Characterizing Derandomization Through Hardness of Levin-Kolmogorov Complexity

Yanyi Liu
Cornell Tech, New York, NY, USA

Rafael Pass
Cornell Tech, New York, NY, USA
Tel-Aviv University, Israel

Abstract

A central open problem in complexity theory concerns the question of whether all efficient randomized algorithms can be simulated by efficient deterministic algorithms. We consider this problem in the context of promise problems (i.e,. the prBPP v.s. prP problem) and show that for all sufficiently large constants c, the following are *equivalent*:

- prBPP = prP.
- For every $\mathsf{BPTIME}(n^c)$ algorithm M, and every sufficiently long $z \in \{0,1\}^n$, there exists some $x \in \{0,1\}^n$ such that M fails to decide whether $Kt(x \mid z)$ is "very large" ($\geq n-1$) or "very small" ($\leq O(\log n)$).

where $Kt(x \mid z)$ denotes the Levin-Kolmogorov complexity of x conditioned on z. As far as we are aware, this yields the first full *characterization* of when prBPP = prP through the hardness of some class of problems. Previous hardness assumptions used for derandomization only provide a one-sided implication.

2012 ACM Subject Classification Theory of computation → Pseudorandomness and derandomization

Keywords and phrases Derandomization, Kolmogorov Complexity, Hitting Set Generators

Digital Object Identifier 10.4230/LIPIcs.CCC.2022.35

Related Version *Full Version*: https://eccc.weizmann.ac.il/report/2022/084/

Funding *Rafael Pass*: Worked done while being on a sabbatical at Tel-Aviv University. Supported in part by NSF Award SATC-1704788, NSF Award RI-1703846, AFOSR Award FA9550-18-1-0267, and a JP Morgan Faculty Award. 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.

Acknowledgements We thank the anonymous reviewers for many helpful comments, and most notably for making us aware of [11].

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 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, and we here consider the question of whether prBPP = prP.

A long sequence of works originating with the works of Blum-Micali [5], Yao [29], Nisan [21], Nisan-Widgerson [22], Babai-Fortnow-Nisan-Wigderson [4], Impagliazzo-Wigderson [14] 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 [22, 14] 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 [13] show a *partial* converse: if $\mathsf{prBPP} = \mathsf{prP}$, then some non-trivial circuit lower bound also must 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 [26, 20] 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 is still 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:

Does there exist some hardness assumption that is equivalent to $\mathsf{prBPP} = \mathsf{prP}$*?*

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.

Circumventing this problem, an elegant result by Goldreich [10] – which will be instrumental in the current work – provides a characterization of $\mathsf{prBPP} = \mathsf{prP}$ through the existence of a generalized form of a pseudo-random generator (PRG). In more detail, Goldreich [10] shows that $\mathsf{prBPP} = \mathsf{prP}$ if and only if a certain type of a *targeted PRG* exists; 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. As noted by Goldreich, such targeted PRGs suffice to derandomize prBPP, where the instance to be decided can be used as the "target"; Goldreich next provides a construction of such a targeted PRG assuming that $\mathsf{prBPP} = \mathsf{prP}$. As pointed out by Goldreich, however, the existence of a PRG is not a "hardness" assumption, and thus his work does not provide a characterization of derandomization in terms of some hardness assumption.

As far as we are aware, the only work that that shows an *equivalence* between $\mathsf{prBPP} = \mathsf{prP}$ and some hardness assumption does so under a conjecture (a weaker version of the non-deterministic exponential-time hypothesis) [8]. Very recently, however, two intriguing works make progress on closing the gap between the necessary and sufficient assumptions for derandomization:

- Chen and Tell [9], relying on the work by Goldreich [10], present a new uniform hardness assumption – roughly speaking, that there exists a multi-output function f 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 inputs – which 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.
- A work by Hirahara [11] presents an equivalence between *hitting set generators* (HSG) [1, 2] with respect to some circuit class and the hardness of approximating a Kolmogorov complexity problem (in more detail, the Levin-Kolmogorov complexity problem) with the same circuit class. While his result is stated with respect to low-level complexity classes (such as $\mathsf{AC}^0 \circ \mathsf{XOR}$), his proof actually extends also to classes such as $\mathsf{P/poly}$. While HSG w.r.t., $\mathsf{P/poly}$ are known to imply that $\mathsf{prBPP} = \mathsf{prP}$, and they are a central tool toward establishing this in known results, they are not known to be implied by derandomization.

1.1 Characterizing prBPP = prP through the Hardness of Conditional Kt-complexity

In this work, we present a hardness assumption that is both *necessary* and *sufficient* for derandomizing prBPP. In more detail, we present an (in our eyes) natural class of promise problems such that prBPP = prP if and only if *every problem* in this class is (almost-everywhere) worst-case hard. The class of problems is related to Kolmogorov complexity.

Conditional Time-bounded Kolmogorov Complexity. What makes the string 121212 12121212121 less random than 604848506683403574 92? The notion of *Kolmogorov complexity* (K-complexity), introduced by Solomonoff [24], Kolmogorov [16] and Chaitin [7], provides a method for measuring the amount of "randomness" in individual strings: The K-complexity of a string is the length of the shortest program Π (to be run on some fixed universal Turing machine U) that outputs the string x. K-complexity, however, disregards the running time of the program Π. Levin-Kolmogorov Complexity [18] is an elegant way for incorporating the running time of the program Π into the complexity: $Kt(x)$ is defined as the minimal *cost* of a program Π that outputs x, where the cost of Π is defined as the sum of the length of Π and the *logarithm* of its running time.

We can also consider a conditional version of Kt-complexity: The *Conditional Levin-Kolmogorov Complexity* [30, 18, 27, 19] of a string x conditioned on a string z – denoted $Kt(x \mid z)$ – is the minimal "cost" of a program that, given the "auxiliary input" z (for free), outputs the string x.

We will here be interested in a *promise version* of the decisional conditional Kt-complexity problem, parametrized by thresholds T_{YES}, T_{NO} that specify on what Kt-complexities a decider needs to work. In more detail, the promise problem $\mathsf{GapMcKtP}[T_{\mathsf{YES}}, T_{\mathsf{NO}}]$ is defined as follows:

- YES instances: (x, z) such that $|x| = |z|$, and $Kt(x \mid z) \leq T_{\mathsf{YES}}(|x|)$.
- NO instances: (x, z) such that $|x| = |z|$, and $Kt(x \mid z) \geq T_{\mathsf{NO}}(|x|)$.

Note that this is a "gap" problem as we are not considering strings that have "intermediary" conditional Kt-complexity.

The Main Theorem. We are now ready to state our main theorem.

▶ **Theorem 1.** *There exists a constant c such that the following are equivalent:*

- $\mathsf{prBPP} = \mathsf{prP}$*;*
- *For all* $\mathsf{BPTIME}(n^c)$ *algorithm M, all sufficiently large $z \in \{0,1\}^n$, there exists some $x \in \{0,1\}^n$ such that M fails to decide whether $(x, z) \in \mathsf{GapMcKtP}[O(\log n), n-1]$.*

In other words, $\mathsf{prBPP} = \mathsf{prP}$ iff $\mathsf{GapMcKtP}[O(\log n), n-1]$ is hard to decide *for all* (sufficiently large) auxiliary inputs z (w.r.t., n^c time probabilistic algorithms).

Comparison with [11]. Let us start by comparing Theorem 1 with the results established by Hirahara [11]. As mentioned above, Hirahara characterizes hitting sets generators as opposed to derandomization, but the problem he considered is very related to the one we consider. More specifically, Hirahara consider the exact same promise problem, but without any conditioning/auxiliary inputs, and shows that hardness with respect to *circuits* (as opposed to uniform probabilistic algorithms as we do) implies HSGs that are hard with respect to the same class of circuits. He also used this result to characterize some non-trivial derandomization (i.e., $\mathsf{RTIME}[2^{\tilde{O}(\sqrt{n})}]$ into $\mathsf{DTIME}[2^{n-\tilde{\omega}(\sqrt{n})}]$ on feasibly generated inputs).

On a high level, Hirahara constructs a HSG by using the first $O(\log n)$ bits of the seed to select a program M, lets the output of the program determine a truthtable of a function f, and then relies on the Impagliazzo-Wigderon (IW) PRG [14] (applied to the second part of the seed) using f as the "hard function".[1] Hirahara shows that any attacker for such a HSG can be used to distinguish whether $Kt(x)$ is smaller than $O(\log n)$ or at least $n-1$.

Our construction of a targeted PRG relies on similar principles. Similarly to [11], we use the first $O(\log n)$ bits of the seed to select a program M, but instead of letting M operate not just on the empty input (as in [11]), we also let the program M access the target string z; in other words, we can think of this approach as using the target/instance to get a hard function, and next applying IW to this function. As we shall see, when doing this, we can show that any distinguisher for the targeted HSG that works given a target z can be used, together with the IW reconstruction procedure, to distinguish for *any* $x \in \{0,1\}^{|z|}$ whether $Kt(x \mid z)$ is small or large.

Comparison with [9]. It is also worthwhile to compare Theorem 1 with the result of Chen and Tell [9]. Most importantly, Theorem 1 present a full characterization of when $\mathsf{prBPP} = \mathsf{prP}$, whereas the result in [9] has a gap between the sufficient and necessary assumptions. Nevertheless, there are also similarities: The condition where we require hardness for *all* auxiliary inputs is closely related to the hardness condition in [9] which requires hardness for (almost) all inputs. Indeed, on a technical level, the reason these requirements arise are quite similar in both works, and our condition is inspired by [9]. On the other hand, our condition is also more complex than the one in [9] in that the input to our problem consists of two parts – the auxiliary input z and the instance x – and whereas we require hardness w.r.t. all sufficiently large z (just like [9]), we only require the algorithm to fail on *some* instance x (similarly to standard notions of worst-case hardness).

An alternative way of looking at our result is as presenting an *explicit* multi-output function F where the ith component of the output of F on input z is $Kt(i,z)$ such that almost-all-input hardness of $n - O(\log n)$ *approximating* F is equivalent to $\mathsf{prBPP} = \mathsf{prP}$. This condition differs from the one in [9] in that (1) we are considering an explicit function, rather that just any function, (2) the output length of the function is exponential (similar to [12]) whereas it is polynomial in [9], and (3) we require hardness of approximating the function, as opposed to computing it exactly.

Finally, we note that we actually prove an even stronger result: we do not actually require hardness of $\mathsf{GapMcKtP}$ w.r.t (almost) all auxiliary input strings to deduce that $\mathsf{prBPP} = \mathsf{prP}$. In fact, we show that for every γ, there exists a *universal* and *uniformly* computable sequence $\mathcal{Z}^\gamma = \{z_1, z_2, \ldots\}$ such that n^c-time hardness of $\mathsf{GapMcKtP}[\gamma \log n, n-1]$ w.r.t. the specific sequence $\mathcal{Z}^\gamma$ implies that $\mathsf{prBPP} = \mathsf{prP}$.

We also mention that we can characterize quasi-polynomial time derandomization of prBPP using the same problems, by changing the YES-threshold to $\mathsf{poly}\log n$. For technical reasons, our approach does not extend to subexponential-time derandomization.[2]

---

[1] Hirahara presented his construction in a somewhat different way. In more detail, his construction of the HSG is simply a "universal HSG" obtained by interpreting the seed as a program M that is executed. He then uses the [14] construction in the analysis of the HSG. For our purposes, the above alternative presentation where incorporating the IW PRG directly into the construction will be helpful.

[2] More precisely, our characterization only works for complexity classes $\mathcal{C}$ of running times T such that if $T \in \mathcal{C}$ then $T(T(\cdot)) \in \mathcal{C}$ as well. This follows from our use of [23, 17, 6].

1.2 Proof Overview

To prove Theorem 1, we prove each direction of the equivalence separately.

Hardness of GapMcKtP from prBPP = prP. The first direction involves showing the hardness of GapMcKtP assuming that prBPP = prP. This direction follows mostly leveraging Goldreich's [10] earlier result showing the existence of a targeted PRG assuming prBPP = prP. We here consider a notion of a targeted PRG that is essentially identical to the notion of a "targeted canonical derandomizer" defined by Goldreich, but generalizes/strengthens his notion in several ways; most notably, we consider randomized distinguishers that may have superlinear running time, whereas Goldreich restricts attention to *deterministic* distinguishers with *linear* running time. Nevertheless, we observe that the PRG constructed by Goldreich (with minor modifications) actually satisfies the notion of a targeted PRG that we consider. Additionally, we observe that this (slightly new) notion of a targeted PRG suffices for demonstrating hardness of GapMcKtP for all sufficiently large auxiliary inputs z – roughly speaking, any solver for GapMcKtP can break the PRG as random strings (most often) are NO-instances, and strings in the range of the PRG are YES-instances; the target of the PRG will here correspond to the auxiliary input z used for the GapMcKtP problem.

prBPP = prP from the Hardness of GapMcKtP. To prove the second direction, we first observe that by the result of Buhrman and Fortnow [6] (building on [23, 17]), it suffices to show that prRP = prP to deduce that prBPP = prP. (We remark that for this result to hold, it is crucial that we are considering promise problems and not languages). Thus, it will suffice to derandomize prRP. Next, we consider the notion of a targeted HSG (discussed above) and demonstrate how to construct such a targeted HSG assuming that GapMcKtP is hard for all sufficiently large auxiliary inputs z. As mentioned above, the construction relies on ideas similar to those employed by Hirahara [11] except that, similarly to [9], we are using the target/instance z to obtain a "hard function" that we can plug into the IW generator.

Finally, to weaken the assumption to require hardness of GapMcKtP with respect to a specific universal and uniformly computable sequence of auxiliary inputs, we observe more generally that for any candidate construction of a HSG, there exists some universal sequence of targets such that security of the HSG w.r.t. this target sequence implies security w.r.t. all target sequences; and furthermore, this target sequence can be computed (in exponential time).

2 Preliminaries and Definitions

We assume familiarity with basic concepts such as Turing machines, polynomial-time algorithms, and probabilistic algorithms and computational classes such as prBPP, prRP, and prP. We let $\mathcal{U}_n$ the uniform distribution over $\{0,1\}^n$. Given a string $x \in \{0,1\}^n$ and an index $j \in [n]$, we let $[x]_j$ denote the length-j prefix of x.

Let $S \subseteq \{0,1\}^*$ be a set. We say that S is *decidable* if there exists a Turing machine M such that for all $x \in \{0,1\}^*$, $x \in S$ iff $M(x) = 1$. Let $\mathcal{Z} = \{z_n\}_{n\in\mathbb{N}}$ be a sequence. We say that $\mathcal{Z}$ is *uniform* if there exists a Turing machine M such that for all $n \in \mathbb{N}$, $z_n = M(1^n)$. We say that a function $f : \mathbb{N} \to \mathbb{N}$ 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 $\mathsf{poly}(f(n))$.

2.1 Levin's Conditional Kolmogorov Complexity

We recall the notion of Levin-Kolmogorov complexity. Roughly speaking, the *Levin's Kolmogorov complexity* [16, 23, 27, 15, 18], $Kt(x \mid z)$, of a string $x \in \{0,1\}^*$ conditioned on a "auxiliary input" string $z \in \{0,1\}^*$ is the cost of the most "efficient" program Π such that $\Pi(z)$ outputs x in t steps, where the efficiency of Π is defined to be the sum of the length of Π and the logarithm of t. We proceed to the formal definition. Let U be some fixed Universal Turing machine that can emulate any Turing machine Π with polynomial overhead. Let $U(\Pi(z), 1^t)$ denote the output of $\Pi(z)$ when emulated on U for t steps.

▶ **Definition 2.** *For all $x \in \{0,1\}^*$ and $z \in \{0,1\}^*$, define*

$$Kt(x \mid z) = \min_{\Pi \in \{0,1\}^*, t \in \mathbb{N}} \{|\Pi| + \lceil \log t \rceil : U(\Pi(z), 1^t) = x\}$$

We will here focus on a promise version of the decisional minimum conditional Levin-Kolmogorov complexity problem, parametrized by thresholds $T_{\mathsf{YES}}, T_{\mathsf{NO}}$.

▶ **Definition 3** (GapMcKtP)**.** *Let $T_{\mathsf{YES}}, T_{\mathsf{NO}}$ be two threshold functions. The promise problem* $\mathsf{GapMcKtP}[T_{\mathsf{YES}}, T_{\mathsf{NO}}]$ *is defined as follows.*

- *YES instances: (x, z) such that $|x| = |z|$, and $Kt(x \mid z) \leq T_{\mathsf{YES}}(|x|)$.*
- *NO instances: (x, z) such that $|x| = |z|$, and $Kt(x \mid z) \geq T_{\mathsf{NO}}(|x|)$.*

▶ **Definition 4** (Deciding GapMcKtP)**.** *We say that a probabilistic machine M* **fails to decides whether** $(x, z) \in \mathsf{GapMcKtP}[T_{\mathsf{YES}}, T_{\mathsf{NO}}]$ *if either $Kt(x \mid z) \leq T_{\mathsf{YES}}(|x|)$ but $\Pr[M(x, z) = 0] > 1/3$ or $Kt(x \mid z) \geq T_{\mathsf{NO}}(|x|)$ but $\Pr[M(x, z) = 1] > 1/3$.*

We will consider two notions of hardness of deciding GapMcKtP: either when the auxiliary input is fixed to some particular sequence (one for each input length), or hardness with respect to almost all auxiliary inputs.

▶ **Definition 5** (Hardness of GapMcKtP)**.** *We say that* $\mathsf{GapMcKtP}[T_{\mathsf{YES}}, T_{\mathsf{NO}}]$ *is:*

- **hard for probabilistic T-time algorithms given the auxiliary input sequence $\mathcal{Z} = \{z_1, z_2, \ldots\}$** *if for all probabilistic T-time algorithms M, all sufficiently large n, there exists a string $x \in \{0,1\}^n$ such that M fails to decide whether $(x, z_n) \in \mathsf{GapMcKtP}[T_{\mathsf{YES}}, T_{\mathsf{NO}}]$.*
- **hard for probabilistic T-time algorithms on almost all auxiliary inputs** *if for all sufficiently large z, there exists some $x \in \{0,1\}^{|z|}$ such that M fails to decide whether $(x, z) \in \mathsf{GapMcKtP}[T_{\mathsf{YES}}, T_{\mathsf{NO}}]$.*

2.2 Targeted Pseudorandom Generator

We consider the notion of a *targeted pseudorandom generator (targeted PRG)*, which is a generalization of the notion of a *targeted derandomizer* due to Goldreich [10]. Roughly speaking, a targeted pseudorandom generator g takes a seed along with a "target" string z as input, and we require that its output is indistinguishable from uniform by (computationally-bounded) distinguishers that additionally 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. We highlight that our notion slightly generalizes the notion of Goldreich by allowing the length of the target string to be different than the length of the output of the PRG, and additionally, we require the PRG to be defined over all output lengths. Furthermore, it strengthens Goldreich's notion by considering randomized distringuishers that may have superlinear running time (whereas Goldreich restricts attention to deterministic distinguishers with linear running time).

▶ **Definition 6** (Targeted pseudorandom generator (generalizing [10])). *Let $g : 1^n \times \{0,1\}^{\ell(n)} \times \{0,1\}^{m(n)} \to \{0,1\}^n$ be an efficiently 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 probabilistic 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 $z \in \{0,1\}^{\ell(n)}$, it holds that*

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

2.3 Targeted Hitting Set Generator

We turn to introducing the notion of *hitting set generator (HSG)* [1, 2] that we rely on. Recall that a standard hitting set generator requires its image set to have an overlap with any dense set that can be accepted by a small circuit. However, we here restrict our attention to uniform deterministic machines and we will consider the targeted variant of HSGs [10] (see also [9]).

▶ **Definition 7** (Targeted hitting set generator). *Let $g : 1^n \times \{0,1\}^{\ell(n)} \times \{0,1\}^{m(n)} \to \{0,1\}^n$ be an efficiently computable function. We say that g is an $T(n)$-secure $(\ell(n), m(n))$-targeted hitting set generator (T-secure (ℓ, m)-targeted HSG) secure w.r.t. deterministic attackers 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 $z \in \{0,1\}^{\ell(n)}$, it holds that if*

$$\Pr[x \leftarrow \{0,1\}^n : D(1^n, z, x) = 1] \geq \frac{1}{6}$$

then

$$\Pr[s \leftarrow \{0,1\}^{m(n)} : D(1^n, z, g(1^n, z, s)) = 1] > 0$$

For any targeted HSG g, we say that g is $O(T(n))$-secure if for all constant $c > 0$, g is $(cT(n))$-secure.

In addition, we will also consider a weaker notion of the targeted hitting set generator, where security is only guaranteed given some particular sequence of target inputs (rather than for all target inputs). Formally, let $\ell(n)$ be a function and let $\mathcal{Z} = \{z_n\}_{n\in\mathbb{N}}$ such that $|z_n| = \ell(n)$. Let $g : 1^n \times \{0,1\}^{\ell(n)} \times \{0,1\}^{m(n)} \to \{0,1\}^n$ be an efficiently computable function. We say that g is a *$T(n)$-secure $(\mathcal{Z}, m(n))$-targeted hitting set generator (T-secure $(\mathcal{Z}, m)$-targeted HSG)* if its security requirement holds for all sufficiently large $n \in \mathbb{N}$ and $z = z_n$.

It is well-known that a (non-uniformly) secure HSG can derandomize prRP. We next show that when considering a targeted uniformly-secure HSG, the same derandomization result still holds. This in essence follow by the standard proof (that non-uniformly secure HSG derandomize RP), but with an additional padding argument to deal with the "target"/auxiliary input.

▶ **Lemma 8.** *Assume that there exist constants $c \geq 1, \theta \geq 1$ and a $O(n^\theta)$-secure $(n^\theta, c \log n)$-targeted HSG $g : 1^n \times \{0,1\}^{n^\theta} \times \{0,1\}^{c \log n} \to \{0,1\}^n$ secure w.r.t. deterministic attackers. Then,* prRP = prP.

Proof. To show that prRP = prP, it suffices to prove that for any polynomial-time randomized algorithm A, there exists a polynomial-time deterministic algorithm B such that for all sufficiently long $x \in \{0,1\}^*$, if $\Pr_r[A(x;r) = 1] \geq \frac{1}{2}$, then $B(x) = 1$; and if $\Pr_r[A(x;r) = 0] = 1$, then $B(x) = 0$ (where r denotes the random coins that A uses).

Consider any poly-time randomized algorithm A. We can without loss of generality assume that A runs in linear time and A uses as many random coins as its input length.[3] If $\theta > 1$, the following padding argument is needed. For any string $x \in \{0,1\}^*$, let x' be the string $x10^{|x|^\theta - |x| - 1}$; that is, we pad as many '0' at the end of $x||1$ until it becomes of length $|x|^\theta$. Let A' be an algorithm such that $A'(x';r) = A(x;r)$ for any x, r.[4] (If $\theta = 1$, we let $x' = x$ and $A' = A$.)

We proceed to constructing a poly-time deterministic algorithm B that deterministically emulates A. On input an instance $x \in \{0,1\}^n$, $B(x)$ tries all possible seeds $v \in \{0,1\}^{c \log n}$ and $B(x)$ outputs 1 if and only if there exists a seed v such that $A'(x', g(1^n, x', v)) = 1$; otherwise $B(x)$ outputs 0.

Observe that if $A(x,r)$ outputs 0 with probability 1 (over the random choice of r), $A'(x',r)$ will output 0 with probability 1 and thus $B(x)$ will also output 0. Also note that B runs in polynomial time.

We show that, for all sufficiently long x, $B(x)$ will output 1 if $A(x,r)$ outputs 1 with probability $\geq \frac{1}{2}$ Since g is a $O(n^\theta)$-secure $(n^\theta, c \log n)$-targeted HSG and A' runs in deterministic $O(n^\theta)$ 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 if

$$\Pr[r \leftarrow \{0,1\}^n : A'(x', r) = 1] \geq \frac{1}{6} \tag{1}$$

then

$$\Pr[v \leftarrow \{0,1\}^{c \log n} : A'(x'; g(1^m, x', v)) = 1] > 0. \tag{2}$$

Consider some string x such that g is secure on auxiliary input x' (with respect to A') and it holds that $\Pr_r[A(x;r) = 1] \geq \frac{1}{2}$. It follows that $\Pr_r[A'(x';r) = 1] \geq \frac{1}{2}$ which implies that Equation 1 holds. Therefore, by the hitting property of g, Equation 2 also holds and there exists a seed $v \in \{0,1\}^{c \log n}$ such that $A'(x', g(1^n, x', v)) = 1$. Thus, $B(x)$ will output 1. Finally note that g will be secure on all sufficiently long x', so B can always find a seed v such that $A'(x', g(1^n, x', v)) = 1$ if $\Pr_r[A(x,r) = 1] \geq \frac{1}{2}$ for all sufficiently long x. ◀

3 Universal Target Strings for Targeted HSG or PRG

In this section, we show a useful statement about targeted PRGs/HSGs: For every candidate PRG/HSG, there exists a *universal* sequence of targets (one for each input lenght) such that if the PRG/HSG is secure with respect to this sequence, then it will be secure with respect to any target. Furthermore, this universal sequence is computable (in exponential time). Looking ahead, this will later be useful to us to show that hardness of GapMcKtPwith respect to a particular, computable, sequence of auxiliary inputs, suffices to characterize derandomization.

▶ **Lemma 9.** *Let $g : 1^n \times \{0,1\}^{\ell(n)} \times \{0,1\}^{m(n)} \to \{0,1\}^n$ be an efficiently computable function such that ℓ, m are polynomially bounded, and let $T(n) \leq 2^n$ be a function. There exists an exponential time uniform sequence $\mathcal{Z} = \{z_n \in \{0,1\}^{\ell(n)}\}_{n \in \mathbb{N}}$ such that if g is a T-secure $(\mathcal{Z}, m(n))$-target HSG (resp PRG) secure on $\mathcal{Z}$, then g is a T-secure $(\ell(n), m(n))$-target HSG (resp PRG) secure on all target inputs.*

[3] If A does not run in linear time (or uses more random coins than its input length), we can pad the input of A so that the padded version (of A) now runs in linear time (or uses equally many random coins).

[4] More formally, A' is an algorithm proceeding as the following. A' takes input (x', r) and then removes as many '0' in the end of x' as it can. A' further remove a single bit '1' and denote the result by x. A' returns $A(x, r)$.

Proof. Let g, T be as in the lemma statement. We consider the Turing machine M that proceeds as follows. On input 1^n, $M(1^n)$ enumerates all TMs D of (description) length $\leq \log n$ in lexicographic order. $M(1^n)$ verifies whether the following two conditions are satisfied.

- $D(y_1, y_2, y_3)$ terminates within $T(|y_1|)$ steps for all strings y_1, y_2, y_3 satisfying $|y_1| \leq n, |y_2| = \ell(|y_1|), |y_3| = |y_1|$.
- There exists a string $x \in \{0,1\}^{\ell(n)}$ such that D breaks g on the target input x. Specifically, if g is a HSG candidate, it requires that

$$\Pr[s \leftarrow \{0,1\}^{m(n)} : D(1^n, x, g(1^n, x, s)) = 0] = 1$$

and

$$\Pr[r \leftarrow \{0,1\}^n : D(1^n, x, r) = 1] > \frac{1}{6}.$$

Let D' be the first machine M finds such that the above two checks are passed. Let x' be the lexicographically smallest string such that the second condition above is satisfied (with respect to D'). $M(1^n)$ will simply output x'. Finally, let $\mathcal{Z} = \{z_n\}$ be a sequence such that for all $n \in \mathbb{N}$, $z_n = M(1^n)$.

We next argue that if g is a T-secure $(\mathcal{Z}, m(n))$-target HSG (resp PRG) secure on $\mathcal{Z}$, then g is a T-secure $(\ell(n), m(n))$-target HSG (resp PRG) secure on all target inputs, which concludes our proof (since M also runs in exponential time). Assume for contradiction that there exists a T-time distinguisher D such that D breaks g on infinitely many target inputs. Let D^* be such a distinguisher with the lexicographically smallest description. Consider all TMs D' that are lexicographically smaller than D^*, and the following two observations will show that $M(1^n)$ will never accept any TM $D' <_{\mathsf{lex}} D^*$ when n is sufficiently large.

- If D' is not a T-time machine. Then there exists an input of the form y_1, y_2, y_3 to D' such that $D'(y_1, y_2, y_3)$ runs more than $T(|y_1|)$ steps before it halts. $M(1^n)$ will not accept D' if $n > |y_1|$.
- If D' is a T-time machine, since $D' <_{\mathsf{lex}} D^*$, D' will only break g on finitely many target inputs (if any). $M(1^n)$ will not accept D' if n is sufficient large (so that D' never breaks g on target inputs of length larger than n).

On the other hand, note that $M(1^n)$ will accept D^* if D^* breaks g on some target input of length n (which happens infinitely often). Whenever $M(1^n)$ accepts D^*, D^* will break g on target z_n since $z_n = M(1^n)$. Therefore, we conclude that D^* breaks the security of g on infinitely many z's $\in \mathcal{Z}$. ◀

4 Main Theorem

We are now ready to formally state out main theorem.

▶ **Theorem 10.** *There exist a constant $c \geq 1$ and a Turing machine M such that the following are equivalent.*

1. $\mathsf{prBPP} = \mathsf{prP}$.
2. *The existence of a constant γ_0 such that for all $\gamma \geq \gamma_0$, $\mathsf{GapMcKtP}[\gamma \log n, n-1]$ is hard for probabilistic n^c-time algorithms on almost all auxiliary inputs.*
3. *The existence of a constant γ_0 such that for all $\gamma \geq \gamma_0$, all uniform auxiliary sequence $\mathcal{Z}$, it holds that $\mathsf{GapMcKtP}[\gamma \log n, n-1]$ is hard for probabilistic n^c-time algorithms given the sequence $\mathcal{Z}$.*

4. *The existence of a constant* γ *such that* GapMcKtP$[\gamma \log n, n-1]$ *is hard for probabilistic* n^c*-time algorithms given the auxiliary input sequence* $\mathcal{Z} = \{z_1, z_2, \ldots\}$ *where* $z_i = M(\gamma, 1^i)$.
5. *The existence of constants* $\sigma \geq 1, \theta \geq 1$ *and an* $O(n^\theta)$*-secure* $(n^\theta, \sigma \log n)$*-targeted HSG.*
6. prRP = prP

Proof. The proof of this theorem relies on many results which will be stated and proved later. Let c be the constant as in Lemma 21. Although it is stated in Lemma 21 that for each constant γ, there exists a uniform auxiliary input sequence $\mathcal{Z}_0 = \{z_{0,n}\}_{n\in\mathbb{N}}$ (such that some desired propertise are satisfied), its proof actually proves a stronger statement: There exists a machine M such that M with γ as input will generate the auxiliary sequence $\mathcal{Z}_0$ in the sense that $z_{0,n} = M(\gamma, 1^n)$ for all $n \in \mathbb{N}$. Given the existence of such c and M, our proof proceeds as the following.

(1) ⇒ (2): it follows from Theorem 11.
(2) ⇒ (3) and (3) ⇒ (4): These two implications trivially hold.
(4) ⇒ (5): This implication follows from Lemma 21 due to our choice of M and the way we pick $\mathcal{Z}$.
(5) ⇒ (6): The proof of this implication relies on the fact that a targeted HSG allows us to emulate prRP computation in deterministic time, and a detailed proof can be found in Lemma 8.
(6) ⇒ (1): This implication can be proved using standard reductions in [23, 17] (and see also [6]). ◀

5 GapMcKtP Hardness from prBPP = prP

In this section, we show that the assumption prBPP = prP will imply the hardness of GapMcKtP with the desired parameters.

▶ **Theorem 11.** *Assume that* prBPP = prP*. Then for all constants* $c \geq 1$*, there exists a constant* $\gamma_0 > 0$ *such that for all constants* $\gamma \geq \gamma_0$*, it holds that* GapMcKtP$[\gamma \log n, n-1]$ *is hard for probabilistic* n^c*-time algorithms on almost all auxiliary inputs.*

Recall that Goldreich [10] showed that if prBPP = prP, then there exists a so-called "targeted derandomizer". Since our notion of a targeted PRG is very similar to his notion, his proof extends with just minor modifications of the parameters also to our notion.

▶ **Theorem 12** (essentially implicit in [10])**.** *Assume that* prBPP = prP*. Then for all constants* $\gamma > 0, c \geq 1$*, there exists a* n^c*-secure* $(n, \gamma \log n)$*-targeted PRG* g*.*

We defer the proof of Theorem 12 (which closely follows [10]) to the full version. [10] further shows that a targeted PRG can be used to derandomize prBPP (and thus shows equivalence of derandomization of prBPP and targeted PRGs). We here instead show that the existence of targeted PRGs implies hardness of GapMcKtP. Roughly speaking, this follows from the observation that any solver for GapMcKtP can break the PRG as random strings (most often) are NO-instances, and strings in the range of the PRG are YES-instances; the target of the PRG will here correspond to the auxiliary input z used for the GapMcKtP problem.

▶ **Lemma 13.** *Assume that for all constants* $\gamma > 0, c \geq 1$*, there exists a* n^c*-secure* $(n, \gamma \log n)$*-targeted PRG. Then, for all constants* $c \geq 1$*, there exists a constant* $\gamma_0 > 0$ *such that for all constants* $\gamma \geq \gamma_0$*,* GapMcKtP$[\gamma \log n, n-1]$ *is hard for probabilistic* n^c*-time algorithms on almost all auxiliary inputs.*

Proof. Consider any constant $c \geq 1$. By our assumption, it follows that there exists a n^c-secure $(n, \log n)$-targeted PRG $g : 1^n \times \{0,1\}^n \times \{0,1\}^{\log n} \to \{0,1\}^n$. Let γ_0 be a constant such that computing the PRG $g(1^n, x, v)$, $x \in \{0,1\}^n, v \in \{0,1\}^{\log n}$ can be done in time $n^{\gamma_0 - 2}$. Let γ be any constant such that $\gamma \geq \gamma_0$. Assume for contradiction that $\mathsf{GapMcKtP}[\gamma \log n, n-1]$ is easy for probabilistic n^c-time algorithms on almost all auxiliary inputs. Then, there exist a n^c-time probabilistic machine M such that for infinitely many $n \in \mathbb{N}$, there exists $z_n \in \{0,1\}^n$ such that for all $x \in \{0,1\}^n$, $\Pr[M(1^n, x, z_n) = 1] \geq 0.9$ if $Kt(x \mid z_n) \leq \gamma \log |x|$ and $\Pr[M(1^n, x, z_n) = 1] \leq 0.1$ if $Kt(x \mid z_n) \geq |x| - 1$. We will show that $M(1^n, z_n, \cdot)$ distinguishes between $g(1^n, z_n, \mathcal{U}_{\log n})$ and $\mathcal{U}_n$ on all z_n on which M succeeds, which contradicts the security of g. Towards this, let us fix some $n \in \mathbb{N}$, $z = z_n$ on which M succeeds.

We first prove that on input $(1^n, z, g(1^n, z, v))$ where $v \in \{0,1\}^{\log n}$, $M(1^n, z, g(1^n, z, v))$ will output 1 with probability ≥ 0.9. Observe that

$$Kt(g(1^n, z, v) \mid z) \leq \log n + \log(n^{\gamma_0 - 2}) + O(1) \leq \gamma \log n$$

when n is sufficiently large since $g(1^n, z, v)$ can be computed by hardwiring the seed v (of length $\log n$) and the code of g (of length $O(1)$) in time $n^{\gamma_0 - 2}$ when having access to the string z. Therefore, $\Pr[M(1^n, z, g(1^n, z, v)) = 1] \geq 0.9$ for every $v \in \{0,1\}^{\log n}$.

We next show that on input $(1^n, z, \mathcal{U}_n)$, M will output 1 with probability ≤ 0.6. Observe that there are at most 2^{n-1} strings x of length n that have conditional Kt-complexity $\leq n-2$ since there are at most 2^{n-1} machines of description length $\leq n-2$. It follows that $\Pr[Kt(\mathcal{U}_n \mid z) \geq n-1] \geq 1 - \frac{2^{n-1}}{2^n} \geq \frac{1}{2}$. Conditioned on this event, we know that the probability that M outputs 1 is at most 0.1. Thus, by a union bound, $\Pr[M(1^n, z, \mathcal{U}_n) = 1] \leq \frac{1}{2} + \frac{1}{2} \times 0.1 \leq 0.6$. ◀

We can now conclude the proof of Theorem 11.

of Theorem 11. Theorem 11 follows directly from Theorem 12 and Lemma 13. ◀

6 Derandomization from Hardness of GapMcKtP

We proceed to proving that hardness of GapMcKtP implies that prBPP = prP. Note that by standard techniques in [23, 17], it suffices to derive prRP = prP. Towards this, we will present how to construct a targeted HSG from the assumption, which is known to enable us to derandomize prRP (see also Lemma 8).

6.1 Targeted HSG from Hardness of GapMcKtP

We here show how to obtain an targeted HSG assuming hardness of GapMcKtP. The following result is the crux of our proof.

▶ **Lemma 14.** *There exists a constant $c \geq 1$ such that the following holds. For each constant $\gamma > 0$, there exist constants $\sigma \geq 1, \theta \geq 1$, and an efficiently computable function $g : 1^m \times \{0,1\}^{m^\theta} \times \{0,1\}^{\sigma \log m} \to \{0,1\}^m$ such that for any target input sequence $\mathcal{Z}_1 = \{z_{1,m}\}_{m \in \mathbb{N}}$, if g is not an $O(m^\theta)$-secure $(\mathcal{Z}_1, \sigma \log m)$-targeted HSG, then $\mathsf{GapMcKtP}[\gamma \log n, n-1]$ is not hard for probabilistic n^c-time algorithms given an auxiliary input sequence $\mathcal{Z}_0 = \{z_{0,n}\}_{n \in \mathbb{N}}$ where for all $n \in \mathbb{N}$, $z_{0,n} = z_{1,m(n)}$ and $m(n) = \lfloor n^{\frac{1}{\theta}} \rfloor$.*

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

▶ **Definition 15** (List-decodable error correcting code (see e.g. [28])). *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 $(L, \frac{1}{2} - \delta)$*-list-decodable error correcting code *if there exists a function $\mathsf{Dec} : \{0,1\}^{n'} \to (\{0,1\}^n)^L$ 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 $x \in \mathsf{Dec}(x')$. We refer to Dec as a decoder of Enc.*

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

▶ **Theorem 16** ([25]; see also [28, Problem 5.2]). *There exist two deterministic polynomial time algorithms $\mathsf{Enc}, \mathsf{Dec}$ 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 $(\mathsf{poly}(1/\delta), \frac{1}{2} - \delta)$-list-decodable error correcting code with $\mathsf{Dec}_{n,\delta}$ being its decoder, and both $\mathsf{Enc}_{n,\delta}$ and $\mathsf{Dec}_{n,\delta}$ run in time $\mathsf{poly}(n, 1/\delta)$.*

Tool 2: The NW PRG. We turn to recalling the construction of the Nisan-Wigderson (NW) PRG [22]. 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 17** (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 $f : \{0,1\}^r \to \{0,1\}$ be a function. The $(\mathcal{I}, f)$-NW generator is the function $\mathsf{NW}^f_{\mathcal{I}} : \{0,1\}^d \to \{0,1\}^m$ that takes any string $y \in \{0,1\}^d$ as a seed and outputs*

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

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 18** (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 19** ([22]; see also [3, Lemma 16.18]). *There exists a deterministic algorithm $\mathsf{GenDesign}$ such that on input $d, r, s \in \mathbb{N}$ where $r > s$ and $d > 10r^2/s$, runs in $\mathsf{poly}(2^d)$ steps and outputs a (d, r, s)-design $\mathcal{I}$ containing $2^{s/10}$ subsets of $[d]$.*

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

▶ **Lemma 20** (Implicit in [22, 14]). *There exists a PPT algorithm $\mathsf{NWRecon}$ such that the following conditions hold.*

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

$$\left| \Pr[y \leftarrow \{0,1\}^d : D(\mathsf{NW}^f_{\mathcal{I}}(y)) = 1] - \Pr[w \leftarrow \{0,1\}^m : D(w) = 1] \right| \geq \frac{1}{6}. \tag{3}$$

- *Output: a (deterministic) program M of description length $\leq m \cdot 2^s + m + d + O(\log drsm)$ such that $M^D(\mathcal{I})$ will output a string $x' \in \{0,1\}^{2^r}$ in $\mathsf{poly}(2^r)$ steps and x' satisfies that*

$$\Pr[p \leftarrow [2^r] : x'_p \neq f(p)] \leq \frac{1}{2} - \frac{1}{12m}.$$

For the sake of completeness, we refer the read to the full version for a proof of Lemma 20.

Returning to the proof of Lemma 14. We are finally ready to prove Lemma 14 by relying on the above two tools/results.

Proof of Lemma 14. 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 HSG g that we hope to construct. Let n denote the length of GapMcKtP instances and let $\theta \in \mathbb{N}$ be a constant such that $\frac{1}{\theta}$ is sufficiently small. In this proof, we usually assume that $n = \mathsf{poly}(m)$ and it holds that $n = m^\theta$. In some cases depending on the context, m is defined w.r.t. n and it holds that $m = \lfloor n^{\frac{1}{\theta}} \rfloor$ (and we can think of m as being sublinear in n).

Let c be a sufficiently large constant (which will be fixed later). Consider any constant $\gamma > 0$ and a YES-threshold of GapMcKtP $T_{\mathsf{YES}} = \gamma \log n$.

Constructing the HSG. Our HSG will take as input a unary string 1^m, a target string z of length $m^\theta = n$, along with a seed. Let $\delta = O(\frac{1}{m})$ and we will need a list-decodable ECC that corrects a $\frac{1}{2} - \delta$ fraction of errors. By Theorem 16, there exists a function $L = \mathsf{poly}(1/\delta)$, a function $r = O(\log n)$, a $(L, \frac{1}{2} - \delta)$-list-decodable ECC $\mathsf{Enc}_{n,\delta}$ that produces codewords of length 2^r, and a decoding algorithm $\mathsf{Dec}_{n,\delta}$ that outputs a candidate message set of size L (if Dec succeeds). 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 HSG). To achieve this, we require a (d, r, s)-design $\mathcal{I}$ that contains m subsets of $[d]$. By Lemma 19, we can pick $s = O(\log m)$ to ensure that $\mathcal{I}$ contains m subsets, and pick $d = \Omega(\log n)$ to satisfy that $d > 10r^2/s$. (Such designs can be efficiently generated by GenDesign.) For our HSG to be secure, it is crucial in the NW security proof that 2^s is small enough, say, $< \sqrt{n}$ (which will also guarantee that $s < r$). This can be achieved by picking θ to be sufficiently large. (For a concrete choice of parameters, consider picking $s = 10 \log m$ and $\theta = 20$.)

We turn to describing our HSG formally. We will consider a function $g : 1^m \times \{0,1\}^n \times \{0,1\}^{\log n + T_{\mathsf{YES}} + d} \to \{0,1\}^m$ defined as follows. On input $(1^m, z, (j, \Pi', y))$ where $z \in \{0,1\}^n, j \in \{0,1\}^{\log n}, \Pi' \in \{0,1\}^{T_{\mathsf{YES}}}, y \in \{0,1\}^d$. Let k' be an integer that $k' = j$ when k' is represented in a binary string and let $k = \min\{k', T_{\mathsf{YES}}\}$. Let $\Pi = [\Pi']_k$ be the length-k prefix of Π'. Let $t = 2^{T_{\mathsf{YES}}}$. The algorithm g proceeds in the following steps.

1. g first interprets the string $\Pi \in \{0,1\}^k$ as a program and computes the output $x_\Pi = U(\Pi(z), 1^t)$ of $\Pi(z)$ after t steps.
2. Then g lets $f : \{0,1\}^r \to \{0,1\}$ be the function $f = \mathsf{fn}(\mathsf{Enc}_{n,\delta}(x_\Pi))$ that is associated with the truthtable $\mathsf{Enc}_{n,\delta}(x_\Pi) \in \{0,1\}^{2^r}$. ($g$ simply aborts if $|x_\Pi| \neq n$.)
3. Next, g invokes the design generating algorithm $\mathsf{GenDesign}(d, r, s)$ to obtain a (d, r, s)-design $\mathcal{I} = \{I_1, \ldots, I_m\}$.
4. Finally, g outputs

$$g(1^m, z, (j, \Pi', y)) = \mathsf{NW}^f_{\mathcal{I}}(y) = f(y_{I_1}) \ldots f(y_{I_m})$$

where the function NW is defined in Definition 17.

(Note that the seed length of g is $\log n + T_{\mathsf{YES}} + d = O(\log n) = O(\log m)$. And we can let σ be the constant such that the seed length of g is $\sigma \log m$. Notice that g is a function of the form $1^m \times \{0,1\}^{m^\theta} \times \{0,1\}^{\sigma \log m} \to \{0,1\}^m$.)

Deciding GapMcKtP. Suppose that g is not an $O(m^\theta)$-secure $(\mathcal{Z}_1, \sigma \log m)$-targeted HSG w.r.t. deterministic algorithms and some target string sequence $\mathcal{Z}_1 = \{z_{1,m} \in \{0,1\}^{m^\theta}\}_{m\in\mathbb{N}}$; then, there exists a $O(m^\theta)$-time deterministic distinguisher D such that for infinitely many $m \in \mathbb{N}$,

$$\Pr[v \leftarrow \{0,1\}^{\sigma \log m} : D(1^m, z_{1,m}, g(1^m, z_{1,m}, v)) = 0] = 1 \tag{4}$$

and

$$\Pr[w \leftarrow \{0,1\}^m : D(1^m, z_{1,m}, w) = 0] < 1 - \frac{1}{6} \tag{5}$$

We will prove that there exists a probabilistic n^c-time algorithm that decides $\mathsf{GapMcKtP}[T_{\mathsf{YES}}, n-1]$ infinitely often given the auxiliary input sequence $\mathcal{Z}_0$, where $\mathcal{Z}_0$ is a sequence of auxiliary input strings such that $z_{0,n} = z_{1,m}$ for all $n \in \mathbb{N}$. (We can think of Z_0 as being a padded version of $\mathcal{Z}_1$ to ensure that $z_{0,n} = z_{1,m}$.)

We will construct an algorithm A that runs in a-priori bounded polynomial time such that for any sufficiently large $m \in \mathbb{N}$, $n = m^\theta$, $z = z_{1,m} = z_{0,n}$, if Equation 4 and Equation 5 hold w.r.t. m, then for any $x \in \{0,1\}^n$, the following is true. If $Kt(x \mid z) \leq T_{\mathsf{YES}}$, then $A(x,z)$ outputs a program Π such that $|\Pi| \leq |x|^{2/3}$ and $\Pi(z)$ produces x within (a-priori bounded) poly time with high probability. Let us fix c to be some sufficiently large constant such that the running time of A (together with the time needed to check whether the output of A is correct) is bounded by n^c. Note that the existence of algorithm A will imply $\mathsf{GapMcKtP}[T_{\mathsf{YES}}, n-1]$ can be decided by a n^c-time algorithm on auxiliary input sequence $\mathcal{Z}_0$ since it suffices to first run $A(x,z)$ and check whether the program Π output by A indeed produces x on input z within some fixed polynomially amount of time. If $Kt(x \mid z) \geq |x| - 1$, it follows that there exists no such machine Π and A will never find it.

We proceed to describing the algorithm A. On input strings $x, z \in \{0,1\}^n$ (and let $m = \lfloor n^{\frac{1}{\theta}} \rfloor$), the algorithm A acts as the follows.

1. $A(x,z)$ lets $f : \{0,1\}^r \to \{0,1\}$ be the function $f = \mathsf{fn}(\mathsf{Enc}_{n,\delta}(x))$ that is associated with the truthtable $\mathsf{Enc}_{n,\delta}(x) \in \{0,1\}^{2^r}$.
2. $A(x,z)$ runs the design generating algorithm $\mathsf{GenDesign}(d,r,s)$ to obtain a (d,r,s)-design $\mathcal{I} = \{I_1, \ldots, I_m\}$.
3. $A(x,z)$ executes the NW reconstruction algorithm $\mathsf{NWRecon}^{D(1^m,z,\cdot)}(f, \mathcal{I})$ and let M denotes the program it outputs.
4. $A(x,z)$ evaluates $M^{D(1^m,z,\cdot)}(\mathcal{I})$ and denotes the output string by x'.
5. $A(x,z)$ computes a list $\vec{x}$ of size L by letting $\vec{x} = \mathsf{Dec}_{n,\delta}(x')$ (which is a list of candidate strings for x output by the list decoding algorithm), and let pos denote a coordinate of $\vec{x}$ such that $\vec{x}_{\mathsf{pos}} = x$. (If x does not appear in $\vec{x}$, $A(x,z)$ simply aborts.)
6. Finally, A outputs a program Π with the values $n, m, d, r, s, \delta^{-1}, \mathsf{pos}$, the code of M, D, Dec, $\mathsf{GenDesign}$ hardwired in it. In addition, on input z, $\Pi(z)$ proceeds in the following steps.
   a. $\Pi(z)$ first invokes $\mathsf{GenDesign}(d,r,s)$ to get a (d,r,s)-design $\mathcal{I} = \{I_1, \ldots, I_m\}$.
   b. $\Pi(z)$ computes $x' = M^{D(1^m,z,\cdot)}(\mathcal{I})$.
   c. $\Pi(z)$ runs the list decoding algorithm $\mathsf{Dec}_{n,\delta}(x')$ and obtains a list $\vec{x}$.
   d. $\Pi(z)$ outputs the string $\vec{x}_{\mathsf{pos}}$ and halts.

Analyzing the reduction. We turn to proving that the program Π will indeed output x on input z within polynomial time if n is of the form $n = m^\theta$ for some m such that Equation 4 and Equation 5 hold w.r.t. m, $z = z_{1,m}$, and $Kt(x \mid z) \leq T_{\mathsf{YES}}$. Fix some such x, z that are sufficiently long. We first show that program Π will indeed output x on input z. Since

$Kt(x \mid z) \leq T_{\mathsf{YES}}$, there exists a machine Π_x with $|\Pi_x| \leq T_{\mathsf{YES}}$ such that $\Pi_x(z)$ will output the string x within $2^{T_{\mathsf{YES}}} = t$ steps. Let $j = |\Pi_x|$ (in its binary representation) and let Π'_x be a string $\in \{0,1\}^{T_{\mathsf{YES}}}$ such that $[\Pi'_x]_j = \Pi_x$. Observe that for all $y \in \{0,1\}^d$, the string $\mathsf{NW}^f_{\mathcal{I}}(y)$ equals $g(1^m, z, (j, \Pi'_x, y))$ and thus $\mathsf{NW}^f_{\mathcal{I}}(y)$ is in the range of the HSG g. Note that $D(1^m, z, \cdot)$ is a HSG distinguisher and will always output 0 in the range of $g(1^m, z, \cdot)$. By Equation 4, it follows that

$$\Pr[y \leftarrow \{0,1\}^d : D(1^m, z, \mathsf{NW}^f_{\mathcal{I}}(y)) = 0] = 1$$

Combining this with Equation 5, it holds that $D(1^m, z, \cdot)$ distinguishes the output of $\mathsf{NW}^f_{\mathcal{I}}$ from uniform with advantage $\frac{1}{6}$ and thus it breaks the NW generator $\mathsf{NW}^f_{\mathcal{I}}$. Then by Lemma 20, the NW reconstruction algorithm $\mathsf{NWRecon}^{D(1^m,z,\cdot)}(f, \mathcal{I})$ will output a good approximation for f; that is, it holds that

$$\Pr[p \leftarrow [2^r] : \mathsf{Enc}_{n,\delta}(x)_p \neq x'_p] = \Pr[p \leftarrow [2^r] : f(p) \neq x'_p] \leq \frac{1}{2} - \frac{1}{12m} \leq \frac{1}{2} - \delta$$

So x' is relatively close to $\mathsf{Enc}_{n,\delta}(x)$. Since Enc is a good error correcting code, by Theorem 16, $\mathsf{Dec}_{n,\delta}(x')$ will return a list contain x; i.e., $x \in \vec{x} = \mathsf{Dec}_{n,\delta}(x')$. $A(x,z)$ will then find the coordinate pos such that $\vec{x}_{\mathsf{pos}} = x$. Note that $\Pi(z)$ will finally output $\vec{x}_{\mathsf{pos}}$ and we conclude that $\Pi(z)$ will indeed produce x.

We next argue that Π has a short description. Π spends $O(\log n)$ bits to store the values $n, m, d, r, s, \delta^{-1}$, the code of $D, \mathsf{Dec}, \mathsf{GenDesign}$. In addition, Π uses

$$m \cdot 2^s + m + d + \log(drsm) \leq n^{\frac{1}{\theta}} \cdot \sqrt{n} + n^{\frac{1}{\theta}} + O(\log n)$$

bits to save the code of M. Π takes $O(\log n)$ bits to hardwire pos since the list $\vec{x}$ is of size L, which is polynomial in n. Thus, the description length of Π is at most $O(n^{\frac{1}{\theta}} \cdot \sqrt{n}) < n^{2/3} = |x|^{2/3}$ (if n is sufficiently large).

We then prove that the running time of $\Pi(z)$ is a priori-bounded and polynomial in n. It takes $\mathsf{poly}(2^d)$ time to compute $\mathsf{GenDesign}(d, r, s)$, executing the program $M^{D(z,\cdot)}(\mathcal{I})$ takes $\mathsf{poly}(2^r) \cdot O(m^\theta)$ time (since the distinguisher D runs in $O(m^\theta)$ time). The list decoding algorithm runs in $\mathsf{poly}(n, 1/\delta)$, and finally to find $\vec{x}_{\mathsf{pos}}$ and output takes $O(nL)$. So the total running time of $\Pi(z)$ is at most an a-priori bounded polynomial in n. Combining this and the proofs given above, we reach the conclusion that Π is of length at most $|x|^{2/3}$ and $\Pi(z)$ indeed outputs x within a fixed number of steps.

It remains to show that the algorithm $A(x,z)$ runs in poly time. Note that A takes $\mathsf{poly}(n, 1/\delta)$ time for running $\mathsf{Enc}_{n,\delta}(x)$, $\mathsf{poly}(2^d)$ time for $\mathsf{GenDesign}(d, r, s)$, $\mathsf{poly}(2^r) \cdot O(m^\theta)$ time for $\mathsf{NWRecon}^{D(z,\cdot)}(f, \mathcal{I})$, $\mathsf{poly}(2^r) \cdot O(m^\theta)$ time for emulating $M^{D(z,\cdot)}(\mathcal{I})$, $\mathsf{poly}(n, 1/\delta)$ for $\mathsf{Dec}_{n,\delta}(x')$, and finally $O(nL)$ to locate x in $\vec{x}$. To sum up, A runs in polynomial time. ◀

As a summary, Lemma 14 shows that if $\mathsf{GapMcKtP}$ is hard given some particular auxiliary input sequence, then we can obtain a "partial" targeted HSG which is only secure on some sequence of targeted strings. We next show how to make use of this result to obtain a full-fledged targeted HSG.

▶ **Lemma 21.** *There exists a constant $c \geq 1$, and for each constant $\gamma > 0$, there exists an exponential time uniform auxiliary input sequence $\mathcal{Z}_0$ such that the following holds. If $\mathsf{GapMcKtP}[\gamma \log n, n-1]$ is hard for probabilistic n^c-time algorithms given auxiliary input $\mathcal{Z}_0$, then there exist constants $\sigma \geq 1, \theta \geq 1$ and a $O(m^\theta)$-secure $(m^\theta, \sigma \log m)$-target HSG.*

Proof. Let c be the constant guaranteed to exist by Lemma 14. Consider any constant $\gamma > 0$, and let σ, θ be the constants and $g : 1^m \times \{0,1\}^{m^\theta} \times \{0,1\}^{\sigma \log m} \to \{0,1\}^m$ be the efficiently computable function where σ, θ, g are guaranteed to exist by Lemma 14. Given g and the security bound of g (which is taken to be $O(m^\theta)$), let $\mathcal{Z}_1 = \{z_{1,n}\}_{n\in\mathbb{N}}$ be the exponential time uniform (universal) sequence of target strings we pick in Lemma 9. Let $\mathcal{Z}_0 = \{z_{0,n}\}_{n\in\mathbb{N}}$ be an auxiliary input sequence such that $z_{0,n} = z_{1,n^{1/\theta}}$ (which is a simply padded version of $\mathcal{Z}_1$). Notice that $\mathcal{Z}_0$ can also be produced by an exponential machine.

Assume that $\mathsf{GapMcKtP}[\gamma \log n, n-1]$ is hard for probabilistic n^c-time algorithms given auxiliary input $\mathcal{Z}_0$. Then, by (the contrapositive of) Lemma 14, g is a $O(m^\theta)$-secure $(\mathcal{Z}_1, \sigma \log m)$-target HSG with respect to the target string sequence $\mathcal{Z}_1$. Finally, by Lemma 9, g is a $O(m^\theta)$-secure $(m^\theta, \sigma \log m)$-target HSG secure on all target strings. ◀

References

1 Alexander E Andreev, Andrea EF Clementi, and Jose DP Rolim. A new general derandomization method. *Journal of the ACM (JACM)*, 45(1):179–213, 1998.

2 Alexander E Andreev, Andrea EF Clementi, José DP Rolim, and Luca Trevisan. Weak random sources, hitting sets, and bpp simulations. *SIAM Journal on Computing*, 28(6):2103–2116, 1999.

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

4 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.

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

6 Harry Buhrman and Lance Fortnow. One-sided versus two-sided error in probabilistic computation. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 100–109. Springer, 1999.

7 Gregory J. Chaitin. On the simplicity and speed of programs for computing infinite sets of natural numbers. *J. ACM*, 16(3):407–422, 1969.

8 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.

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

10 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.

11 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.

12 Rahul Ilango, Bruno Loff, and Igor Carboni Oliveira. NP-hardness of circuit minimization for multi-output functions. In *35th Computational Complexity Conference, CCC 2020*, pages 22:1–22:36, 2020.

13 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.

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

15 Ker-I Ko. On the notion of infinite pseudorandom sequences. *Theor. Comput. Sci.*, 48(3):9–33, 1986. `doi:10.1016/0304-3975(86)90081-2`.

16 A. N. Kolmogorov. Three approaches to the quantitative definition of information. *International Journal of Computer Mathematics*, 2(1-4):157–168, 1968.

17 Clemens Lautemann. BPP and the polynomial hierarchy. *Inf. Process. Lett.*, 17(4):215–217, 1983.

18 Leonid A. Levin. Universal search problems (russian), translated into English by BA Trakhtenbrot in [27]. *Problems of Information Transmission*, 9(3):265–266, 1973.

19 Luc Longpré and Sarah Mocas. Symmetry of information and one-way functions. In Wen-Lian Hsu and Richard C. T. Lee, editors, *ISA '91 Algorithms, 2nd International Symposium on Algorithms, Taipei, Republic of China, December 16-18, 1991, Proceedings*, volume 557 of *Lecture Notes in Computer Science*, pages 308–315. Springer, 1991. doi:10.1007/3-540-54945-5_75.

20 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.

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

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

23 Michael Sipser. A complexity theoretic approach to randomness. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 330–335. ACM, 1983.

24 R.J. Solomonoff. A formal theory of inductive inference. part i. *Information and Control*, 7(1):1–22, 1964. doi:10.1016/S0019-9958(64)90223-2.

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

26 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.

27 Boris A Trakhtenbrot. A survey of Russian approaches to perebor (brute-force searches) algorithms. *Annals of the History of Computing*, 6(4):384–400, 1984.

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

29 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.

30 A. K. Zvonkin and L. A. Levin. the Complexity of Finite Objects and the Development of the Concepts of Information and Randomness by Means of the Theory of Algorithms. *Russian Mathematical Surveys*, 25(6):83–124, December 1970. doi:10.1070/RM1970v025n06ABEH001269.