Leakage-Resilient Hardness Equivalence to Logspace Derandomization

Yakov Shalunov[1] ✉ ⊕
University of Chicago, IL, USA

───── **Abstract** ─────

Efficient derandomization has long been a goal in complexity theory, and a major recent result by Yanyi Liu and Rafael Pass identifies a new class of hardness assumption under which it is possible to perform time-bounded derandomization efficiently: that of "leakage-resilient hardness." They identify a specific form of this assumption which is *equivalent* to prP = prBPP.

In this paper, we pursue an equivalence to derandomization of prBP·L (logspace promise problems with two-way randomness) through techniques analogous to Liu and Pass.

We are able to obtain an equivalence between a similar "leakage-resilient hardness" assumption and a slightly stronger statement than derandomization of prBP·L, that of finding "non-no" instances of "promise search problems."

1 Introduction

In a time-bounded setting, pseudorandom generators have historically been used to show that certain circuit-complexity lower bounds imply that P = BPP (in particular, problems in E which require exponential size circuits [5]). Another key result showed that P = BPP (and specifically, a deterministic polynomial-time algorithm for polynomial identity testing) implies that either NEXP requires super-polynomial boolean circuits or computing permanents requires super-polynomial arithmetic circuits [6]. However, historically the lower bounds known to imply P = BPP and those implied by P = BPP have not matched. Seeing these results, one might hope to find a hardness assumption H such that

> "P = BPP *if and only if there exists a problem satisfying H*."

A recent exciting paper, "Leakage-Resilient Hardness v.s. Randomness" [8] by Liu and Pass, has done exactly this, building on prior work by Chen and Tell [2]; Nisan and Wigderson [9]; Impagliazzo and Wigderson [5]; Sudan, Trevisan, and Vadhan [13]; and

---

[1] Research conducted as a student at the California Institute of Technology.

Goldreich [4]. Liu and Pass found an alternate form of hardness assumption which they were able to formulate as equivalent to prP = prBPP using a cryptographic notion of "leakage-resilient hard functions" [11] – i.e., hard functions which are still uniformly hard even if you have some "leaked" information about the output (formalized as a "leakage function" of bounded output length taking the hard function's output as an input).

In the space-bounded derandomization setting, there's been an enormous effort to prove unconditionally that L = BPL, and while it's believed that there's no obstruction to being able to do so, this goal continues to remain elusive. However, it is also known that it is possible to get a conditional derandomization in logarithmic space with PRG machinery similar to the time-bounded randomization setting [7].

1.1   Overview

In the vein of conditional derandomization, this paper constructs a result similar to Liu and Pass for logarithmic space-bounded computation. We obtain an equivalence utilizing a space-bounded analog to the time-bounded assumption in Liu and Pass's paper.

Liu and Pass's ideas do not carry directly to the space-bounded setting because certain manipulations required by the proof cannot be carried out in logspace. In particular:

Worst-case vs. average-case hardness.   A key step in the Liu and Pass proof that "hardness implies derandomization" uses error-correcting codes to convert worst-case hardness to average-case hardness. In the space-bounded setting, we do not have access to error-correcting codes capable of list-decoding the available $\frac{1}{2} + \mathcal{O}\left(\frac{1}{n}\right)$ fraction of correct bits. As a consequence, our version of leakage-resilient hardness is average case instead of worst case. Using a stronger hardness assumption then required additional work in the converse direction.

Average-case hardness.   When analyzing the constructed Nisan-Wigderson generator in the standard way, Liu and Pass use rejection sampling to find a prefix fixing for Yao's distingiusher-to-predictor transformation with a sublinear number of trials. However, in the space-bounded setting, we cannot write down a single fixing (since the size of the fixing is superlogarithmic), so our hardness assumption must be strengthened until even a linearly small chance of having the correct fixing contradicts the hardness assumption. This version then allows us to sample once and write it directly to the output tape without verification.

Search problem reduction.   When Liu and Pass prove that prP = prBPP implies their hardness assumption, they rely on a prior result by Goldreich to reduce the task of finding a non-no solution for a prBPP search problem to a polynomial number of prBPP decision problems [4], which can then be solved deterministically by assumption. In the space-bounded setting, we've found no equivalent transformation and, as a result, our main theorem works directly with the non-standard derandomization of search problems (which in polynomial time is equivalent to prP = prBPP). Specifically, we use a "one-sided search" which, given a "promise search problem," finds a solution which is not a no-instance (these terms are more precisely defined in the following section).

Nevertheless, we are able to obtain:

▶ **Theorem 1.1** (Main Theorem (informal)). *There is a logspace-computable function which is leakage-resilient hard if and only if it is possible, in logspace, to deterministically solve a one-sided search for any* prBP·L *search problem.*

This shows a meaningful equivalence between a hardness assumption and a derandomization of a search version of prBP·L. Notably, the derandomization necessary follows from the existence of a standard pseudorandom generator construction powerful enough to derandomize prBP·L.

Shortly after the original release of this paper as a preprint, Pass and Renard released their own, independent result "Characterizing the Power of (Persistent) Randomness in Log-space" [10] which presents a theorem very similar to our own – their result is a full derandomization of prBP·L search problems under a roughly equivalent condition. Their techniques in both directions are analogous to ours and their result is fundamentally similar.

In particular, despite presenting their result differently, Pass and Renard's work demonstrates the same one-sided search as ours. While their derandomization direction (claims 5.3, 4.12, 4.9, and 4.10 in [10]) relies on a non-promise version of BP·L search problems (where non-no instances are yes instances), their hardness direction relies on promise search like ours. (Notably, Goldreich's reduction in polynomial time also produces non-no instances rather than yes instances, suggesting this is a more fundamental limit of this formulation of search problems.) Unfortunately, this produces a mismatch that makes their results not quite correct-as-written.

Pass and Renard's paper further contributes a "partial derandomization" analogous to Liu and Pass's low-end regime (i.e. for positive γ, containment of searchBP·L in searchL$^{1+\gamma}$ is equivalent to existence of $\mathcal{O}(\log^{1+\gamma} n)$ computable hard f). Though we believe it contains the same mismatch of directions, after some simple reformatting into a one-sided derandomization, it will be an elegant extension to the equivalence.

We believe our proof of the key equivalence is more direct, briefer, and clearer as it, in particular, achieves equivalence without the use of Doron and Tell's [3] recent results in logspace PRGs (with no significant increase in complexity) by using a better suited version of leakage resilient hardness. Their result does, however, show that our versions of leakage resilient hardness are equivalent to each other.

Because our result follows a similar path to Liu and Pass's, in Section 2 we provide an accessible summary and informal description of the key ideas in their result.

In Section 3, we provide a formal statement of our result and the definitions necessary for it.

In Sections 4 and 5, we prove the forward and reverse directions of the claim respectively, and in Section 6 we discuss potential further work.

2 Informal description of Liu and Pass's polynomial-time equivalence result

In this section, we provide a high-level summary of Liu and Pass's proof, which also serves as an outline for our own work.

Since it is central to the proof, we start with the definition of "leakage-resilient hardness," which is a uniform hardness assumption coined by Rivest and Shamir in 1985 [11] and modified by Liu and Pass based on Chen and Tell's [2] work with almost-all input hardness.

In the proof, leakage is used to uniformly substitute for non-uniform hard-coded bits in the proof of correctness of the Nisan-Wigderson generator, which we expand on later in the section.

▶ **Definition 2.1** (Almost-all input leakage-resilient hardness [8][11][2]). *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a function. We say that f is* almost-all-input (T,ℓ)-leakage-resilient hard *if for all T-time probabilistic algorithms* (leak, A) *satisfying* $|\mathsf{leak}(x, f(x))| \leqslant \ell(|x|)$, *for all sufficiently long strings x, $A(x, \mathsf{leak}(x, f(x))) \neq f(x)$ with probability at least $\frac{2}{3}$.*

To illustrate this property, consider when f is the classical example of a (potentially) hard function used in cryptography: prime factorization of a product, x, of two primes. Given $f(x)$ (a correct factorization of x), a leakage function $\mathsf{leak}(x, f(x))$ which picks the smaller prime number from $f(x)$ and leaks it would make prime factorization "easy" as it would allow the attacker A to simply be a division algorithm. Since both division and identifying the smaller factor can be done in polynomial (say, n^k) time, and the smaller factor must be at most $n/2$ bits, prime factorization is *not* $(n^k, n/2)$-leakage-resilient hard – even if, without leakage, it requires superpolynomial time.

The simplest expression of Liu and Pass's result as relevant to ours is the following:

▶ **Theorem 2.2** (Liu and Pass [8])**.** *There exists a constant c such that for all $\varepsilon \in (0, 1)$, the following are equivalent:*

- *There exists a function $f : \{0, 1\}^n \to \{0, 1\}^n$, computable in deterministic polynomial time, which is almost-all input (n^c, n^ε)-leakage-resilient hard; and*
- $\mathsf{prP} = \mathsf{prBPP}$.

2.1   Forward direction (Liu and Pass)

Liu and Pass's forward direction – hardness assumption to $\mathsf{prP} = \mathsf{prBPP}$ – adapts the original proof that the Nisan-Wigderson pseudorandom generator (NW PRG) yields derandomization, with a few key differences. The generator they construct (invented by Goldreich [4]) is "targeted," meaning both it and the distinguishers it beats get access to a "target" string as an input in addition to the seed. When performing derandomization using the generator, the appropriately padded input becomes the target. (The utility of this is explained at the end of this subsection.)

Their construction of a NW-like PRG uses the error-correcting encoded output of the leakage-resilient hard function f on the "target" x instead of the standard hard-coded truth table of a non-uniformly hard function. (In our proof, the error-correcting encoding step is replaced with average-case hardness.)

Recall that in the standard proof of correctness for a Nisan-Wigderson pseudorandom generator, when converting a distinguisher to a predictor, there is a step where a prefix is fixed and a small table of a subset of outputs from the hard truth-table is hard-coded. The key difference here is that, in order to obtain a contradiction, instead of hard-coding this prefix and table non-uniformly, it is instead computed by the leak function, which has access to the hard truth-table (since it is efficiently computed from the output of f, which leak has access to). The prefixes are found by rejection sampling to pick a "good one." (In our proof, we select one randomly.) This output can be made polynomially small compared to n to satisfy the length bound ℓ on leak.

The final step of the proof is to argue that if there is a problem in prBPP that cannot be derandomized by this PRG, we can turn this into a distinguisher by fixing the problem. Observe, however, that for a contradiction, we need derandomization to fail in infinitely many cases (since the hardness assumption is *almost*-all input). Furthermore, we are only fooling *uniform* distinguishers. This is where the targeting of the PRG is necessary.

By using (a padded version of) the input to the decision problem, x, as the target, we ensure that we can uniformly "find" the infinitely many values of x where the distinguisher succeeds. If the PRG were untargeted, we would have no way to uniformly identify which value of x the PRG fails on for a given length, so failing to derandomize some decision problem would not guarantee a distinguisher.

(In our proof, we additionally perform a one-sided search derandomization. The search problem's search algorithm (the "finder") is run a with pseudorandom string generated for each possible seed until one is found which the verifier accepts. We show that such a seed must exist and that this process produces the desired derandomization. In the polynomial time regime, this one-sided search derandomization follows from decision derandomization directly without further access to a PRG and so happens on the converse side.)

2.2 Backward direction (Liu and Pass)

The key idea of the backward direction – $\mathsf{prP} = \mathsf{prBPP}$ to hardness assumption – is that, if we were able to assign a "random" string to each input, we could build such an f since leaking some information about that random string would not help you reconstruct the rest of the output.

Liu and Pass make use of a result by Goldreich [4] which states that finding non-no solutions to a prBPP search problem can be reduced to a polynomial number of prBPP decision problems. Liu and Pass then formulate finding a "good output" $f(x)$ for any given input x as a prBPP search problem (defined below). They then apply Goldreich's result and then the derandomization assumption to create a hard function whose output is deterministic but behaves sufficiently like a "random string" would.

In Goldreich's formulation, a prBPP search problem is defined as follows:

▶ **Definition 2.3** (prBPP search problems [4])**.** $R_{\mathrm{YES}}, R_{\mathrm{NO}} \subseteq \{0,1\}^* \times \{0,1\}^*$ *where $R_{\mathrm{YES}} \cap R_{\mathrm{NO}} = \varnothing$ represent a* prBPP *search problem if both of the following hold:*

- *R_{YES} and R_{NO} represent a* prBPP *decision problem. I.e., there is a PPT machine V (the "verifier") such that*

$$\forall (x,y) \in R_{\mathrm{YES}}, \ \mathbb{P}[V(x,y) = 1] \geqslant \frac{2}{3} \quad and \quad \forall (x,y) \in R_{\mathrm{NO}}, \ \mathbb{P}[V(x,y) = 0] \geqslant \frac{2}{3}$$

- *There exists a PPT machine F (the "finder") such that for all x, if $\exists y, (x,y) \in R_{\mathrm{YES}}$, then*

$$\mathbb{P}[(x, F(x)) \in R_{\mathrm{YES}}] \geqslant \frac{2}{3}$$

Let $S_R = \{x : \exists y, (x,y) \in R_{\mathrm{YES}}\}$ be the set of inputs for which a solution exists.

Liu and Pass formulate computing f as finding solutions to a search problem. A given pair (x,y) is a yes-instance of the problem if every pair (A, leak) of attacker and leakage functions whose program descriptions have length at most $\log n$ and which run in the desired time bound has $A(x, \mathsf{leak}(x,y)) = y$ with probability at most $1/6$. Similarly, (x,y) is a no-instance if there exists (A, leak) such that $A(x, \mathsf{leak}(x,y)) = y$ with probability at least $1/3$. (The gap between yes- and no-instance probabilities is allowed by it being a *promise* search problem and is used by the verifier.)

Only attacker pairs bounded to length $2\log n$ need to be considered because this only excludes *any particular* machine from consideration for a finite number of inputs x.

When allowed to use randomness, the finder is simple – as established, simply ignoring x and picking a random string y works with high probability. The verifier can check whether a yes-or-no instance is a yes-instance by simply iterating over all attackers and leakage functions up to length $\log n$ (truncating their run time to the required bound) and checking whether any of them compute y with probability greater than $1/4$. This runs in polynomial time since there are n machines each truncated to a polynomial time bound and each pair only needs to be run polynomially many times to be correct with high probability.

The desired hard function can then be computed by using Goldreich's reduction to reduce the search problem to a polynomial number of prBPP problems. These are then derandomized to deterministically produce a non-no instance. The y-component of the resulting non-no instance is then exactly the desired hard output, by the construction of the search problem.

(In our case, we need access to the PRG rather than merely $\mathsf{prL} = \mathsf{prBP \cdot L}$ to perform the one-sided search, so it is performed when proving the forward direction.)

3 Results

▶ **Theorem 3.1** (Main theorem). *There exist parameters $c_1, c_2, \alpha > 0$ (in particular, $\alpha = \frac{1}{3}$ works) such that the following are equivalent:*

1. *There exists a logspace function f which is almost-all-input $(n^{c_1}, c_2 \log n, \ell, \delta)$-leakage-resilient average hard where $\ell = n^\varepsilon$ (for $\varepsilon = 2\alpha + \frac{\alpha^3}{5}$) and $\delta = \left(\frac{1}{2} - \frac{1}{m^2}\right)n$ (for $m = n^{\alpha^3/5}$).*
2. *For any $\mathsf{prBP \cdot L}$ search problem $R_{\mathrm{YES}}, R_{\mathrm{NO}}$, we can create a deterministic algorithm F' such that for any $x \in S_R$, we have that $(x, F'(x)) \notin R_{\mathrm{NO}}$. (In particular, this implies $\mathsf{prBP \cdot L} = \mathsf{prL}$.)*

Proof. The forward direction is theorem 4.1 and the reverse direction is theorem 5.1.   ◀

3.1 Preliminaries

Note that we are working with a read-*only* random tape with two-way movement. For decision problems, this definition corresponds to the complexity class $\mathsf{BP \cdot L}$, though we work with its promise and search promise counterparts.

Our result is formulated in terms of "search problems," analogous to the prBPP search problems defined by Goldreich in [4].

▶ **Definition 3.2** (prBP·L search problems). *A pair of sets $R_{\mathrm{YES}}, R_{\mathrm{NO}} \subseteq \{0,1\}^* \times \{0,1\}^*$ where $R_{\mathrm{YES}} \cap R_{\mathrm{NO}} = \varnothing$ represent a $\mathsf{prBP \cdot L}$ search problem if both of the following hold*

- *R_{YES} and R_{NO} represent a $\mathsf{prBP \cdot L}$ decision problem. I.e., there is a PPT logspace machine V (the "verifier") such that*

$$\forall (x,y) \in R_{\mathrm{YES}}, \ \mathbb{P}[V(x,y) = 1] \geqslant \frac{2}{3} \quad and \quad \forall (x,y) \in R_{\mathrm{NO}}, \ \mathbb{P}[V(x,y) = 0] \geqslant \frac{2}{3}$$

- *There exists a PPT logspace machine F (the "finder") such that for all x, if $\exists y, (x,y) \in R_{\mathrm{YES}}$, then*

$$\mathbb{P}[(x, F(x)) \in R_{\mathrm{YES}}] \geqslant \frac{2}{3}$$

Let $S_R = \{x : \exists y, (x,y) \in R_{\mathrm{YES}}\}$ be the set of inputs for which a solution exists.

This definition is identical to the polynomial variant with the added constraint that the machines are bounded to logarithmic space in addition to the polynomial time bound.

Our hardness assumption is a modification of leakage resilient hardness as defined in definition 2.1 of [8], which in turn builds on [11].

▶ **Definition 3.3** (Almost-all-input leakage resilient average hard). *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a multi-output function. f is almost-all-input (T, S, ℓ, d)-leakage resilient average hard if for all S-space T-time probabilistic algorithms (A, leak) satisfying $|\mathsf{leak}(x, f(x))| \leqslant \ell(n)$, we have that for all sufficiently long strings x,*

$$\mathbb{P}[d_H(A(x, \mathsf{leak}(x, f(x))), f(x)) < d(n)] < \frac{1}{n}$$

with probability over the internal randomness of the algorithms, where d_H is Hamming distance.

Note that this is "stronger" than a direct analog to the time-bounded assumption [8] in two ways: first, and most importantly, we've added another parameter which parameterizes how "good of an approximation" is allowed, whereas in the time-bounded setting, forbidding strict equality was sufficient. Second, the allowed probability of "success" is $\frac{1}{n}$ rather than $\frac{1}{3}$.

The first difference converts the worst-case hardness to average-case hardness. It is necessary because in the time-bounded setting, we have access to much more powerful error-correcting codes which are not available in the logspace setting. In the time-bounded setting, error-correcting codes let us convert a small edge in breaking the generator to an exact computation of the hard function. Meanwhile, in the logspace setting, we're limited to just that small edge, so we bake it into the hardness assumption instead.

The second difference exists to handle issues in the distinguisher to predictor transformation. In polynomial time, the attacker functions can "try" $o(1/n)$ strings and figure out which one is the ideal choice, but because the strings are super-logarithmic in length, we cannot verify them in logspace. Thus, the hardness assumption is strengthened to allow simply choosing a random one without verification.

While much of the high-level structure of the following proof is similar to that presented in section 2, we have that:

- in the "hard function implies derandomization" direction, we show the desired derandomization of search problems; and
- in the "derandomization implies hard function" direction, the search problem is constructed such that anything which is not a no-instance is a valid value for the function to take for that x. Then, instead of Goldreich's reduction, we apply the derandomization reduction directly.

4 Existence of hard function implies derandomization

▶ **Theorem 4.1** (Existence implies derandomization)**.** *The forward direction of the main theorem (3.1). I.e., (1) implies (2).*

Proof. Apply lemma 4.4 with $C = 3$ to get the necessary PRG, then apply lemma 4.7 to perform the desired derandomization with the PRG. ◀

The following are fairly standard definitions of a distinguisher and pseudorandom generator with the caveat of the additional "target" string. For brevity, write $D_x^m(\gamma)$ for $D(1^m, x, \gamma)$ and similarly for G.

▶ **Definition 4.2** (Targeted distinguisher)**.** *Given a function $G : \{1\}^m \times \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$, a targeted distinguisher for G with advantage $\beta > 0$ is a machine $D : \{1\}^m \times \{0,1\}^n \times \{0,1\}^m$ such that for all sufficiently large m and for all $x \in \{0,1\}^n$, we have that*

$$\left| \mathbb{P}_{s \sim \mathcal{U}_d}\left[D_x^m(G_x^m(s)) = 1 \right] - \mathbb{P}_{\gamma \sim \mathcal{U}_m}\left[D_x^m(\gamma) = 1 \right] \right| \geqslant \beta \tag{1}$$

where $\mathcal{U}_k$ is the uniform distribution on $\{0,1\}^k$.

Throughout, we will refer to targeted distinguishers as "distinguishers."

This definition becomes useful in the context of the following (modified from [4, 8]):

▶ **Definition 4.3** (Uniform targeted pseudorandom generator)**.** *For parameters S, n, and d dependent on output length m, an S-secure uniform (n,d)-targeted PRG is a deterministic function*

$$G : \{1\}^m \times \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$$

which satisfies the property that there does not exist a targeted distinguisher with advantage β running in space S for any $\beta > 0$.

Throughout the rest of this article, "PRG" and "pseudorandom generator" refer to a uniform targeted PRG. n is the "target length" of the generator and d is the "seed length."

We show that our hardness assumption implies an $\mathcal{O}(\log m)$-secure $(\mathsf{poly}(m), \mathcal{O}(\log m))$ PRG and then that the existence of an such a PRG implies the desired derandomization.

4.1 Hard function implies existence of PRG

First, we prove that a hard function with the given parameters implies the existence of a PRG.

Fix a sufficiently small value of $\alpha > 0$. $\alpha = 1/3$ works.

▶ **Lemma 4.4** (Hard function implies existence of PRG). *For any C, there exist constants c_1, c_2 such that if there exists a logspace computable function f such that f is almost-all-input $\left(n^{c_1}, c_2 \log n, n^\varepsilon, \left(\frac{1}{2} - \frac{1}{m^2}\right)n\right)$-leakage resilient average hard with $m = n^{\frac{\alpha^3}{5}}$ and $\varepsilon = 2\alpha + \frac{\alpha^3}{5}$, then there exists a $C \log m$-secure $(n, \frac{\log n}{\alpha})$ PRG which is logspace computable.*

As is standard for a Nisan-Wigderson generator, we will require designs for the construction.

▶ **Definition 4.5** (Combinatorial design). *A design is a collection of (potentially large) sets with a bounded pairwise intersection size. More precisely:*

For any natural numbers (d, r, s) such that $d > r > s$, a (d, r, s)-design of size m is a collection $\mathcal{I} = \{I_1, \ldots, I_m\}$ of subsets of $[d]$ such that for each $j \in [m]$, $|I_j| = r$ and for each $k \in [m]$ such that $k \neq j$, $|I_j \cap I_k| \leqslant s$.

Because we're working in logspace, we will need the following lemma:

▶ **Lemma 4.6** (Creating designs). *There is a deterministic algorithm we call $\mathsf{GenDesign}(d, \alpha)$ which produces a $(d, \alpha d, 2\alpha^2 d)$ design of size $2^{\frac{\alpha^4 d}{5}}$ in space $S_{\mathsf{GenDesign}} = \mathcal{O}(d)$.*

Proof. Proven by Klivans and Melkebeek in the appendix of [7]. ◀

Outline of proof of 4.4. As presented in the overview, the construction of the PRG is a fairly standard Nisan-Wigderson generator [9] construction, with the caveats that it is a targeted PRG and that instead of the truth table of a hard-coded non-uniformly hard function, we use the output of our f on the target string. The key change is that the leakage function then replaces the hard-coded tables in the search-to-decision reduction.

We show that this can be done in logspace and that the hard-coded tables fit in the inverse polynomial leakage bound n^ε. In order to compute the tables in logspace, the probability of generating "good tables" needs to be low because it's impossible to verify the tables. However, a probability high enough to contradict leakage-resilient hardness, i.e., over $\frac{1}{n}$, is achievable.

Assuming a distinguisher that defeats the PRG, the attacker function is then able to use the tables from the leakage function together with the distinguisher to perform the search-to-decision reduction with enough accuracy to violate the average-case hardness constraint. ◀

Proof of 4.4. With an appropriate choice of c_1, c_2, we can construct a $C \log m$-secure PRG for any constant $C \geqslant 0$.

We construct a targeted PRG $G : \{1\}^m \times \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$, where $n = m^{\frac{5}{\alpha^3}}$ and $d = \frac{\log n}{\alpha}$.

The algorithm is similar to that in time-bounded case [8]: it is simply a Nisan Wigderson generator [9], with the hard-coded truth table replaced with one outputted by the hard function.

On input $1^m, x, s$, it proceeds as follows:

1. Compute $z = f(x)$. Define $h(i) = z_i$.
2. Compute $\mathcal{I} = \mathsf{GenDesign}(d, \alpha)$. Note that $r = d\alpha = \log n$ and $2^{\frac{\alpha^4 d}{5}} \geqslant m$ due to our choice of n and d.
3. Output

$$G(1^m, x, s) = h(s_{I_1}) \cdots h(s_{I_m})$$

Suppose there exists a distinguisher D computable in space $C \log m$ with advantage $\beta > 0$. We will use this to approximate f with enough of an advantage to violate the average hardness assumption by a standard hybrid argument (lemma 3.15 in [8], proposition 7.16 in [14], theorem 10.12 in [1]).

We can remove the absolute value from the distinguisher definition (1) by noting that there exists $b \in \{0, 1\}$ such that

$$\mathbb{P}_{s \sim \mathcal{U}_d}[D_x^m(G_x^m(s)) = b] - \mathbb{P}_{\gamma \sim \mathcal{U}_m}[D_x^m(\gamma) = b] \geqslant \beta$$

since flipping b flips the sign.

Then, for every $j \in \{0, \ldots, m\}$, define

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

(with h defined as in the generator) where $s \sim \mathcal{U}_d$ and each $w_k \sim U_1$ (for $j + 1 \leqslant k \leqslant m$). Notice that $H_0 = \mathcal{U}_m$ and $H_m = G(1^m, x, \mathcal{U}_d)$. Therefore it follows that

$$\frac{1}{m} \sum_{j \in 1}^{m} \left(\mathbb{P}_{y,w}[D_x^m(H_j) = b] - \mathbb{P}_{y,w}[D_x^m(H_{j-1}) = b] \right)$$

$$= \frac{1}{m} \left(\mathbb{P}_{y,w}[D_x^m(H_m) = b] - \mathbb{P}_{y,w}[D_x^m(\mathcal{U}_0) = b] \right) \geqslant \frac{\beta}{m}$$

Considering j as a random variable distributed uniformly over $[m]$, we get that

$$\mathbb{E}_{j \in [m]} \left[\mathbb{P}_{y,w}[D_x^m(H_j) = b] - \mathbb{P}_{y,w}[D_x^m(H_{j-1}) = b] \right] \geqslant \frac{\beta}{m}$$

Since $\mathbb{P}_{y,w}[D_x^m(H_j) = b] - \mathbb{P}_{y,w}[D_x^m(H_{j-1}) = b]$ is upper bounded by 1, by an averaging argument, with probability at least $\frac{\beta}{2m}$ over the choice of $j \leftarrow [m]$, $y_{[d] \setminus I_j} \leftarrow \{0, 1\}^{d-r}$, and $w_{[m] \setminus [j]} \leftarrow \{0, 1\}^{m-j}$, the strings j, $y_{[d] \setminus I_j}$, and $w_{[m] \setminus [j]}$ will satisfy:

$$\mathbb{P}_{y_{I_j} \sim \mathcal{U}_r}[D_x^m(H_j) = b] - \mathbb{P}_{(y_{I_j}, w_j) \sim \mathcal{U}_{r+1}}[D(H_{j-1}) = b] \geqslant \frac{\beta}{2m} \tag{2}$$

Suppose we have a choice j, $y_{[d] \setminus I_j}$, and $w_{[m] \setminus [j]}$ satisfying the above equation. By Yao's prediction vs. indistinguishability theorem [15], it holds that

$$\mathbb{P}_{(y_{I_j}, w_j) \sim \mathcal{U}_{r+1}}\left[D_x^m(H_{j-1}) \oplus b \oplus w_j = h(y_{I_j})\right] \geqslant \frac{1}{2} + \frac{\beta}{2m}$$

There then must exist a choice of w_j such that

$$\mathbb{P}_{y_{I_j} \sim \mathcal{U}_r}\left[D_x^m(H_{j-1}) \oplus b \oplus w_j = h(y_{I_j})\right] \geqslant \frac{1}{2} + \frac{\beta}{2m} \tag{3}$$

Observe that in order to compute H_{j-1} as a function of y_{I_j}, one does not need to know the entire truth table of h. Instead, note that the overlap between I_j and I_i, $i < j$, is at most $2\alpha^2 d = 2\alpha r$. Thus, as a function of y_{I_j}, $h(y_{I_i})$ can only take $2^{2\alpha r}$ possible values. Thus, to compute all of H_{j-1}, one needs $j2^{2\alpha r} \leqslant m2^{2\alpha r}$ bits of the truth table of h.

Then, given a distinguisher D with advantage β, we can approximate f with the functions (A, leak) described below.

On input $x, z = f(x)$, leak proceeds as follows (denote $h(i) = z_i$):

1. Evaluates $\mathcal{I} = \mathsf{GenDesign}(d, \alpha)$, with d chosen the same way as the PRG.
2. Chooses a random $j \in [m]$ and a random $y_{[d]\setminus I_j}$, storing them on its work tape.
3. Writes j and $y_{[d]\setminus I_j}$ to the output tape.
4. For $0 < i < j$ and for each value of y obtained by ranging over possible values of y_{I_j} (since $y_{[d]\setminus I_j}$ is already fixed), leak writes $h(y_{I_i})$ to the output tape.
5. Writes random bits b, w_j and string $w_{[m]-[j]}$ to the output tape.

Then, on input $x, w = \mathsf{leak}(x, f(x))$, A proceeds as follows:

1. A evaluates $\mathcal{I} = \mathsf{GenDesign}(d, \alpha)$, with d chosen the same way as the PRG.
2. Defines $b, j, y_{[d]\setminus I_j}, w_j$, and $w_{[m]\setminus[j]}$ to be those read from its input tape.
3. For each $y_{I_j} = k$ in $[n]$ yielding a full string y,
   a. Computes H_{j-1}, with bit $i < j$ found by looking in position k in the ith table output by leak in step 4 and $i \geqslant j$ found by taking the appropriate bit of w, also output by leak.
   b. Computes $D_x^m(H_{j-1}) \oplus b \oplus w_j$ and writes it to the output tape.

We will show that the result is a pair which runs in the necessary time and space bounds and, with high probability, is a good enough approximation of $f(x)$ to achieve contradiction.

Space-bound. leak step 1 requires space $S_{\mathsf{GenDesign}} = \mathcal{O}(d) = O\left(\frac{\log n}{\alpha}\right)$. Since it is locally computed by future steps, this becomes $\mathcal{O}(S_{\mathsf{GenDesign}})$. The second step requires space $d + \log m$, and uses the local computation of step 1. The third step requires space $\log(\log m + d)$, since it is simply iteration and printing. The fourth step uses space $\log m + d$ again since it iterates over indices i and values of y. For each value, it simply performs a lookup into the input, which requires at most space equal to the size of the index, bounded by $d + 1$. Finally, step 5 requires space $\log m$, since it needs to keep track of the number of bits it needs to write.

Thus, the overall space used by leak is $\mathcal{O}(d) + \mathcal{O}(\log n) + \mathcal{O}(\log m) = \mathcal{O}(\log n + \log \alpha) = \mathcal{O}(\log n)$.

For A, step 1 again requires space $\mathcal{O}(S_{\mathsf{GenDesign}})$. Step 2 is simply a definition and thus requires no space. Step 3 requires $\log n$ bits of iteration overhead plus the space used by 3.(a) and 3.(b). 3.(a) is simply indexing into the input based on k and so requires space $\log n$. This is the locally recomputed by step 3.(b), so 3.(a) requires space $\mathcal{O}(\log n)$. Finally, 3.(b) requires space $C \log m$.

Thus, A and leak run in space $c_2 \log n$ for a constant c_2 dependent only on C.

Time-bound.   Due to space constraints, the straightforward time complexity analysis can be found in the full version [12].

Contradiction of hardness. First, note that leak outputs at most $\log m + d - r + m + m2^{2\alpha^2 d}$ bits. The $\log m$ and $d - r$ terms are logarithmic and thus fall within any polynomial bound. The constraints on d and α, chosen such that $d\alpha = \log n$ and $2^{\frac{\alpha^4 d}{5}} = m$ tell us that for the $m2^{2\alpha^2 d}$ term, we have that

$$2\alpha^2 d = \frac{10}{\alpha^2}\log m \quad \text{so} \quad 2^{2\alpha^2 d} = m^{\frac{10}{\alpha^2}}$$

Thus, $m2^{2\alpha^2 d} = m^{\frac{10}{\alpha^2}+1}$, which is

$$m^{\frac{10}{\alpha^2}+1} = \left(n^{\frac{\alpha^3}{5}}\right)^{\frac{10}{\alpha^2}+1} = n^{2\alpha + \frac{\alpha^3}{5}}$$

Thus, if α is small enough (e.g., $\frac{1}{3}$), then for $\varepsilon = 2\alpha + \frac{\alpha^3}{5} < 1$, we have that $|\mathsf{leak}(x, f(x))| \leqslant n^\varepsilon$.

Observe that (2) is satisfied with probability $\frac{\beta}{2m}$ by leak's random choice of j, $y_{[d]\setminus I_j}$, and $w_{[m]\setminus[j]}$. Furthermore, with probability $1/4$, leak's random choices of b and w_j are the correct choices such that (3) holds. Thus, with probability $\frac{\beta}{8m}$, we have that (3) holds for our choices.

Consider the case where it does. Note that the randomness in (3) is over the argument to h, which is the index into $f(x)$. Then, when A evaluates $D_x^m(H_{j-1}) \oplus b \oplus w_j$ for each index, we conclude that at least a $\frac{1}{2} + \frac{\beta}{2m}$ fraction of bits output by A agree with $f(x)$. We thus have that

$$\mathbb{P}\left[d_H(A(x, \mathsf{leak}(x, f(x))), f(x)) \leqslant \left(\frac{1}{2} - \frac{\beta}{2m}\right)n\right] \geqslant \frac{\beta}{8m}$$

However, note that $\frac{\beta}{8m} > \frac{1}{n}$ for sufficiently large n (since m is polynomially smaller than n) and $\frac{1}{2} - \frac{\beta}{2m} < \frac{1}{2} - \frac{1}{m^2}$ for any constant β and sufficiently large n. This contradicts the hardness assumption of f, so such a distinguisher can't exist and we must have that G is a $C\log m$-secure PRG. ◀

4.2 Derandomization of search problems using the PRG

Now, we prove that the PRG is sufficient. This formulation is similar to work by Goldreich [4] which was seen in Liu and Pass's [8] reproduction.

▶ **Lemma 4.7** (Derandomization of search problems using the PRG)**.** *The existence of a* $3\log m$*-secure* $(\mathsf{poly}(m), \mathcal{O}(\log m))$*, logspace-computable PRG implies (2) of main theorem (3.1).*

As is typical for derandomization using a PRG, we will need the ability to pad strings. We will treat functions $\mathsf{pad}(x, k)$ and $\mathsf{unpad}(x')$ as having "negligible" complexity. The exact definition and proof of this can be seen in the "efficient padding" lemma of the full version [12], but we effectively use an expanded alphabet (encoded as pairs of bits) to add an "pad" character.

We separate our proof of 4.7 into two parts: derandomizing $\mathsf{prBP}\cdot\mathsf{L}$ using a PRG and then derandomizing $\mathsf{prBP}\cdot\mathsf{L}$ search problems using both $\mathsf{prBP}\cdot\mathsf{L} = \mathsf{prL}$ and the PRG.

▶ **Lemma 4.8** (Derandomization of promise problems)**.** *The existence of a* $3\log m$*-secure* $(\mathsf{poly}(m), \mathcal{O}(\log m))$*, logspace-computable PRG implies* $\mathsf{prBP}\cdot\mathsf{L} = \mathsf{prL}$*.*

Proof sketch. This is an adaptation of the time-bounded derandomization construction as seen in Goldreich's and Liu and Pass's work. The detailed working is omitted due to space constraints, but can be found in the full version [12] to verify that there are no hiccups related to logarithmic space. (In particular, the unpadded case becomes time $\mathcal{O}(k)$ and space $\log k + \mathcal{O}(1)$, and the padding exponent becomes the maximum of the exponent on time and the multiplier on space.) ◀

Proof of 4.7. Suppose G is an $3 \log m$-secure (n, d) PRG with parameters $n = m^{\theta}$ for some θ and $d = \mathcal{O}(\log m)$. By lemma 4.8, we have that $\mathsf{prBP\cdot L} = \mathsf{prL}$.

Suppose $R_{\mathrm{YES}}, R_{\mathrm{NO}}$ is a $\mathsf{prBP\cdot L}$ search problem with verifier $V(x, y; \omega)$ and finder $F(x; \gamma)$ where $|\gamma| \leqslant t(|x|)$. Observe that we can derandomize V to $V(x, y)$ (since it is a $\mathsf{prBP\cdot L}$ decision problem) such that $(x, y) \in R_{\mathrm{NO}} \implies V(x, y) = 0$ and $(x, y) \in R_{\mathrm{YES}} \implies V(x, y) = 1$.

First we will perform this derandomization for search problems where $F(x; \gamma)$ has time complexity $t(k) = \mathcal{O}(k)$ and the random procedure $M(x; \gamma) = V(x, F(x; \gamma))$ has space complexity $\log k + \mathcal{O}(1)$. Define $m = t(k)$ as the maximum amount of randomness used.

Consider the following deterministic procedure F': On input x,

1. For each seed $s \in \{0, 1\}^d$,
   - Compute $b \leftarrow V(x, F(x; G_{x'}^m(s)))$, where $x' = \mathsf{pad}(x, n)$.
   - If $b = 1$, output $F(x; G_{x'}^m(s))$.
2. Output the empty string.

Note that this procedure operates in logspace since iterating seeds requires space $d = \mathcal{O}(\log m) = \mathcal{O}(\log k)$ and step 3.(a) and 3.(b) are logspace compositions (a more detailed treatment can be see in the proof of 4.8).

Observe also that if this procedure terminates within the loop, it outputs a string y such that $V(x, y) = 1$ and thus such that $(x, y) \notin R_{\mathrm{NO}}$. It remains to show that this procedure terminates within the loop on all but finitely many inputs (since we can hard-code any finite number of misbehaving cases).

Suppose the contrary. Then there exist infinitely many inputs for which,

$$\forall s \in \{0, 1\}^d \quad V(x, F(x; G_{x'}^m(s))) = 0$$

Further, by definition of a finder, $\mathbb{P}_{\gamma}[(x, F(x; \gamma)) \in R_{\mathrm{YES}}] \geqslant \frac{2}{3}$. Thus,

$$\mathbb{P}_s[V(x, F(x; G_{x'}^m(s))) = 1] = 0 \quad \text{and} \quad \mathbb{P}_{\gamma}[(x, F(x; \gamma)) \in R_{\mathrm{YES}}] \geqslant \frac{2}{3}$$

This tells us that $D(1^m, x', r) = M(\mathsf{unpad}(x); r) = M(x; r)$ is a distinguisher with advantage $\beta \geqslant \frac{2}{3}$ which works for infinitely many values of x' and runs in space $2 \log m$ (since it is the truncation machine composed with M; again, a more detailed treatment can be seen in the proof of 4.8). This contradicts the security of the generator.

Thus, $(x, F'(x)) \notin R_{\mathrm{NO}}$ for all but finitely many x (which can be hard-coded).

Now consider an arbitrary $\mathsf{prBP\cdot L}$ search problem, $(R_{\mathrm{YES}}, R_{\mathrm{NO}})$ with derandomized verifier V and finder $F(x; \gamma)$ such that the time complexity of F is $\mathcal{O}(k^a)$ and the space complexity of $V(x, F(x; \gamma))$ is $b \log k + \mathcal{O}(1)$. Again, let $c = \max(a, b)$ and define $x' = \mathsf{pad}(x, k^c)$. We create $R_{\mathrm{YES}\mathsf{pad}} = \{(x', y) : (x, y) \in R_{\mathrm{YES}}\}$ and $R_{\mathrm{NO}\mathsf{pad}} = \{(x', y) : (x, y) \in R_{\mathrm{NO}}\}$.

Then $V_{\mathsf{pad}}(x', y) = V(\mathsf{unpad}(x'), y)$, $F_{\mathsf{pad}}(x') = F(\mathsf{unpad}(x'))$ satisfy

$$M_{\mathsf{pad}}(x'; r) = V_{\mathsf{pad}}(x', F_{\mathsf{pad}}(x'; r))$$

being a machine with space complexity $\log k' + \mathcal{O}(1)$ and F_{unpad} having time complexity $\mathcal{O}(k')$. Thus, we have deterministic algorithm $F'_{\mathsf{pad}}(x')$ such that $(x', F'_{\mathsf{pad}}(x')) \notin R_{\mathrm{NO}\mathsf{pad}}$.

We can then define F' as the function which on input x computes $F'_{\mathsf{pad}}(\mathsf{pad}(x, k^c))$. Observe that if $(x, F'(x)) \in R_{\mathrm{NO}}$ then $(x', F'_{\mathsf{pad}}(x')) = (x', F'(x)) \in R_{\mathrm{NO\,pad}}$, which contradicts F'_{pad} being a derandomized finder for $R_{\mathrm{NO\,pad}}$.

Thus, we have achieved the desired partial derandomization of $\mathsf{prBP \cdot L}$ search problems. ◀

5 Derandomization implies existence of hard function

▶ **Theorem 5.1** (Derandomization implies existence). *Suppose that for any $\mathsf{prBP \cdot L}$ search problem $(R_{\mathrm{YES}}, R_{\mathrm{NO}})$, we can create a deterministic algorithm F' such that for any $x \in S_R$, we have that $(x, F'(x)) \notin R_{\mathrm{NO}}$.*

We will show that for any constants c_1, c_2 there exists a logspace computable function f such that f is almost-all-input $(T, S, \ell, d) = \left(n^{c_1}, c_2 \log n, n^\varepsilon, \left(\frac{1}{2} - \frac{1}{m^2}\right)n\right)$-leakage-resilient average hard where $\varepsilon = 2\alpha + \frac{\alpha^3}{5}$ and $m = n^{\frac{\alpha^3}{5}}$ for $\alpha = \frac{1}{3}$.

The "Random is hard" lemma is similar to claim 1 in section 3.1 of Liu and Pass's paper [8] but needs a much stronger claim than in the time-bounded setting, corresponding to the use of an average-case hardness assumption.

5.1 Random is hard

▶ **Lemma 5.2** (Random is hard). *For any probabilistic algorithms (A, leak) and for all $n \in \mathbb{N}$, $x \in \{0, 1\}^n$, it holds that*

$$\mathbb{P}[|\mathsf{leak}(x, r)| \leqslant \ell \wedge d_H(A(x, \mathsf{leak}(x, r)), r) < d] \geqslant \frac{1}{2n}$$

with probability at most $n \cdot 2^{n \cdot (H(d/n) - 1) + \ell + \mathcal{O}(1)}$ (where $H(p)$ is the binary entropy function $H(p) = p \log \frac{1}{p} + (1 - p) \log \frac{1}{1-p}$) over $r \sim \mathcal{U}_n$.

Proof. Consider any $n \in \mathbb{N}$, $x \in \{0, 1\}^n$, and any probabilistic algorithm A. We will show that for any *deterministic* function leak' that outputs at most ℓ bits,

$$\mathbb{P}_{r \sim \mathcal{U}_n}\left[\mathbb{P}\big[d_H(A(x, \mathsf{leak}'(x, r)), r) < d \big] \geqslant \frac{1}{2n} \right] \leqslant n \cdot 2^{n \cdot (H(d/n) - 1) + \ell + \mathcal{O}(1)} \tag{4}$$

The claim for any probabilistic algorithm leak follows immediately by letting leak' be leak with the best random tape fixed. To show (4), consider the set of "bad" r's defined as

$$B = \left\{ r \in \{0, 1\}^n : \exists w \in \{0, 1\}^\ell \text{ s.t. } \mathbb{P}[d_H(A(x, \mathsf{leak}(x, r)), r) < d] \geqslant \frac{1}{2n} \right\}$$

For any r, if there exists a leak' such that $|\mathsf{leak}'(x, r)| \leqslant \ell$ and $\mathbb{P}\big[d_H(A(x, \mathsf{leak}'(x, r)), r) < d \big] \geqslant \frac{1}{2n}$, then $r \in B$. Thus, the probability that $A(x, \mathsf{leak}(x, r))$ successfully approximates r is bounded by the probability that $r \in B$.

We now bound B. Observe that for a given choice of $w \in \{0, 1\}^{\leqslant \ell}$, there exists $2n$ Hamming balls (which, in the worst case, do not overlap) of radius d contained in B, since there are at most $2n$ Hamming balls which $g(x, w)$ can occupy with probability $\geqslant \frac{1}{2n}$.

Finally, note that a Hamming ball of radius d on strings of length n has volume

$$2^{nH(d/n) - \frac{1}{2} \log n + \mathcal{O}(1)} \leqslant 2^{nH(d/n) + \mathcal{O}(1)}. \quad \text{Thus,} \quad |B| \leqslant 2^{\ell+1} \cdot 2n \cdot 2^{nH(d/n) + \mathcal{O}(1)}$$

We can roll the 2 in $2n$ into the $\mathcal{O}(1)$ in the final exponent, along with the $+1$ in $\ell + 1$.

Thus, the probability that a randomly chosen r satisfies the condition in (4) is at most

$$\frac{|B|}{2^n} \leqslant n \cdot 2^{n(H(d/n) - 1) + \ell + \mathcal{O}(1)} \qquad \qquad ◀$$

5.2   Construction of hard function

Proof of theorem 5.1. We can construct a prBP·L search problem to represent computing our desired hard problem. This search problem is similar to the corresponding search problem from [8] with the changes corresponding directly to the differences between the time-bounded worst-case version of leakage-resilient hardness and the space-bounded average-case version presented in this work.

The search problem.   Note that, since we consider only efficiently and uniformly computable (A, leak), a given pair has some constant length. We can thus consider only pairs (A, leak) with descriptions of length at most $\log n$. For any given attacker pair, then, we will only skip it under our search problem in a finite number of cases, which is covered by the almost-all-input hardness.

Let R_{YES} be a binary relation such that $(x, r) \in R_{\mathrm{YES}}$ if both of the following are true:

1. $|x| = |r|$.
2. For all probabilistic (A, leak) such that $|A|, |\mathsf{leak}| \leqslant \log n$ (where $|A|$ (resp. leak) refers to the length of the description of the program A in some arbitrary encoding scheme), it holds that

$$\mathbb{P}\big[|\mathsf{leak}'(x, r)| \leqslant \ell \wedge d_H(A'(x, \mathsf{leak}'(x, r)), r) < d\big] < \frac{1}{2n} \qquad (5)$$

where A' and leak' are time-space-truncated versions of A and leak which are only executed until they run for n^{c_1} steps or try to use more than $c_2 \log n$ space (where c_1, c_2 are from theorem 5.1).

Let R_{NO} be a binary relation such that $(x, r) \in R_{\mathrm{NO}}$ if $|x| \neq |r|$ or for *at least one pair* of (A, leak) (within $\log n$ size bound), (5) doesn't hold with a $\frac{1}{n}$ lower bound instead of an $\frac{1}{2n}$ upper bound.

We then show this is a prBP·L search problem by finding a verifier and finder.

Verifier.   On input (x, r), the verifier immediately rejects if $|x| \neq |r|$. Otherwise, it enumerates all probabilistic machines (A, leak) such that $|A|, |\mathsf{leak}| \leqslant \log n$. For each one, V estimates the value

$$p_{A, \mathsf{leak}} = \mathbb{P}\big[|\mathsf{leak}'(x, r)| \leqslant \ell \wedge d_H(A'(x, \mathsf{leak}'(x, r)), r) < d\big]$$

by simulating $A'(x, \mathsf{leak}'(x, r))$ polynomially many times and recording whether the Hamming distance is at most d. If during this process, for any (A, leak), we have that the sample estimate $\hat{p}_{A, \mathsf{leak}}$ exceeds $\frac{3}{4n}$, V outputs 0 and terminates. Otherwise, the value $p_{A, \mathsf{leak}}$ is discarded for this attacker pair and estimated for the next attacker pair. If we iterate every attacker pair without terminating, we output 1 and terminate. By the Chernoff bound and the union bound, V will accept with high probability if $(x, r) \in R_{\mathrm{YES}}$ and reject with high probability if $(x, r) \in R_{\mathrm{NO}}$.

This step is exactly where two-way randomness becomes necessary. We can compute $A'(x, \mathsf{leak}'(x, r))$ in logspace since with two-way randomness of BP·L, we can compose functions. The rest is logspace, since computing Hamming distance can be done in a read-once manner by tallying number of agreeing bits, which requires $\log n$ bits, and keeping a tally of successes vs. total count represents the success probability in $\log \mathsf{num\_trials}$ space. Since only a polynomial number of trials is necessary, this is all logspace.

We can compute it in polynomial time, since we only need polynomially many samples and each of A' and leak' is time truncated to a polynomial time-bound.

Finder. On input x, the finder outputs a random string of the same length. Observe that for any fixed $x \in \{0,1\}^n$, by the "random is hard" lemma (5.2), and a union bound over choices of (A, leak), we conclude that $F(x)$ outputs an invalid witness with probability at most

$$n^2 \cdot n \cdot 2^{n(H(d/n)-1)+\ell+\mathcal{O}(1)}$$

When $\ell = n^\varepsilon = n^{2\alpha + \frac{\alpha^3}{5}}$ and $\frac{d}{n} = \frac{1}{2} - \frac{1}{m^2}$ with $m = n^{\frac{\alpha^3}{5}}$ and $\alpha = \frac{1}{3}$, this converges to 0, and thus satisfies the search problem requirement of error probability less than $\frac{1}{3}$ (except for perhaps some finite number of small samples where answers can be hard-coded).

Computing the hard function. By the derandomization assumption, there exists a deterministic, logspace function F' such that $(x, F'(x)) \notin R_{\mathrm{NO}}$.

Note that $F'(x) \notin R_{\mathrm{NO}}$, so $|F'(x)| = |x|$ and for all machines (A, leak) which run within the time and space bounds, (5) (with $\frac{1}{2n}$ replaced with $\frac{1}{n}$) holds for all but the finitely many strings too short to include the descriptions of A, leak.

Therefore, by construction, $f = F'$ is a logspace computable $(n^{c_1}, c_2 \log n, \ell, d)$-leakage-resilient average hard function as desired. ◄

6 Discussion

We suspect it is possible to reduce the derandomization part of our assumption to $\mathsf{prBP \cdot L} = \mathsf{prL}$, such that we do not explicitly need the separate partial derandomization of $\mathsf{prBP \cdot L}$ search problems. However, we were not able to identify such a reduction within the duration of this research.

───── **References** ─────

1 Sanjeev Arora and Boaz Barak. *Computational Complexity: a Modern Approach.* Cambridge University Press, 2009.

2 Lijie Chen and Roei Tell. Hardness vs randomness, revised: Uniform, non-black-box, and instance-wise. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 125–136, 2022. `doi:10.1109/FOCS52979.2021.00021`.

3 Dean Doron and Roei Tell. Derandomization with minimal memory footprint. In *Proceedings of the Conference on Proceedings of the 38th Computational Complexity Conference*, CCC '23, Dagstuhl, DEU, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.CCC.2023.11`.

4 Oded Goldreich. *In a World of P=BPP*, pages 191–232. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. `doi:10.1007/978-3-642-22670-0_20`.

5 Russell Impagliazzo and Avi Wigderson. P = bpp if e requires exponential circuits: derandomizing the xor lemma. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 220–229, New York, NY, USA, 1997. Association for Computing Machinery. `doi:10.1145/258533.258590`.

6 Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13:1–46, 2004. `doi:10.1007/s00037-004-0182-6`.

7 Adam R. Klivans and Dieter van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM Journal on Computing*, 31(5):1501–1526, 2002. `doi:10.1137/S0097539700389652`.

8   Yanyi Liu and Rafael Pass. Leakage-resilient hardness vs randomness. In *Proceedings of the Conference on Proceedings of the 38th Computational Complexity Conference*, CCC '23, Dagstuhl, DEU, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.CCC.2023.32`.

9   Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994. `doi:10.1016/S0022-0000(05)80043-1`.

10   Rafael Pass and Oren Renard. Characterizing the power of (persistent) randomness in log-space. Electronic Consortium on Computational Complexity, 2023. URL: `https://eccc.weizmann.ac.il/report/2023/211/`.

11   Ronald L. Rivest and Adi Shamir. Efficient factoring based on partial information. In Franz Pichler, editor, *Advances in Cryptology—EUROCRYPT '85, Workshop on the Theory and Applications of Cryptographic Techniques*, volume 219 of *Lecture Notes in Computer Science*, pages 31–34. Springer, April 1985. `doi:10.1007/3-540-39805-8_3`.

12   Yakov Shalunov. Leakage-resilient hardness equivalence to logspace derandomization. arXiv. `arXiv:2312.14023`.

13   Madhu Sudan, Luca Trevisan, and Salil Vadhan. Pseudorandom generators without the xor lemma. *Journal of Computer and System Sciences*, 62(2):236–266, 2001. `doi:10.1006/jcss.2000.1730`.

14   Salil P. Vadhan. Pseudorandomness. *Foundations and Trends® in Theoretical Computer Science*, 7(1–3):1–336, 2012. `doi:10.1561/0400000010`.

15   Andrew C. Yao. Theory and application of trapdoor functions. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pages 80–91, 1982. `doi:10.1109/SFCS.1982.45`.