarrow$ (1), we apply Lemma 4.9 (stated and proved below) to obtain a targeted PRG and (1) follows from Lemma 4.8. ■

Remark 4.4. *We remark that Theorem 4.3 can be generalized to the low-end setting. For example, to effectively derandomize* prBPP *into* prSUBEXP*, we need to consider a stronger notion of average-case leakage-resilient hardness and effective containment where the finder/refuter is allowed to run in subexponential time. (Note that we also allow the hard function to be computable in subexponential time (as opposed to just polynomial time) so the actual assumption seems incomparable.) See more details in Remark 4.6 after Theorem 4.5. (On the other hand, the rest of Lemmas we need in the above proof (Lemma 4.9 and Lemma 4.8) directly extend to the low-end setting with the stronger notion of effective containment and average-case hardness.)*

Towards proving Theorem 4.3, we start by showing that average-case leakage-resilient hard function can be obtained from "effective" derandomization of prBPP, which proves one implication in Theorem 4.3.

Theorem 4.5. *Let t be some polynomial. If $\mathsf{prBPTIME}[O(n)] \subseteq_{\mathsf{poly}} \mathsf{prDTIME}[t(n)]$, then for any constant $c \geq 1$, there exists a function $f : \{0,1\}^n \to \{0,1\}^n$ running in time $t(p(n))q(n)$ (for some polynomials p, q) such that f is average-case $(n^c, n - 3\log n)$-leakage resilient hard on all samplable distributions.*

Proof: This theorem can be viewed as an average-case analog of Theorem 3.8, so we will borrow many notations from Theorem 3.8. For any constant $c \geq 1$, let $(R_{\mathsf{YES}}, R_{\mathsf{NO}})$ be the prBPP search problem, R be the binary relation as in the proof of Theorem 3.8. Let Π_1 denote the decisional problem $\in$ prBPP that the search problem of R can be reduced to (see more details in Theorem 3.8). Let $M(x)$ be the deterministic decision-to-search reduction (for R) such that $M(x)^{\Pi_1}$ will output a R-witness of x.

Recall that the leakage resilient hard function obtained in Theorem 3.8 runs the reduction M and answers its oracle queries by using a (perfect) derandomization of Π_1. Since a perfect derandomization is no longer available in this proof, we will instead assist M with an effective derandomization of Π_1. To obtain the effective derandomization, we first pad instances in Π_1 such that the probabilistic algorithm for Π_1 now runs in linear time in the length of padded instances, and let $\Pi_{1,\mathsf{padded}}$ denote the padded version. Since $\Pi_{1,\mathsf{padded}} \in \mathsf{prBPTIME}[O(n)]$, there exists a promise problem $\Pi_{2,\mathsf{padded}} \in \mathsf{prDTIME}[t]$ such that no one can tell apart $\Pi_{1,\mathsf{padded}}$ and $\Pi_{2,\mathsf{padded}}$. We will use $\Pi_{2,\mathsf{padded}}$ to construct our function f. On input x, $f(x)$ runs the reduction $M(x)$ with the help of $\Pi_{2,\mathsf{padded}}$ (by sending the padded oracle queries to $\Pi_{2,\mathsf{padded}}$) and outputs the output of the reduction. Note that f is deterministic since M (and $\Pi_{2,\mathsf{padded}}$) is, and the running time of f follows from the same analysis as in Theorem 3.8.

We next show that f is average-case $(n^c, n - 3\log n)$-leakage resilient hard on all samplable distributions. Assume for contradiction that there exist a polynomial p', a PPT sampler S, n^c-time algorithms g, leak, such that with probability $1/p'(n)$ (over the internal randomness of S), $S(1^n)$ finds an input x on which $g(x, \mathsf{leak}(f(x), x))$ computes $f(x)$, for infinitely many $n \in \mathbb{N}$. We will construct a PPT refuter F (using S) to tell apart $\Pi_{1,\mathsf{padded}}$ and $\Pi_{2,\mathsf{padded}}$ (which is a contradiction). On input $1^{n'}$ (the padded input length), our refuter $F(1^{n'})$ runs $S(1^n)$ to obtain an input x (where n is the unpadded input length and n can be computed from n'). Consider some sufficiently large n and x such that $g(x, \mathsf{leak}(f(x), x))$ computes $f(x)$ (so such x can be sampled by S with probability

at least $1/p'(n)$). It follows that $(x, f(x)) \in R_{\mathsf{NO}}$ and $(x, f(x)) \notin R$. Since M is a good reduction, there exists at least one oracle query of $M(x)$ that is answered incorrectly (by $\Pi_{2,\mathsf{padded}}$). Since $M(x)$ makes at most polynomially many queries, the refuter F can guess randomly the index of the first (incorrectly answered) query $i \in [q'(n)]$ (for some poly q'), and output the i-the query of $M(x)$ (by feeding the answers to the prior $(i-1)$ queries to $M(x)$ using $\Pi_{2,\mathsf{padded}}$). Therefore, $F(1^{n'})$ will output an instance in the symmetric difference between $\Pi_{1,\mathsf{padded}}$ and $\Pi_{2,\mathsf{padded}}$ with probability $\frac{1}{p'(n)q'(n)} \geq \frac{1}{p'(n')q'(n')}$ and F runs in time $p'(n) + q'(n) \cdot t(n') \leq p'(n') + q'(n') \cdot t(n')$, for infinitely many n'. This contradicts to the assumption and we conclude that f is average-case leakage-resilient hard. ■

Remark 4.6. *We observe that the sampler F we construct in the above proof runs in time $p' + q' \cdot t$. When considering the low-end derandomization of* prBPP*, it only holds that* prBPTIME$[O(n)]$ *is effectively contained in* prDTIME$[t(n)]$ *for some super-polynomial t. Therefore, we will require a stronger notion of average-case leakage-resilient hardness and a stronger notion of effective containment (where the refuter is also allowed to run in super-polynomial time).*

We proceed to proving the other implication in Theorem 4.3, which asserts that an average-case leakage-resilient hard function can be employed to derandomize prBPP effectively. Towards this, we introduce the notion of an average-case targeted PRG, which is an average-case analog of a targeted PRG where the security is only required to hold on targets sampled from some samplable distribution.

Definition 4.7 (Average-case Targeted pseudorandom generator)**.** *Let $G : 1^n \times \{0,1\}^{\ell(n)} \times \{0,1\}^{m(n)} \to \{0,1\}^n$ be a computable function. We say that G is an $T(n)$-*secure *$\alpha(n)$-*average-case *$(\ell(n), m(n))$-*targeted pseudorandom generator on ensemble $\mathcal{D} = \{D_n\}$ of targets (T-secure α-average-case $(\ell(n), m(n))$-targeted PRG on $\mathcal{D}$) *if for all deterministic attackers A that run in $T(n)$ time (where n is the length of its first input), for all sufficiently large $n \in \mathbb{N}$, it holds that*

$$\Pr[x \leftarrow D_n : G \text{ is secure on } x \text{ w.r.t. } A] \geq \alpha(n)$$

where for all $x \in \{0,1\}^n$, we say that G is secure on *x* w.r.t. *A if*

$$|\Pr[s \leftarrow \{0,1\}^{m(n)} : A(1^n, x, G(1^n, x, s)) = 1] - \Pr[y \leftarrow \{0,1\}^n : A(1^n, x, y) = 1]| < \frac{1}{6}.$$

*We say that G is simply T-*secure average-case $(\ell(n), m(n))$-targeted PRG on $\mathcal{D}$ *if it is $(1-1/p)$-average-case targeted on $\mathcal{D}$ for all polynomials p.*

Analogous to the almost-all-input setting (where we showed that a targeted PRG is sufficient to derandomize prBPP), we show that the existence of an average-case targeted PRG implies an effective derandomization of prBPP.

Lemma 4.8. *Assume that there exist constants $\sigma \geq 1, \theta \geq 1$ and a $O(n)$-secure average-case $(n^\theta, \sigma \log n)$-targeted PRG G running in time $t(n) \geq n$ that is average-case targeted on all samplable distributions. Then,* prBPP *is effectively contained in* $\cup_{p,q \in \mathsf{poly}}$prDTIME$[t(p(n))q(n)]$.

Proof: This lemma follows from the proof of Lemma 2.6 since the security reduction in Lemma 2.6 is one-to-one with respect to instances/targets. The reduction shows that an instance on which the derandomization fails[6] is also a target (up to some padding) on which the targeted PRG G is broken, and this mapping is one-to-one. The lemma follows since the padding argument only pads

[6]Note that we can guarantee that an instance in the symmetry difference is also an instance on which the derandomization fails by restricting the derandomized promise problem within the promise of the original problem.

an instance to a polynomially longer target string (so the same sampler will still succeed with high probability with respect to the length of the padded target string). ■

Finally, we show that a targeted PRG can be obtained from an average-case leakage-resilient hard function, and (as consequence) the targeted PRG will be only average-case targeted. This completes the characterization between effective derandomization and leakage-resilient hardness in the average-case.

Lemma 4.9. *There exists a constant $c \geq 2$ such that the following holds. Assume that there exist a constant $\varepsilon > 0$ and a function $f : \{0,1\}^n \to \{0,1\}^n$ computable in deterministic time $t(n) \geq n$ that is average-case (n^c, n^ε)-leakage resilient hard on all samplable distributions. Then there exist constants $\sigma, \theta \geq 1$ and a $O(n)$-secure average-case $(n^\theta, \sigma \log n)$-targeted PRG computable in time $t(n^\theta)\mathsf{poly}(n)$ that is average-case targeted on all samplable distributions.*

Proof: This lemma will also follow from the proof of Lemma 3.9 since the security reduction in Lemma 3.9 is one-to-one with respect to instances/targets. The reduction shows that a target on which the targeted PRG G is broken is also an input on which the function f can be computed with some leakage, and this mapping is one-to-one. Note that Lemma 3.9 requires a local hard function, and a (n^c, n^ε)-leakage resilient hard function is trivially a (n^c, n^ε)-leakage resilient n^ε-locally hard function if $c \geq 2$. Thus, the lemma follows. ■

5 Characterizing Derandomization of prMA

In this section, we present a characterization between derandomization and leakage-resilient hardness regarding prMA and prNP. In the non-deterministic setting, we need to consider a notion of leakage-resilient hard relations (generalizing the notion of leakage-resilient hard functions), which will be both sufficient and necessary to derandomize prMA.

For any relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$ and any class of languages $\mathcal{C}$, we say that R is computable in $\mathcal{C}$ if there exists a language $L \in \mathcal{C}$ such that $(x,y) \in R$ iff $(x,y) \in L$.

Definition 5.1 (Leakage Resilient Hard Relation)**.** *Let $R \subseteq \{0,1\}^* \times \{0,1\}^*$ be a relation such that for every $(x,y) \in R$, $|x| = |y|$. We say that R is* almost-all-input $(T(\cdot), \ell(\cdot))$-leakage resilient hard *if the following two conditions hold.*

- *(Non-triviality.) For every $x \in \{0,1\}^*$, $R(x) = \{y : (x,y) \in R\} \neq \emptyset$.*
- *(Leakage-resilient.) For all T-time probabilistic algorithms leak, A satisfying $\mathsf{leak}(x,y) \leq \ell(|x|)$, for all sufficiently long strings x, y satisfying $(x,y) \in R$, $A(x, \mathsf{leak}(x,y)) \neq y$ with probability $\geq 2/3$ (over their internal randomness).*

Now we are ready to state our characterization of derandomizing prMA.

Theorem 5.2. *There exists a constant $c \geq 1$ such that for all nice classes of functions $\mathcal{C}$, all constants $0 < \varepsilon < 1$, the following are equivalent.*

1. $\mathsf{prMA} \subseteq \cup_{T \in \mathcal{C}} \mathsf{prNTIME}[T]$.
2. *The existence of a function $T \in \mathcal{C}$ and an almost-all-input (n^c, n^ε)-leakage resilient hard relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$ computable in* $\mathsf{NTIME}[T]$.

Proof: The implication (1) $\Rightarrow$ (2) follows from Theorem 5.3 (stated and proved below). To show (2) $\Rightarrow$ (1), we apply Lemma 5.6 (stated and proved in below) to obtain a targeted PRG and (1) follows from Lemma 5.5. ■

As demonstrated in the proof above, the proof Theorem 5.2 contains two parts. In the first part, we show that we can directly obtain a leakage-resilient hard relation from the assumption that $\mathsf{prMA} = \mathsf{prNP}$ without relying on a search-to-decision reduction.

Theorem 5.3. *If* $\mathsf{prMATIME}[O(n)] \subseteq \mathsf{prNTIME}[t(n)]$*, then for any constant* $c \geq 1$*, there exists a relation* R *computable in* $\mathsf{NTIME}[t(p(n))]$ *(for some polynomials* p*) that is almost-all-input* $(n^c, n - 3\log n)$*-leakage resilient hard.*

Proof: For any constant $c \geq 1$, let $(R_{\mathsf{YES}}, R_{\mathsf{NO}})$ be the prBPP-search problem defined in the proof of Theorem 3.8. Note that the promise problem $\Pi = (R_{\mathsf{YES}}, R_{\mathsf{NO}})$ is in prBPP, and therefore $\Pi \in \mathsf{prMA}$. Since $\mathsf{prMATIME}[O(n)] \subseteq \mathsf{prNTIME}[t(n)]$, by padding instances in Π so that the probabilistic verifier for Π now runs in linear time in the length of the padded instances, it follows that $\Pi \in \mathsf{prNTIME}[t(p(n))]$ (for some polynomial p). Let V be the deterministic verifier for Π and let L be the language associated with V. Let R be a relation such that $(x, y) \in R$ iff $(x, y) \in L$ and $|x| = |y|$. It follows that R can be computed in $\mathsf{NTIME}[t(p(n))]$.

We then show that R is almost-all-input $(n^c, n - 3\log n)$-leakage resilient hard. Consider any probabilistic n^c-time algorithms leak and g satisfying $\mathsf{leak}(x, y) \leq |x| - 3\log|x|$, all sufficiently large $n \in \mathbb{N}$ such that $|g| \leq \log n$ and $|\mathsf{leak}| \leq \log n$, all strings $x, y \in \{0,1\}^n$ such that $(x, y) \in R$. Since $(x, y) \in R$, $(x, y) \notin R_{\mathsf{NO}}$. It follows that $g(x, \mathsf{leak}(x, y)) \neq y$ with probability $2/3$. This shows that R is almost-all-input leakage-resilient hard and concludes the proof. ■

In the second part of the proof, we prove the converse implication of Theorem 5.3. We rely on the following non-deterministic variant of a targeted PRG (where the PRG takes as input an additional witness whose validity can be checked by an verifier).

Definition 5.4 (Targeted non-deterministic pseudorandom generator)**.** *Let* $G : 1^n \times \{0,1\}^{\ell(n)} \times \{0,1\}^{\ell(n)} \times \{0,1\}^{m(n)} \to \{0,1\}^n$ *be a computable function. We say that* G *is an* $T(n)$-secure $(\ell(n), m(n))$-targeted non-deterministic pseudorandom generator (T-secure $(\ell(n), m(n))$-targeted NPRG) *if there exists a non-deterministic verifier* V *such that the following two conditions hold:*

- *For all sufficiently large* $n \in \mathbb{N}$*, for all* $x \in \{0,1\}^{\ell(n)}$*, there exists* $w \in \{0,1\}^{\ell(n)}$*,* $|w| = |x|$*, and* $V(1^n, x, w) = 1$*.*
- *For all deterministic attackers* D *that run in* $T(n)$ *time (where* n *is the length of its first input), for all sufficiently large* $n \in \mathbb{N}$ *and all strings* $x, w \in \{0,1\}^{\ell(n)}$ *satisfying* $V(1^n, x, w) = 1$*, it holds that*

$$|\Pr[s \leftarrow \{0,1\}^{m(n)} : D(1^n, x, G(1^n, x, w, s)) = 1] - \Pr[y \leftarrow \{0,1\}^n : D(1^n, x, y) = 1]| < \frac{1}{6}.$$

We say that a targeted NPRG G is computable in time T (for some function T) if G is computable in time T (with respect to the length of its first input) and there exists a verifier for G computable in non-deterministic time T (w.r.t. the length of its first input).

We then show that the notion of a targeted NPRG is indeed useful by proving that it can be used to derandomize prMA.

Lemma 5.5. *Assume that there exist constants* $\sigma \geq 1, \theta \geq 1$ *and a* $O(n)$*-secure* $(n^\theta, \sigma \log n)$*-targeted NPRG* G *that is computable in time* $t(n) \geq n$*. Then,* $\mathsf{prMA} \subseteq \cup_{p,q \in \mathsf{poly}} \mathsf{prNTIME}[t(p(n))q(n)]$.

Proof: To show that $\mathsf{prMA} \subseteq \cup_{p,q \in \mathsf{poly}} \mathsf{prNTIME}[t(p(n))q(n)]$, it suffices to prove that $\mathsf{prMATIME}[O(n)] \subseteq \cup_{p,q \in \mathsf{poly}} \mathsf{prNTIME}[t(O(n))\mathsf{poly}(n)]$ (and by a standard padding argument in which we pad instances in $\mathsf{prMATIME}[p(n)]$ to length $p(n)$, the claim follows). We will show that for any linear-time randomized verifier V (that uses as many random coins as its input length and also takes a witness

of the same length), there exists a non-deterministic $t(n)\mathsf{poly}(n)$-time machine N such that for all sufficiently large $n \in \mathbb{N}$, $x \in \{0,1\}^n$, if there exists $w \in \{0,1\}^n$ such that $\Pr_r[V(x,w;r)=1] \geq \frac{2}{3}$, then $N(x)$ accepts; and if for all $w \in \{0,1\}^n$, $\Pr_r[V(x,w;r)=0] \geq \frac{2}{3}$, $N(x)$ rejects (where r denotes the random coins of V).

If $\theta > 1$, we need to consider the following padded version of V (to obtain a long enough target string for the targeted NPRG). For any $x, w \in \{0,1\}^*$ such that $|x| = |w|$, let $\mathsf{pad}(x,w) = xw0^{(2|x|)^\theta - 2|x|}$; that is, we pad as many '0' at the end of $x||w$ until it becomes of length $(2|x|)^\theta$. Let V' be an algorithm such that $V'(1^{2|x|}, \mathsf{pad}(x,w); r) = V(x,w;r)$ for any x, w, r. (In more detail, V' on input $(1^m, y; r)$, splits the m-bit prefix of y into two parts of equal length to obtain x, w, and simply returns $V(x,w;r)$.) If $\theta = 1$, we define pad and V' in the natural way.

We proceed to describing the non-deterministic machine N. Let V_{NPRG} be the verifier of G. On input $x \in \{0,1\}^n$, $N(x)$ guesses a witness w and a NPRG witness w_{NPRG}. $N(x)$ then enumerates all possible seeds v of the targeted NPRG G. Finally, $N(x)$ accepts if $V_{\mathsf{NPRG}}(1^{2n}, \mathsf{pad}(x,w), w_{\mathsf{NPRG}}) = 1$ and the majority of $V'(1^{2n}, \mathsf{pad}(x,w); G(1^{2n}, \mathsf{pad}(x,w), w_{\mathsf{NPRG}}, v))$ among all possible seeds v equals 1. Note that $N(x)$ invokes V_{NPRG} once, and invokes G and V' for $2^{|v|} = \mathsf{poly}(n)$ times. Thus, $N(x)$ runs in time $t(2n)\mathsf{poly}(n)$.

Finally, we prove the correctness of N. Since G is a $O(m)$-secure $(m^\theta, \sigma \log m)$-targeted NPRG G and V' runs in deterministic time $O(|x|)$, it follows that G is secure with respect to V'; that is, for all sufficiently large $n \in \mathbb{N}$, all $x, w \in \{0,1\}^n$, all w_{NPRG} such that $V_{\mathsf{NPRG}}(1^{2n}, \mathsf{pad}(x,w), w_{\mathsf{NPRG}}) = 1$, it holds that

$$\Big|\Pr[r \leftarrow \{0,1\}^{2n} : V'(1^{2n}, \mathsf{pad}(x,w); r) = 1] - \Pr[v \leftarrow \{0,1\}^{\sigma \log(2n)} : V'(1^{2n}, \mathsf{pad}(x,w); G(1^{2n}, \mathsf{pad}(x,w), w_{\mathsf{NPRG}}, v)) = 1]\Big| < \frac{1}{6}$$

Consider any sufficiently large $n \in \mathbb{N}$ and $x \in \{0,1\}^n$. If there exists a witness $w \in \{0,1\}^n$ such that $\Pr_r[V(x,w;r)=1] \geq \frac{2}{3}$, it follows that $V'(1^{2n}, \mathsf{pad}(x,w); r) = 1$ with probability $\frac{2}{3}$ over r. Since V_{NPRG} is the verifier of G, there exists a NPRG witness w_{NPRG} such that $V_{\mathsf{NPRG}}(1^{2n}, \mathsf{pad}(x,w), w_{\mathsf{NPRG}}) = 1$. Given w and w_{NPRG}, the majority bit computed in $N(x)$ will be 1 (due to the security of G) and $N(x)$ will accept. If for all $w \in \{0,1\}^n$, $\Pr_r[V(x,w;r)=0] \geq \frac{2}{3}$. Consider any NPRG witness w_{NPRG}. If V_{NPRG} accepts w_{NPRG}, the majority bit $N(x)$ obtains will be 0. Therefore, $N(x)$ will reject. ■

It remains to show that the existence of a leakage-resilient hard relation implies the existence of a targeted NPRG, which can be proved by generalizing Lemma 3.9 to allow non-deterministic computation.

Lemma 5.6. *There exists a constant $c \geq 1$ such that the following holds. Assume that there exist a constant $\varepsilon > 0$ and an almost-all-input (n^c, n^ε)-leakage resilient hard relation R computable in $\mathsf{NTIME}[t]$. Then there exist constants $\sigma, \theta \geq 1$ and a $O(n)$-secure $(n^\theta, \sigma \log n)$-targeted NPRG G_{N} computable in time $t(n^\theta)\mathsf{poly}(n)$.*

Proof: The proof of Lemma 3.9 will also prove Lemma 5.6 with some modifications to adapt to non-determinism. For any $\varepsilon > 0$, let θ, σ be the constants, and $G : 1^m \times \{0,1\}^{m^\theta} \times \{0,1\}^{\sigma \log m} \to \{0,1\}^m$ be the targeted PRG as in Lemma 3.9. We will replace the leakage-resilient hard function f used in the construction of G by a leakage-resilient hard relation. Let $G_{\mathsf{N}} : 1^m \times \{0,1\}^{m^\theta} \times \{0,1\}^{m^\theta} \times \{0,1\}^{\sigma \log m} \to \{0,1\}^m$ be a function defined as follows. On input $(1^m, x, w, y)$, G_{N} proceeds as G does on input $(1^m, x, y)$, except that in the first step, G_{N} lets $z = w$. (Note that G only depends on the the function f in the first step, so G_{N} can proceed to the rest steps without f.) Since R

is computable in $\mathsf{NTIME}[t]$, let N be the non-deterministic machine deciding R. Let V be a non-deterministic verifier such that $V(1^m, x, w) = N(x, w)$ where $x, w \in \{0,1\}^{m^\theta}$. We will show that G_{N} is a $O(n^\theta)$-secure $(n^\theta, \sigma \log n)$-targeted NPRG with verifier V.

Note that the function G_{N} is computable in polynomial time, and the verifier runs in non-deterministic time $t(m^\theta)$. So the targeted NPRG G_{N} is computable in time $t(m^\theta)\mathsf{poly}(m)$. Due to the non-triviality of R, it holds that for every $m \in \mathbb{N}$, $x \in \{0,1\}^{m^\theta}$, there exists a witness $w \in \{0,1\}^{m^\theta}$ such that $V(1^m, x, w) = 1$.

We proceed to arguing the security of the targeted NPRG G_{N}. Assume for contradiction that there exists a $O(m)$-time deterministic distinguisher D that breaks the targeted NPRG $G_{\mathsf{N}}(1^m, x, w, \cdot)$ on infinitely many $m \in \mathbb{N}$, $n = m^\theta$, $x, w \in \{0,1\}^n$ satisfying $V(1^m, x, w) = 1$. We refer to such (x, w) as being good. Let leak, A be the attackers constructed in Lemma 3.9. Note that the attacker leak run in probabilistic time n^c, $\mathsf{leak}(x, w) \leq |x|^\varepsilon$. The attacker A runs (locally) in n^ε time, and therefore A runs in $n \cdot n^\varepsilon$ to produce every bit in the output. The security proof in Lemma 3.9 will also show that $A(x, \mathsf{leak}(x, w))$ computes w (with probability $2/3$) on such good x, w. Since $V(1^m, x, w) = 1$, it follows that $(x, w) \in R$, which contradicts to the leakage-resilient hardness of R. This concludes the proof. ■

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[AGV09] Adi Akavia, Shafi Goldwasser, and Vinod Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *Theory of cryptography conference*, pages 474–495. Springer, 2009.

[BFNW93] László Babai, Lance Fortnow, Noam Nisan, and Avi Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3:307–318, 1993.

[BK12] Zvika Brakerski and Yael Tauman Kalai. A parallel repetition theorem for leakage resilience. In *Theory of Cryptography Conference*, pages 248–265. Springer, 2012.

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

[Cop97] Don Coppersmith. Small solutions to polynomial equations, and low exponent rsa vulnerabilities. *Journal of cryptology*, 10(4):233–260, 1997.

[CRTY20] Lijie Chen, Ron D Rothblum, Roei Tell, and Eylon Yogev. On exponential-time hypotheses, derandomization, and circuit lower bounds. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 13–23. IEEE, 2020.

[CT21] Lijie Chen and Roei Tell. Hardness vs randomness, revised: Uniform, non-black-box, and instance-wise. *Electronic Colloquium on Computational Complexity*, 2021. https://eccc.weizmann.ac.il/report/2021/080/l.

[DP08] Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302, 2008.

[Gol11a] Oded Goldreich. In a world of p= bpp. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, pages 191–232. Springer, 2011.

[Gol11b] Oded Goldreich. Two comments on targeted canonical derandomizers. In *Electron. Colloquium Comput. Complex.*, volume 18, page 47, 2011.

[Hir20] Shuichi Hirahara. Non-disjoint promise problems from meta-computational view of pseudorandom generator constructions. In *35th Computational Complexity Conference (CCC 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[IKW02] Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. In search of an easy witness: Exponential time vs. probabilistic polynomial time. *Journal of Computer and System Sciences*, 65(4):672–694, 2002.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *Annual International Cryptology Conference*, pages 463–481. Springer, 2003.

[IW97] Russell Impagliazzo and Avi Wigderson. $P = BPP$ if e requires exponential circuits: Derandomizing the xor lemma. In *STOC '97*, pages 220–229, 1997.

[IW98] R Impagliazzo and A Wigderson. Randomness vs. time: de-randomization under a uniform assumption. In *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No. 98CB36280)*, pages 734–743. IEEE, 1998.

[Kab01] Valentine Kabanets. Easiness assumptions and hardness tests: Trading time for zero error. *Journal of Computer and System Sciences*, 63(2):236–252, 2001.

[Kor22] Oliver Korten. Derandomization from time-space tradeoffs. In *37th Computational Complexity Conference (CCC 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

[LP22] Yanyi Liu and Rafael Pass. Characterizing derandomization through hardness of levin-kolmogorov complexity. In CCC, 2022.

[Mau92] Ueli M Maurer. Factoring with an oracle. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 429–436. Springer, 1992.

[MR04] Silvio Micali and Leonid Reyzin. Physically observable cryptography. In *Theory of Cryptography Conference*, pages 278–296. Springer, 2004.

[MW18] Cody Murray and Ryan Williams. Circuit lower bounds for nondeterministic quasi-polytime: an easy witness lemma for np and nqp. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 890–901, 2018.

[Nis91] Noam Nisan. Pseudorandom bits for constant depth circuits. *Combinatorica*, 11(1):63–70, 1991.

[NW94] Noam Nisan and Avi Wigderson. Hardness vs randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, 1994.

[Pas20] Rafael Pass. Unprovability of leakage-resilient cryptography beyond the information-theoretic limit. In Clemente Galdi and Vladimir Kolesnikov, editors, *Security and Cryptography for Networks - 12th International Conference, SCN 2020, Amalfi, Italy, September 14-16, 2020, Proceedings*, volume 12238 of *Lecture Notes in Computer Science*, pages 621–642. Springer, 2020.

[RS85] Ronald L Rivest and Adi Shamir. Efficient factoring based on partial information. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 31–34. Springer, 1985.

[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.

[Tel19] Roei Tell. Proving that prbpp= prp is as hard as proving that "almost np" is not contained in p/poly. *Information Processing Letters*, 152:105841, 2019.

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

[Yao82] Andrew Chi-Chih Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 80–91, 1982.

A Proof of Lemma 2.6

Proof: To show that $\mathsf{prBPP} \subseteq \cup_{p,q\in\mathsf{poly}}\mathsf{prDTIME}[t(p(n))q(n)]$, it suffices to prove that $\mathsf{prBPTIME}[O(n)] \subseteq \mathsf{prDTIME}[t(O(n))]\mathsf{poly}(n)$ (and by a standard padding argument in which we pad instances in $\mathsf{prBPTIME}[p(n)]$ languages to length $p(n)$, the claim follows). We will show that for any linear-time randomized algorithm A (that uses as many random coins as its input length), there exists a $t(n)\mathsf{poly}(n)$ time deterministic algorithm B such that for all sufficiently long $x \in \{0,1\}^*$, if $\Pr_r[A(x;r) = 1] \geq \frac{2}{3}$, then $B(x) = 1$; and if $\Pr_r[A(x;r) = 0] \geq \frac{2}{3}$, then $B(x) = 0$ (where r denotes the random coins that A uses).

If $\theta > 1$, the following padding argument is needed. For any string $x \in \{0,1\}^*$, let x' be the string $x0^{|x|^\theta - |x|}$; that is, we pad as many '0' at the end of x until it becomes of length $|x|^\theta$. Let A' be an algorithm such that $A'(1^{|x|}, x'; r) = A(x; r)$ for any x, r. [7]

We proceed to constructing a deterministic algorithm B that deterministically emulates A. On input an instance $x \in \{0,1\}^n$, $B(x)$ tries all possible seeds $v \in \{0,1\}^{\sigma \log n}$ and $B(x)$ outputs the majority of $A'(1^n, x'; G(1^n, x', v)) = 1$ among all seeds v. Note that B will invoke $G(1^n, x', \cdot)$ and $\mathcal{A}'(1^n, x'; \cdot)$ for $2^{\sigma \log n} = \mathsf{poly}(n)$ times, and thus B runs in time $t(n)\mathsf{poly}(n)$.

We show that, for all sufficiently long x, $B(x)$ will derandomize $A(x, \cdot)$ correctly. Since G is a $O(n)$-secure $(n^\theta, \sigma \log n)$-targeted PRG and A' runs in deterministic $O(n)$ time with respect to $n = |r| = |x|$, it follows that for all sufficiently large $n \in \mathbb{N}$, $x' \in \{0,1\}^{n^\theta}$, it holds that

$$\left|\Pr[r \leftarrow \{0,1\}^n : A'(1^n, x'; r) = 1] - \Pr[v \leftarrow \{0,1\}^{\sigma \log n} : A'(1^n, x'; G(1^m, x', v)) = 1]\right| < \frac{1}{6} \qquad (5)$$

Consider some string x such that G is secure on auxiliary input x' (with respect to A'). By the above equation, if $\Pr_r[A(x;r) = 1] \geq \frac{2}{3}$, the majority bit B obtains will equal 1; If $\Pr_r[A(x;r) = 0] \geq \frac{2}{3}$, the majority bit B obtains will equal 0. Finally note that G will be secure on all sufficiently long x', which concludes the proof. ■

[7] More formally, A' on input $(1^n, x'; r)$, picks x to be the first n bits of x', and simply returns $A(x; r)$.

B Proof of Theorem 3.11

The proof of Theorem 3.11 relies on several important tools/results developed in the literature.

Tool 1: Locally List-decodable ECCs We start by recalling the notion of a locally list-decodable error correcting code that we will be relying on.

Definition B.1 (Locally list-decodable error correcting code (see e.g. [Vad12]))**.** *For any $n, n', L \in \mathbb{N}$ and $\delta > 0$, a function $\mathsf{Enc} : \{0,1\}^n \to \{0,1\}^{n'}$ is said to be a* locally $(L, \frac{1}{2} - \delta)$-list-decodable error correcting code *if there exists a probabilistic oracle machine $\mathsf{Dec} : [n] \times [L] \to \{0,1\}$ such that for any $x \in \{0,1\}^n$ and $x' \in \{0,1\}^{n'}$ satisfying $\Pr_{i \in [n']}[\mathsf{Enc}(x)_i \neq x'_i] \leq \frac{1}{2} - \delta$ it holds that with probability at least $2/3$ over the internal randomness r of Dec, there exists $j \in [L]$ such that*

$$\forall i \in [n], \mathsf{Dec}^{x'}(i, j; r) = x_i$$

We refer to Dec as a decoder of Enc.

The following construction of a locally list-decodable error correcting code will be useful for us.

Theorem B.2 ([STV01]; see also [Vad12, Theorem 7.61])**.** *There exist a deterministic polynomial time algorithms Enc and a probabilistic oracle algorithm Dec such that for all $n \in \mathbb{N}$, $\delta > 0$, the function $\mathsf{Enc}_{n,\delta} : \{0,1\}^n \to \{0,1\}^{2^r}$ where $r = O(\log(n/\delta))$ is a locally $(\mathsf{poly}(1/\delta), \frac{1}{2} - \delta)$-list-decodable error correcting code with $\mathsf{Dec}_{n,\delta}$ being its decoder, $\mathsf{Enc}_{n,\delta}$ runs in time $\mathsf{poly}(n, 1/\delta)$, and $\mathsf{Dec}_{n,\delta}$ runs in time $\mathsf{poly}(\log n, 1/\delta)$.*

Tool 2: The NW PRG We turn to recalling the construction of the Nisan-Wigderson (NW) PRG [NW94]. For any string $y \in \{0,1\}^d$ and subset $I \subseteq [d]$, we let y_I denote the $|I|$-bit string consisting of the the projection of y to the coordinates $\in I$.

Definition B.3 (NW generator)**.** *Let $\mathcal{I} = (I_1, \ldots, I_m)$ be a family of m subsets of $[d]$ with each $|I_j| = r$ and let $h : \{0,1\}^r \to \{0,1\}$ be a function. The $(\mathcal{I}, h)$-NW generator is the function $\mathsf{NW}^h_{\mathcal{I}} : \{0,1\}^d \to \{0,1\}^m$ that takes any string $y \in \{0,1\}^d$ as a seed and outputs*

$$\mathsf{NW}^h_{\mathcal{I}}(y) = h(y_{I_1}) \ldots h(y_{I_m})$$

The core ingredient of the Nisan-Wigderson construction is a combinatorial design which will be used as the family of subsets in a NW generator.

Definition B.4 (Combinatorial designs)**.** *For any integers $d, r, s \in \mathbb{N}$ such that $d > r > s$, a family $\mathcal{I} = \{I_1, \ldots, I_m\}$ of subsets of $[d]$ is said to be a* (d, r, s)-design *if for every $j \in [m]$, $|I_j| = r$, and for every $k \in [m], k \neq j$, $|I_j \cap I_k| \leq s$.*

Recall that combinatorial designs can be efficiently constructed.

Lemma B.5 ([NW94]; see also [Vad12, Problem 3.2])**.** *There exists a deterministic algorithm $\mathsf{GenDesign}$ such that on input $d, r, s \in \mathbb{N}$ where $r > s$ and $d = O(r^2/s)$, runs in $\mathsf{poly}(2^s, d)$ steps and outputs a (d, r, s)-design $\mathcal{I}$ containing 2^{3s} subsets of $[d]$.*

The following version of the reconstruction theorem will be useful for us.

Lemma B.6 (Implicit in [NW94, IW97])**.** *There exist a PPT algorithm $\mathsf{NWRecon}$ such that the following conditions hold.*

- *Input: the truthtable of a function* $h : \{0,1\}^r \to \{0,1\}$*, a* (d, r, s)*-design* $\mathcal{I} = \{I_1, \ldots, I_m\}$*, a parameter* $\alpha^{-1} \in \mathbb{N}$ *(presented in unary).*
- *Given oracle access to an oracle* $D \subseteq \{0,1\}^m$ *such that*

$$\left|\Pr[y \leftarrow \{0,1\}^d : D(\mathsf{NW}^h_{\mathcal{I}}(y)) = 1] - \Pr[w \leftarrow \{0,1\}^m : D(w) = 1]\right| \geq \alpha. \tag{6}$$

- *Output: a program* M *of description length* $\leq m \cdot 2^s + m + d + O(\log drsm)$ *such that*

$$\Pr[p \leftarrow [2^r] : M^D(\mathcal{I}, p) \neq h(p)] \leq \frac{1}{2} - \frac{\alpha}{2m}$$

  and for every $p \in [2^r]$*,* $M^D(\mathcal{I}, p)$ *runs in* $\mathsf{poly}(r, m, 2^s)$ *steps.*
- $\mathsf{NWRecon}(h, \mathcal{I}, \alpha)$ *runs in* $\mathsf{poly}(2^r, m, 2^s, \alpha^{-1})$ *time.*

Proof: Our proof starts with a standard hybrid argument. To remove the absolute value in Equation 6, observe that there exists a bit $b \in \{0,1\}$ such that

$$\Pr[y \leftarrow \{0,1\}^d : D(\mathsf{NW}^f_{\mathcal{I}}(y)) = b] - \Pr[w \leftarrow \{0,1\}^m : D(w) = b] \geq \alpha.$$

For every $j \in \{0, 1, \ldots, m\}$, we consider the hybrid distribution H_j defined as the following:

$$H_j = (f(y_{I_1}), \ldots, f(y_{I_j}), w_{j+1}, \ldots, w_m)$$

where $y \leftarrow \{0,1\}^d$ and each $w_k \leftarrow \{0,1\}$ (for $j+1 \leq k \leq m$). Notice that H_0 and $\mathcal{U}_m$ are identically distributed, and H_m is a distribution identical to $\mathsf{NW}^f_{\mathcal{I}}(\mathcal{U}_d)$. Therefore, it follows that

$$\frac{1}{m} \sum_{j \in [m]} \left(\Pr_{y,w}[D(H_j) = b] - \Pr_{y,w}[D(H_{j-1}) = b] \right) = \frac{1}{m} \left(\Pr_{y,w}[D(H_m) = b] - \Pr_{y,w}[D(H_0) = b] \right) \geq \frac{\alpha}{m}$$

If we think of j as a random variable distributed over $[m]$, we obtain

$$\mathsf{E}_{j \leftarrow [m], y, w} \left[\Pr[D(H_j) = b] - \Pr[D(H_{j-1}) = b] \right] = \mathsf{E}_{j \in [m]} \left[\Pr_{y,w}[D(H_j) = b] - \Pr_{y,w}[D(H_{j-1}) = b] \right] \geq \frac{\alpha}{m}$$

Observe that the value $\Pr[D(H_j) = b] - \Pr[D(H_{j-1}) = b]$ is upper bounded by 1. By an averaging argument, with probability at least $\frac{\alpha}{2m}$ over the choice of $j \leftarrow [m], y_{[d] \setminus I_j} \leftarrow \{0,1\}^{d-r}, w_{[m] \setminus [j]} \leftarrow \{0,1\}^{m-j}$, the strings $j, y_{[d] \setminus I_j}, w_{[m] \setminus [j]}$ will satisfy

$$\Pr_{y_{I_j} \leftarrow \{0,1\}^r} \left[D(H_j) = b \right] - \Pr_{y_{I_j} \leftarrow \{0,1\}^r, w_j \leftarrow \{0,1\}} \left[D(H_{j-1}) = b \right] \geq \frac{\alpha}{2m} \tag{7}$$

We refer to a choice of $j, y_{[d] \setminus I_j}, w_{[m] \setminus [j]}$ as being good if it satisfies the above condition. Note that we can find a good choice of $j, y_{[d] \setminus I_j}, w_{[m] \setminus [j]}$ with high probability ($\geq 2/3$) by drawing $O(m/\alpha)$ random samples and verifying if Equation 7 holds. Fix some good choice of $j, y_{[d] \setminus I_j}, w_{[m] \setminus [j]}$. By Yao's prediction versus indistinguishability theorem [Yao82] (see also [AB09, Theorem 10.12] and [Vad12, Proposition 7.16]), Equation 7 turns out to imply a good approximation of the function f, and it holds that

$$\Pr_{y_{I_j} \leftarrow \{0,1\}^r, w_j \leftarrow \{0,1\}} \left[D(H_{j-1}) \oplus b \oplus w_j = f(y_{I_j}) \right] \geq \frac{1}{2} + \frac{\alpha}{2m}$$

By an average argument, there exists $w_j \in \{0,1\}$ such that

$$\Pr_{y_{I_j} \leftarrow \{0,1\}^r} \left[D(H_{j-1}) \oplus b \oplus w_j = f(y_{I_j}) \right] \geq \frac{1}{2} + \frac{\alpha}{2m} \tag{8}$$

Notice that we can use Equation 8 to approximate the function f. Let M be a machine with the values d, r, s, m, the bit b, the choice of $j \in [m]$, $y_{[d]\setminus I_j} \in \{0,1\}^{d-r}$, $w_{[m]\setminus[j]} \in \{0,1\}^{m-j}$, and the bit w_j hardwired in it. M also needs to hardwire some values of f in order to compute H_{j-1}. For each $k < j$, note that $|I_k \cap I_j| \leq s$ and there are only s bits in y_{I_k} depends on y_{I_j}. Thus, we need to hardwire 2^s values of f to compute $f(y_{I_k})$. Finally, for every $p \in [2^r]$, let $y_{I_j} = p$ and $M^D(\mathcal{I}, p)$ will output

$$M^D(\mathcal{I}, p) = D(f(y_{I_1}), \ldots, f(y_{I_{j-1}}), w_j, w_{j+1}, \ldots, w_m) \oplus b \oplus w_j.$$

Note that $M^D(\mathcal{I}, p) = D(H_{j-1}) \oplus b \oplus w_j$ and thus by Equation 8, $M^D(\mathcal{I}, \cdot)$ is a good approximation of the function $f(\cdot)$. M uses $O(\log drsm)$ bits to include d, r, s, m, b, j, w_j and its code, no more than $d + m$ bits to store $y_{[d]\setminus I_j}$ and $w_{[m]\setminus[j]}$, and no more than $m2^s$ bits to save the values of f that it needs. So the description length of M is $\leq m \cdot 2^s + m + d + O(\log drsm)$. On every $p \in [2^r]$, $M^D(\mathcal{I}, p)$ runs in time $\mathsf{poly}(rm2^s)$.

Finally, note that to find the machine M, we need to pick the bit $b \in \{0,1\}$, select a good choice of $j, y_{[d]\setminus I_j}, w_{[m]\setminus[j]}$, and then pick the bit $w_j \in \{0,1\}$. As mentioned, we can select a good choice of $j, y_{[d]\setminus I_j}, w_{[m]\setminus[j]}$ by sampling and checking (which takes $O(m/\alpha) \cdot 2^r \mathsf{poly}(m2^s)$ time), and we can also use the same approach to determine b and w_j. We conclude that there exists a randomized algorithm finding M in polynomial time. ■

Returning to the proof of Theorem 3.11 We are finally ready to prove Theorem 3.11 by relying on the above two tools/results.

Proof: [of Theorem 3.11] Before presenting a formal proof, it may be helpful to first discuss the choice of parameters in our construction.

Notations. Let m denote the output length of the targeted PRG that we hope to construct. Let n denote the "input length" of oracle f, $f : [n] \to \{0,1\}$. Let $\alpha = \frac{1}{6}$ be a constant.

Constructing the PRG. Our PRG will take as input a unary string 1^n, a unary string 1^m, along with a seed y. Let $\delta = O(\frac{1}{m})$ and we will need a locally list-decodable ECC that corrects a $\frac{1}{2} - \delta$ fraction of errors. By Theorem B.2, there exists a function $L = \mathsf{poly}(1/\delta)$, a function $r = O(\log n)$, a locally $(L, \frac{1}{2} - \delta)$-list-decodable ECC $\mathsf{Enc}_{n,\delta}$ that produces codewords of length 2^r, and a decoding algorithm $\mathsf{Dec}_{n,\delta}$ that produces a list of L elements (in a local fashion). We will also need a NW generator that takes functions with truthtable length 2^r (matching the output length of the ECC) and outputs m bits (matching the output length of our PRG). To achieve this, we require a (d, r, s)-design $\mathcal{I}$ that contains m subsets of $[d]$. By Lemma B.5, we can pick $s = \log m/3$ to ensure that $\mathcal{I}$ contains m subsets, and pick $d = \Omega(\log^2 n / \log m)$ to satisfy that $d > 10r^2/s$. (Such designs can be efficiently generated by $\mathsf{GenDesign}$.)

We proceed to describe our PRG formally. We will consider a function $G : 1^n \times 1^m \times \{0,1\}^d \to \{0,1\}^m$ defined as follows. For any function $f : [n] \to \{0,1\}$, on input $(1^n, 1^m, y)$ where $y \in \{0,1\}^d$, the algorithm G proceeds in the following steps.

- G (given access to f) first computes $z = f(1)||f(2)||\ldots||f(n)$.
- Then G lets $h : \{0,1\}^r \to \{0,1\}$ be the function $h = \mathsf{fn}(\mathsf{Enc}_{n,\delta}(z))$ that is associated with the truthtable $\mathsf{Enc}_{n,\delta}(z) \in \{0,1\}^{2^r}$.
- Next, G invokes the design generating algorithm $\mathsf{GenDesign}(d, r, s)$ to obtain a (d, r, s)-design $\mathcal{I} = \{I_1, \ldots, I_m\}$.

- Finally, G outputs
$$G(1^n, 1^m, y) = \mathsf{NW}^h_{\mathcal{I}}(y) = h(y_{I_1}) \dots h(y_{I_m})$$
where the function NW is defined in Definition B.3.

Note that the seed length of the PRG $d = O(\log^2 n / \log m)$, so the seed length condition in Theorem 3.11 is satisfied.

We claim that G is a $k(\cdot)$-reconstructive PRG for $k(n, m) = \mathsf{poly}(m, \log n)$. Fix some integer n, m, some function $f : [n] \to \{0, 1\}$, and consider any statistic test $T : \{0, 1\}^m \to \{0, 1\}$ such that

$$\left| \Pr[v \leftarrow \{0,1\}^d : T(G(1^n, 1^m, v)) = 1] - \Pr[w \leftarrow \{0,1\}^m : T(w) = 1] \right| \geq \alpha \tag{9}$$

We will show that there exist algorithms M and R (as defined in Definition 3.10).

Constructing M and R. Consider an algorithm M that defines as follows.

1. On input $1^n, 1^m$, given access to f and T, M computes $z = f(1)||f(2)|| \dots ||f(n)$ and the function $h = \mathsf{fn}(\mathsf{Enc}_{n,\delta}(z))$.
2. M runs the design generating algorithm $\mathsf{GenDesign}(d, r, s)$ to obtain a (d, r, s)-design $\mathcal{I} = \{I_1, \dots, I_m\}$.
3. M executes the NW reconstruction algorithm $\mathsf{NWRecon}^T(h, \mathcal{I}, \alpha)$ and let Π denotes the program it outputs.
4. M will let $\mathcal{O}$ be an oracle such that $\mathcal{O}(p) = \Pi^T(\mathcal{I}, p)$ (for any $p \in [2^r]$).
5. leak will pick a good random tape γ for $\mathsf{Dec}_{n,\delta}$ such that there exists an index pos satisfying
$$\forall i \in [n], \mathsf{Dec}^{\mathcal{O}}_{n,\delta}(\mathsf{pos}, i; \gamma) = z_i.$$
This can be done in a brute-force way by repeated sampling γ and enumerating every value of pos and i to check if γ is good.
6. $\mathsf{leak}(x, z)$ will output $(\Pi, \gamma, \mathsf{pos})$.

We then turn to describing the algorithm R (given the access to T).

1. On input 1^m, the advice string a (interpreted as $(\Pi, \gamma, \mathsf{pos})$), and a bit index i, R first invokes $\mathsf{GenDesign}(d, r, s)$ to get a (d, r, s)-design $\mathcal{I} = \{I_1, \dots, I_m\}$.
2. R lets $\mathcal{O} = \Pi^T(\mathcal{I}, \cdot)$.
3. R outputs $\mathsf{Dec}^{\mathcal{O}}_{n,\delta}(\mathsf{pos}, i; \gamma)$.

Analyzing the reconstruction. We turn to analyzing the algorithms M and R. We first show that $R^T(1^m, M^{f,T}(1^n, 1^m), i)$ will indeed compute $f(i)$ for all $i \in [n]$. This follows from the correctness of the distinguisher T, the NW reconstruction algorithm, and the list decoding algorithm. Observe that $G(1^n, 1^m, v) = \mathsf{NW}^{\mathsf{fn}(\mathsf{Enc}_{n,\delta}(z))}_{\mathcal{I}}(v)$. Therefore, by Equation 9, T distinguishes the output of $\mathsf{NW}^h_{\mathcal{I}}$ from uniform with advantage α and breaks the NW generator. Then by Lemma B.6, the NW reconstruction algorithm $\mathsf{NWRecon}^T(h, \mathcal{I}, \alpha)$ will output a good approximation for h; that is, it holds that

$$\Pr[p \leftarrow [2^r] : \mathsf{Enc}_{n,\delta}(z)_p \neq \Pi^T(\mathcal{I}, p)] \leq \frac{1}{2} - \frac{\alpha}{2m} \leq \frac{1}{2} - \delta$$

So $\mathcal{O} = \Pi^T(\mathcal{I}, \cdot)$ is an oracle that is relatively close to $\mathsf{Enc}_{n,\delta}(z)$. Since Enc is a good error correcting code, by Theorem B.2, $\mathsf{Dec}^{\mathcal{O}}_{n,\delta}$ will return a list containing z and will be able to produce each element in the list locally. $M^{f,T}(1^n, 1^m)$ will then find the coordinate pos and a good random tape γ such that $\mathsf{Dec}^{\mathcal{O}}_{n,\delta}(\mathsf{pos}, \cdot; \gamma)$ locally computes z. Note that $R^T(1^m, M^{f,T}(1^n, 1^m), i)$ will finally output $\mathsf{Dec}^{\mathcal{O}}_{n,\delta}(\mathsf{pos}, i; \gamma)$ and we conclude that $R^T(1^m, M^{f,T}(1^n, 1^m), i)$ will produce $z_i = f(i)$.

We next argue that $M^{f,T}(1^n, 1^m)$ is short and of length at most n^{ε}. $M^{f,T}(1^n, 1^m)$ consists of Π, γ, and pos. Note that by Lemma B.6, the description of machine Π is of length $m \cdot 2^s + m + d + \log(drsm) < \mathsf{poly}(m, \log n)$. The random tape γ is of length at most the running time of $\mathsf{Dec}_{n,\delta}$, which is at most $\mathsf{poly}(\log n, 1/\delta) < \mathsf{poly}(m, \log n)$. The coordinate pos is of length at most $\log L(n) = O(\log m)$. Therefore, $|M^{f,T}(1^n, 1^m)| \leq \mathsf{poly}(m, \log n)$.

We then prove that the algorithm $M^{f,T}(1^n, 1^m)$ runs in polynomial time. Note that f is given as oracle to M, so $\mathsf{Enc}_{n,\delta}(z)$ takes only $\mathsf{poly}(n, 1/\delta) = \mathsf{poly}(n)$ steps. $\mathsf{GenDesign}(d, r, s)$ runs in time $\mathsf{poly}(2^s, d) \leq \mathsf{poly}(m, O(1/\varepsilon) \log n)$. It takes $\mathsf{poly}(2^r, m, 2^s, 1/\alpha) = \mathsf{poly}(n)$ steps to execute $\mathsf{NWRecon}^T(h, \mathcal{I}, \alpha)$. The list-decoding algorithm runs in time $\mathsf{poly}(\log n, 1/\delta) = \mathsf{poly}(m)$, the oracle $\mathcal{O}$ runs in $\mathsf{poly}(r, m, 2^s) = \mathsf{poly}(m)$ time. Thus, M takes $\mathsf{poly}(m)L \cdot O(n) = \mathsf{poly}(n)$ time to pick a good random tape γ and an index pos. To sum up, the running time of M can be bounded by $\mathsf{poly}(n)$.

We now show that the algorithm R runs in polynomial. R takes (a, i) of input, which is of length $m + \log n$. R first invokes $\mathsf{GenDesign}(d, r, s)$ which runs in $\mathsf{poly}(m)$ time. Note that both the oracle $\mathcal{O}$ and the list-decoding algorithm runs in time $\mathsf{poly}(m)$, it follows that R runs in $\mathsf{poly}(m)$ time.

It remains to show that G runs in polynomial. Note that it takes $O(n)$ time to compute $z = f(1)||f(2)||\ldots||f(n)$, and algorithms $\mathsf{Enc}, \mathsf{GenDesign}, \mathsf{NW}$ run in time polynomial in n and m. Thus, it follows that $G(1^n, 1^m, \cdot)$ runs in time $\mathsf{poly}(n, m)$. ■

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