On One-way Functions from NP-Complete Problems

Yanyi Liu
Cornell University
yl2866@cornell.edu

Rafael Pass*
Cornell Tech
rafael@cs.cornell.edu

April 30, 2021

Abstract

We present the first natural NP-complete problem whose average-case hardness w.r.t. the uniform distribution over instances is *equivalent* to the existence of one-way functions (OWFs). The problem, which originated in the 1960s, is the *Conditional Time-Bounded Kolmogorov Complexity Problem*: let $K^t(x \mid z)$ be the length of the shortest "program" that, given the "auxiliary input" z, outputs the string x within time $t(|x|)$, and let $\mathsf{McKTP}[t, \zeta]$ be the set of strings (x, z, k) where $|z| = \zeta(|x|)$, $|k| = \log |x|$ and $K^t(x \mid z) < k$, where, for our purposes, a "program" is defined as a RAM machine.

Our main results shows that for every polynomial $t(n) \geq n^2$, there exists some polynomial ζ such that $\mathsf{McKTP}[t, \zeta]$ is NP-complete. We additionally extend the result of Liu-Pass (FOCS'20) to show that for every polynomial $t(n) \geq 1.1n$, and every polynomial $\zeta(\cdot)$, mild average-case hardness of $\mathsf{McKTP}[t, \zeta]$ is equivalent to the existence of OWFs.

*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.

1 Introduction

A one-way function (OWF) [DH76] is a function f that can be efficiently computed (in polynomial time), yet no probabilistic polynomial-time (PPT) algorithm can invert f with inverse polynomial probability for infinitely many input lengths n. Whether OWFs exist is unequivocally the most important open problem in Cryptography: OWFs are both necessary [IL89] and sufficient for many of the most central cryptographic primitives and protocols (e.g., pseudorandom generators [BM88, HILL99], pseudorandom functions [GGM84], private-key encryption [GM84], digital signatures [Rom90], commitment schemes [Nao91], identification protocols [FS90], coin-flipping protocols [Blu82], and more). These primitives and protocols are often referred to as *private-key primitives*, or "Minicrypt" primitives [Imp95] as they exclude the notable task of public-key encryption [DH76, RSA83].

While many candidate constructions of OWFs are known—most notably based on factoring [RSA83], the discrete logarithm problem [DH76], or the hardness of lattice problems [Ajt96]—the question of whether OWFs can be based on some "standard" complexity-theoretic assumption is mostly wide open. Indeed, a central open problem, originating in the seminal work of Diffie and Hellman [DH76] is whether the existence of OWFs can be based on the assumptions that $\mathsf{NP} \neq \mathsf{BPP}$, or even that NP is average-case hard (i.e., that there exists some language in NP that is hard-on-average). So far, however, most results in the literature have been negative. Notably, starting with the work by Brassard [Bra83] in 1983, a long sequence of works have shown various types of black-box separations between *restricted* types of OWFs (e.g., one-way permutations) and NP-hardness (see e.g., [Bra83, BT03, AGGM06, Pas06, GWXY10, Liv10, HMX10, BB15]). We emphasize, however, that these results only show limited separations: they either consider restricted types of one-way functions, or restricted classes of black-box reductions. Thus, even w.r.t. black-box reductions, the question of whether OWFs can be based on the assumption that $\mathsf{NP} \neq \mathsf{BPP}$, is wide open.

In fact, up until recently, the question of whether there exists some *natural* NP-language whose average-case hardness characterizes the existence of OWFs was open. Such a language was recently demonstrated by Liu and Pass [LP20]. More precisely, they demonstrated that for any polynomial $t(n) \geq 1.1n$, OWF exist if and only the *t-bounded Kolmogorov complexity problem*, $\mathsf{MKTP}[t]$, is mildly hard-on-average, where a language L is said to be *mildly hard-on-average* if there exists some polynomial $p(\cdot)$ such that no PPT heuristic $\mathcal{H}$ can decide L with probability $1 - 1/p(n)$ over random n-bit instances for infinitely many input lengths n. (We provide more details on the definition of $\mathsf{MKTP}[t]$ below.) $\mathsf{MKTP}[\mathsf{poly}(\cdot)]$ is contained in NP, but it is unknown whether this problem (which has been studied since the 1960s) is NP-complete. Indeed, this is one of the long-standing open problems in algorithmic information-theory [Ko91].

This leaves open the question of whether there exists some natural NP-*complete* language that characterizes the existence of OWFs:

> *Does there exists some "natural" NP-complete language L such that OWFs exist iff L is mildly hard-on-average?*

We note that "naturality" of the language L is key for this question to make sense: It is easy to modify $\mathsf{MKTP}[t]$ into a new "artificial" language L' which is both NP-complete, yet mild average-case hardness of L' is equivalent to mild average-case hardness of $\mathsf{MKTP}[t]$ (and thus equivalent to the existence of OWFs).[1]

[1]Simply consider the language L' of $2n$-bit instances $x||y$ where $x, y \in \{0,1\}^n$, and either (a) $x = 0^n$ and $y \in \mathsf{SAT}$, or (b) $x \neq 0^n$ and $y \in \mathsf{MKTP}[t]$. In other words, L' is a combination of SAT and $\mathsf{MKTP}[t]$, so clearly this language is NP-complete, but when considering uniform statements, we only hit SAT instances with negligible probability, and thus this language behaves essentially just like $\mathsf{MKTP}[t]$ on average.

The above question goes back to the work of Merkle and Hellman [MH78], who first attempted to base the security of cryptographic primitives on average-case hardness of NP-complete problems. While the original attempts failed to produce secure schemes (see [Odl90] for a survey), more recent approaches pioneered by Impaglizzo and Naor [IN89], Ajtai [Ajt96] and Ajtai and Dwork [AD97] produced not just OWFs but also more advanced cryptographic primitives (such as collision-resistant hash functions and public-key encryption) based on well-founded average-case hardness assumptions on the subset sum problem (which is NP-complete). However, it is not known whether the existence of OWFs implies average-case hardness of the subset sum problem (i.e., we only have a one-sided implication).

In this work, we provide a full resolution to the above-mentioned question. We identify the first natural NP-complete language L such that mild average-case hardness of L (with respect to the uniform distribution on instances) is equivalent to the existence of OWFs.

1.1 Our Results

Before describing our results in detail, let us first briefly recall the notion of Time-bounded Kolmogorov Complexity and the result of [LP20] that we will be relying on.

Time-bounded Kolmogorov Complexity and OWFs What makes the string 12121212121212121 less random than 60484850668340357492? The notion of *Kolmogorov complexity* (K-complexity), introduced by Solomonoff [Sol64], Kolmogorov [Kol68] and Chaitin [Cha69], provides an elegant method for measuring the amount of "randomness" in individual strings: The K-complexity of a string is the length of the shortest program (to be run on some fixed universal Turing machine U) that outputs the string x. The notion of *$t(\cdot)$-time-bounded Kolmogorov Complexity (K^t-complexity)* is a computationally-restricted version of K-complexity: $K^t(x)$ is defined as the length of the shortest program that outputs the string x within time $t(|x|)$. As surveyed by Trakhtenbrot [Tra84], the problem of efficiently determining the K^t-complexity for $t(n) = \mathsf{poly}(n)$ predates the theory of NP-completeness and was studied in the Soviet Union since the 60s as a candidate for a problem that requires "brute-force search". The modern complexity-theoretic study of this problem goes back to Sipser [Sip83], Ko [Ko86] and Hartmanis [Har83]. Let $\mathsf{MKTP}[t(\cdot)]$ denote the decisional $t(n)$-time bounded Kolmogorov complexity problem; namely, the language of pairs (x, k) where $|k| = \lceil \log |x| \rceil$ and $K^t(x) \leq k$.

As mentionned above, Liu and Pass [LP20] demonstrated that for every polynomial $t(n) \geq 1.1n$, mild average-case hardness of $\mathsf{MKTP}[t]$ is equivalent to the existence of OWFs. But as mentionned, it is not known whether $\mathsf{MKTP}[t]$ is NP-complete (for any polynomial t). Towards getting a characterization of OWFs based on average-case hardness of an NP-complete problem, we will consider a generalization of $\mathsf{MKTP}[t]$ based on *conditional* Kolmogorov complexity.

Conditional Time-bounded Kolmogorov Complexity The *$t(\cdot)$-time-bounded Conditional Kolmogorov Complexity* [ZL70, Lev73, Tra84, LM91] of a string x conditionned on the string z—denoted $K^t(x \mid z)$—is the length of the shortest program that, given the "auxiliary input" z, outputs the string x within time $t(|x|)$. More formally,

$$K^t(x \mid z) = \min_{\Pi \in \{0,1\}^*} \{|\Pi| : U(\Pi(z), 1^{t(|x|)}) = x\},$$

where U is a universal Turing machine, and we let $U(\Pi, 1^t)$ denote the output of the program Π after t steps. Whereas the notion of a "program" typically is taken to be a Turing machine, in this work we focus on the setting where a program is taken to be a RAM-machine—namely Π is now allowed to be a RAM-machine that can make Random Access queries into the auxiliary string z. Let $\mathsf{McKTP}[t(\cdot), \zeta(\cdot)]$ denote the decisional $t(\cdot)$-time bounded $\zeta(\cdot)$-conditional Kolmogorov complexity problem; namely, the language of triples (x, z, k) where $|z| = \zeta(|x|)$, $|k| = \lceil \log |x| \rceil$ and $K^t(x \mid z) \leq k$. Whereas conditional (time-bounded) Kolmogorov complexity has been studies for decades (see e.g., [LM91]), it has also remained an open question to determine whether this problem is NP-complete.[2]

We observe that the result of [LP20] extends, with only relatively minor modifications in the proof, also to conditional Kolmogorov complexity: We show that for every polynomial $t(\cdot) \geq 1.1n$, and every polynomial $\zeta(\cdot)$, mild average-case hardness of $\mathsf{McKTP}[t, \zeta]$ is equivalent to the existence of OWFs.

Theorem 1.1 (closely following [LP20])**.** *For every polynomial $t(n) \geq 1.1n$, every polynomial $\zeta(\cdot)$, mild average-case hardness of $\mathsf{McKTP}[t, \zeta]$ is equivalent to the existence of OWFs.*

So, if we could show that $\mathsf{McKTP}[t, \zeta]$ is NP-complete for some polynomials t, ζ, we would be done. Our main theorem does exactly this.

Theorem 1.2 (Main Theorem)**.** *For every polynomial $t(n) \geq n^2$, there exists some polynomial $\zeta(\cdot)$, such that $\mathsf{McKTP}[t, \zeta]$ is NP-complete (under randomized polynomial-time reductions).*

We believe this result is interesting in its own right and may provide a stepping stone towards the problem of proving that the (unconditional) time-bounded Kolmogorov complexity problem is NP-complete.

Let us emphasize that for the NP-completeness result to hold, it is imperative that our notion of conditional Kolmogorov complexity views programs as *RAM-machines* (as opposed to *Turing machines*). We leave it as an intriguing open problem to determine whether the "standard" conditional time-bounded Kolmogorov complexity (where interpreting a program as a Turing machine) is also NP-complete.

We proceed to providing a proof overview of the main theorem.

Proof Overview We first note that it directly follows that for all polynomials t, ζ, $\mathsf{McKTP}[t, \zeta] \in \mathsf{NP}$—the witness for an instance (x, z, k) is simply a RAM program Π such that $|\Pi| \leq k$ and $\Pi(z)$ generates x within $t(|x|)$ steps. We turn to discussing how to prove that there exists polynomials t, ζ, such that $\mathsf{McKTP}[t, \zeta]$ is NP-hard. On a high-level, our approach will start off by using the recent breakthrough approach by Ilango [Ila19, Ila20] showing NP-hardness of an *oracle*-variant of the *circuit minimization problem (MCSP)* [KC00]—that is, the problem of, given a truth table of a boolean function, determining the size of the smallest circuit that computes the function—and next extend it to deal with the conditional Kolmogorov complexity problem by appropriately embedding the "oracles" used in the construction of [Ila19] in the auxiliary input.

In more detail, following [AHM+08, HOS18, Ila19, Ila20, ILO20], we will embed an (approximate) Bounded Set Cover instance into an $\mathsf{McKTP}[t, \zeta]$ instance; the approximate Bounded Set Cover problem is known to be NP-complete [Tre01]. Recall that in the Bounded Set Cover problem, we are given a collection of sets $S_1, S_2, \ldots S_r$, each of which is a *constant*-size subset of the universe $U = [n]$ and the goal is to find a minimal set of indexes, s, such that $\cup_{i \in s} S_i = [n]$ (i.e., finding the minimal collection $\mathcal{S}$ of sets S_i that cover $[n]$). We start off by generalizing an idea from [Ila19, Ila20, ILO20] and replace the universe $U = [n]$ with n random strings $A_i \in \{0,1\}^m$, where $m(n)$ is some sufficiently large polynomial (in the formal proof $m(n) = n^3$). Roughly speaking, the rational for doing this is that a set cover, intuitively, should give a succinct (proportional to the size of the set cover) way to

[2] We remark, however, that as far as we know, we are the first to consider this problem w.r.t. RAM programs as opposed to Turing machines. In our view, this RAM version of the problem is as natural (if not more) than the "standard" TM version.

generate the random string $A = A_1||A_2||\dots||A_n$ if we have *oracle access* to the sets S_i—we simply need to specify the sets in the set cover and can then reconstruct the union of these sets. This construction was used in [Ila19] to prove NP-hardness of the oracle-version of the MCSP problem—the sets S_i were simply placed into the oracle. ([Ila20] provides a more elaborate construction that also shows NP-hardness of a conditional variant of the MCSP problem; we will, however, not rely on that extension.)

To convert the above set-cover instance into a conditional Kolmogorov complexity problem, our new idea will be to place the description of the sets S_i (each of which consists of some set of strings A_i) at *random locations* in the auxiliary string z and to make sure z is very long (yet still only of polynomial length), and consider the conditional Kolmogorov complexity problem of computing $K^t(A \mid z)$ where $A = A_1||A_2||\dots||A_n$. Conceptually, one can view this approach as a way to *obfuscate* the oracle used in [Ila19] and placing the obfuscation in z. Intuitively, since we are placing the descriptions of the sets S_i at random locations in z, a time-bounded algorithm can only access S_i if it "knows" the random location where it has been put, and thus, intuitively, we can view z as an information-theoretic obfuscation of gates that compute these descriptions.[3]

If there exists a set cover s of size ℓ, $K^t(A \mid z)$ should be no more than $\ell O(\log n) + O(1)$, by considering the program that simply hardcodes the location in z of the descriptions of the sets S_i for $i \in s$. The harder part is showing that if $K^t(A \mid z) \leq \ell O(\log n)$ then there exists a set-cover of size $O(\ell)$. Relying on the intuition that z acts as an obfuscation of the description of the sets $\{S_i\}$, the intuition for why this holds is that if z is sufficiently longer than $t(|x|)$, and the description of the sets are put into random positions of z, any program with running-time $t(|x|)$ that reconstructs the string $A = A_1||A_2||\dots||A_n$ (which with overwhelming probability has high Kolmogorov complexity) must "know" the location in the auxiliary string z of sets $\{S_i\}_{i\in s}$ in some set cover s, in the sense that by running this program, those positions can be "reconstructed". In a bit more detail, by running this program and looking at the memory access queries made by the program into z, we must be hitting the locations where the sets have been put. But since these locations are random (by construction of z), the program needs to basically "hard-code" them, or else we would be able to compress the indexes of these locations, but these indexes have high Kolmogorov complexity as they were picked at random, which is a contradiction.

The reader may note that, perhaps curiously, we are using an argument based on Kolmogorov complexity to formalize the statement that $K^t(A \mid z) \leq \ell O(\log n)$. In more detail, we are relying on a Kolmogorov-complexity style compression argument to formalize that z acts as a good "obfuscation" of the description of the sets $\{S_i\}$. This proof technique bears similarities to the proof technique pioneered by Gennero and Trevisan [GT00] in the context of proving that a random permutation is one-way w.r.t. polynomial-size circuits.

Let us end this section by noting that the above proof outline oversimplifies and misses several crucial details that make the actual proof quite a bit more complicated.

1.2 Related Works

As mentionned above, there has been a recent sequence of surprising works proving NP-hardness results for variants of the MCSP problem [AHM+08, HOS18, Ila19, Ila20, ILO20]; in particular, as mentionned, Ilango [Ila20] proves that a conditional version of the MCSP problem is NP-hard. As observed already in [Tra84], and further explored in [ABK+06], the MCSP problem is closely related to the time-bounded Kolmogorov complexity problem—intuitively, the two problems capture the

[3] This intuition is somewhat misleading: since z is only of polynomial length, the "obfuscation" only works with respect to a-priori time-bounded attackers (that can only explore a small fraction of z) and can only have inverse polynomial security. But in our context, such a relaxed notion of security suffices.

same concept, but using a different model of computation—but a formal reduction between these problems are not known so these results do not directly extend to the setting we consider. (However, as mentionned above, the starting point for our approach is the result of [Ila19] showing NP-hardness for an oracle version of the MCSP problem.)

A recent result by Hirahara [Hir20] directly addresses conditional time-bounded Kolmogorov complexity and shows NP-hardness for a variant of this problem, $\mathsf{McKTP}^{\mathsf{SAT}}$, where the program has access to a SAT-oracle. (The $\mathsf{McKTP}^{\mathsf{SAT}}$ problem, however, is not known to be in NP, but is in $\mathsf{NP}^{\mathsf{NP}}$, so NP-completeness is now shown).

An intriguing recent paper by Allender et al [ACM+21b] presented a natural NP-complete problem L—a sparse variant of the MCSP problem—such that average-case hardness of L was claimed to imply the existence of OWFs; the authors also claimed a "weak" converse of this implication—that the existence of OWFs implies a very weak, so-called "non-trivial", notion of average-case hardness of the language[4]; unfortunately, an error was found in the paper. Concurrently and independently from the current work, the authors of [ACM+21b] show how to repair the issues in their proof and present a different NP-complete language whose average-case hardness implies the existence of OWFs, and for which the same weak converse holds. While their original posting—which inspired the current work—attempted to base OWFs on the average-case hardness of a sparse version of the MCSP problem, their new paper [ACM+21a] instead bases OWFs on average-case hardness of a conditional Kolmogorov complexity style problem, just as in the current work. Their conditional Kolmogorov complexity problem differs from ours in several aspects: (1) whereas we consider conditional Kolmogorov complexity w.r.t. RAM programs, [ACM+21a] considers it w.r.t. Turing machines with "oracle-access" to the auxiliary input z; and (2) instead of considering a time-bounded version of conditional Kolmogorov complexity (as we do), [ACM+21a] instead *charge* for running-time in their notion of Kolmogorov complexity, following the KT notion of [ABK+06]. This second difference is appealing as it makes the notion of Kolmogorov complexity less parametrized, whereas we need to consider some polynomial time bound $t(\cdot)$. Due to these differences, NP-completeness of their problem follows essentially directly from the NP-completeness results of [Ila19] (whereas we have to work a lot harder, as explained above). However, due to these differences, they only manage to show a one-directional implication between average-case hardness of their problem and OWFs (and only a weak converse in the other direction), whereas we establish an *equivalence* between average-case hardness of $\mathsf{McKTP}[t, \zeta]$ (for any polynomials $t(n) > 1.1n, \zeta(\cdot)$) and OWFs.

After the initial posting of this paper, we were informed that Rahul Ilango [Ila21] had independently also shown NP-completeness of a conditional time-bounded Kolmogorov complexity problem (w.r.t. a RAM model of computation) but these results were not written down.

Resource bounded notions of conditional Kolmogorov complexity are useful also in other (related) contexts. In a companion paper to the current work [LP21], we rely on a notion of *space-bounded* conditional Kolmogorov complexity (defined similarly to the time-bounded notion of conditional Kolmogorov complexity used in the current paper) to characterize OWFs in NC_0.

In [LP21], we also identify a problem whose (infinitely-often) average-case hardness w.r.t. errorless heuristics is equivalent to $\mathsf{EXP} \neq \mathsf{BPP}$ (i.e., the problem is EXP-average-case complete w.r.t. errorless heuristics), yet (two-sided error) average-case hardness of this problem is equivalent to the existence of OWFs. Taken together, the current work and [LP21], demonstrate that the existence of OWFs can be characterized through the average-case hardness of both NP-complete (this work) and EXP-complete ([LP21]) languages.

[4]Roughly speaking, that average-case hardness holds for an inverse *exponential*, as opposed to inverse polynomial, fraction of inputs.

2 Preliminaries

We let $[n]$ denote the set $\{1, 2, \ldots, n\}$ for any integer $n \in \mathbb{N}$. For any two strings x, y, let $x||y$ denote the concatenation of x and y. In this work, we sometimes consider strings that contain a special symbol $\perp$ (besides 0 and 1). We will use the following standard encoding scheme—which we refer to as simple the *standard encoding scheme* $\mathsf{enc}_\perp$) to transform a string that may contain $\perp$ into a binary string: $\mathsf{enc}_\perp(x)$, of a string $x \in \{0, 1, \perp\}^*$ is a $2|x|$-bit binary string where we replace each bit in x by 00 for 0, 01 for 1, and 11 for $\perp$.

2.1 Set Cover

Let n be an integer and $S_1, S_2, \ldots, S_\ell, T$ be sets $\subseteq [n]$. We say that the sets $S_1, S_2, \ldots, S_\ell$ cover T if $T \subseteq S_1 \cup S_2 \cup \ldots \cup S_\ell$. Let $\mathcal{S}$ be a collection of sets. We define $\mathsf{cover}(T, \mathcal{S})$ to be the minimum number of sets in $\mathcal{S}$ necessary to cover T.

We recall the γ-Bounded Set Cover Problem:

- Input: $(1^n, 1^\ell, \mathcal{S})$ where n, ℓ are integers $\in \mathbb{N}$ and $\mathcal{S} = \{S_1, S_2, \ldots, S_r\}$ is a collection of subsets $\subseteq [n]$. It is guaranteed that all the sets in $\mathcal{S}$ covers $[n]$ together and for all $i \in [r]$, $|S_i| \leq \gamma$.

- Decide: Is $\mathsf{cover}([n], \mathcal{S}) \leq \ell$.

We also consider the approximate version of the γ-Bounded Set Cover problem. The α-approximate γ-Bounded Set Cover Problem is a promise problem $(\Pi_{\mathsf{yes}}, \Pi_{\mathsf{no}})$ where Π_{yes} contains $(1^n, 1^\ell, \mathcal{S})$ such that $\mathsf{cover}([n], \mathcal{S}) \leq \ell$ and Π_{no} consists of $(1^n, 1^\ell, \mathcal{S})$ such that $\mathsf{cover}([n], \mathcal{S}) > \alpha \cdot \ell$.

Trevisan [Tre01] showed that approximating the γ-Bounded Set Cover Problem within a constant factor is NP-hard:

Theorem 2.1 ([Tre01]). *For every constant $\alpha \geq 1$, there exists a constant $\gamma \in \mathbb{N}$ such that the α-approximate γ-Bounded Set Cover Problem is NP-hard.*

2.2 The RAM Model

A RAM program $\Pi = (M, y)$ consists of a CPU "next-step" Turing machine M, and some initial input $y \in \{0, 1\}^*$. Let $\mathsf{state} = 0$ be an initial state. The execution of this RAM program Π on input $z \in \{0, 1\}^*$ (which may be empty) proceeds as follows.

- At initialization, the memory is set to $y||\perp||z$, and the "read bit" b^{read} is set to $\perp$. (For simplicity, we assume that each memory position contains a symbol $\in \{0, 1, \perp\}$.[5] We assume that the memory is of infinite length and the rest of the positions in the memory are filled with $\perp$.)

- At each CPU step, M receives as input $\mathsf{state} \in \{0, 1\}^*$, the most recently read bit b^{read}, and outputs a new state $\mathsf{state}' \in \{0, 1\}^*$, a read position i^{read}, a write position i^{write} and some bit b^{write} (to be written to position i^{write}).[6]

- The execution of this step replaces state with state', sets b^{read} to the content of memory position i^{read}, and replaces the content of memory position i^{write} by b^{write}.

[5] When we implement this, we always use the standard encoding scheme, $\mathsf{enc}_\perp$. We also note that the string y and z can never contain the symbol $\perp$ (since they exclusively consist of 0s and 1s). When we load y and z into the memory, instead of storing y and z directly, we store the standard encoding of y and z (where 0 becomes 00 and 1 becomes 01).

[6] Formally, the inputs and outputs of M are separated by the $\perp$ symbol so that state can be of variable length.

- When $\mathsf{state} = \varepsilon$ (i.e., the empty string), the computation ends and the output of the of the computation is defined as the content of the memory tape up to the symbol $\perp$.[7]
- The running time of Π is defined to be the sum of the running time of M in all CPU steps.

Note that any polynomial-time Turing machine can be simulated by a polynomial-time RAM program by simply copying the content of the memory into state, next letting M run the original Turing machine using state as its tape, and finally copying the content of state back into the memory.

2.3 Time-bounded Conditional Kolmogorov Complexity

We introduce the notion of time-bounded conditional Kolmogorov complexity with respect to RAM programs. Roughly speaking, the *t-time-bounded Kolmogorov complexity*, $K^t(x \mid z)$, of a string $x \in \{0,1\}^*$ conditioned on a string $z \in \{0,1\}^*$ is the length of the shortest RAM program $\Pi = (M, y)$ such that $\Pi(z)$ outputs x in $t(|x|)$ steps.

Let U be some fixed Universal Turing machine that can emulate any RAM program Π with polynomial overhead. Let $U(\Pi(z), 1^t)$ denote the output of $\Pi(z)$ when emulated on U for t steps. We now define the notion of t-time-bounded conditional Kolmogorov complexity.

Definition 2.2. *Let t be a polynomial. For all $x \in \{0,1\}^*$ and $z \in \{0,1\}^*$, define*

$$K^t(x \mid z) = \min_{\Pi \in \{0,1\}^*} \{|\Pi| : U(\Pi(z), 1^{t(|x|)}) = x\}$$

where $|\Pi|$ is referred to as the description length *of Π. When there is no time bound, we define*

$$K(x \mid z) = \min_{\Pi \in \{0,1\}^*} \{|\Pi| : U(\Pi(z), 1^{t'} = x) \text{ for some finite } t'\}$$

We also consider the decisional variant of the minimum t-time-bounded conditional Kolmogorov complexity problem. Let t, ζ be two polynomials, and let $\mathsf{McKTP}[t, \zeta]$ denote the language of triples (x, z, k), having the property that $K^t(x \mid z) \leq k$, where $z \in \{0,1\}^{\zeta(|x|)}$ and $k \in \{0,1\}^{\lceil \log n \rceil}$.

We note that for any string $z \in \{0,1\}^*, x \in \{0,1\}^*$, for any polynomial $t(\cdot)$, $K^t(x \mid z)$, is always upper bounded by $|x| + O(1)$.

Fact 2.1. *There exists a constant $c \in \mathbb{N}$ such that for all polynomial $t(\cdot)$, for all string $z \in \{0,1\}^*, x \in \{0,1\}^*$, $K^t(x \mid z) \leq |x| + c$.*

Proof: Consider the RAM program $\Pi = (M, x)$ where M is a Turing machine that directly sets $\mathsf{state} = \varepsilon$. Note that in the execution of Π, x will be put into the memory and Π will halt immediately. Thus Π will output the string x. Note that M is a constant-size machine, so the description length of Π is at most $|x| + c$ for some constant c. ■

We finally remark that for any polynomials $t(\cdot), \zeta(\cdot)$, $\mathsf{McKTP}[t, \zeta] \in \mathsf{NP}$.

Claim 1. *For all polynomials $t(\cdot), \zeta(\cdot)$, $\mathsf{McKTP}[t, \zeta] \in \mathsf{NP}$.*

Proof: On input an instance $(x, z, k) \in \mathsf{McKTP}[t, \zeta]$, and a witness Π, checking if $|\Pi| \leq k$, $|z| = \zeta(|x|)$ and $U(\Pi(z), 1^{t(|x|)}) = x$ can be done in polynomial time. ■

[7] In a real execution, the content of the memory is encoded by the standard encoding scheme. The output of the computation is then defined by the *decoded* content of the memory.

2.4 One-way Functions

We recall the definition of one-way functions [DH76]. Roughly speaking, a function f is one-way if it is polynomial-time computable, but hard to invert for PPT attackers.

Definition 2.3. *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a polynomial-time computable function. f is said to be a* one-way function (OWF) *if for every* PPT *algorithm $\mathcal{A}$, there exists a negligible function μ such that for all $n \in \mathbb{N}$,*

$$\Pr[x \leftarrow \{0,1\}^n; y = f(x) : A(1^n, y) \in f^{-1}(f(x))] \leq \mu(n)$$

We may also consider a weaker notion of a *weak one-way function* [Yao82], where we only require all PPT attackers to fail with probability noticeably bounded away from 1:

Definition 2.4. *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a polynomial-time computable function. f is said to be a* α-weak one-way function (α-weak OWF) *if for every* PPT *algorithm $\mathcal{A}$, for all sufficiently large $n \in N$,*

$$\Pr[x \leftarrow \{0,1\}^n; y = f(x) : A(1^n, y) \in f^{-1}(f(x))] < 1 - \alpha(n)$$

We say that f is simply a weak one-way function (weak OWF) *if there exists some polynomial $q > 0$ such that f is a $\frac{1}{q(\cdot)}$-weak OWF.*

Yao's hardness amplification theorem [Yao82] shows that any weak OWF can be turned into a (strong) OWF.

Theorem 2.5 ([Yao82])**.** *Assume there exists a weak one-way function. Then there exists a one-way function.*

2.5 Average-case Hard Languages

We turn to defining what it means for a language to be average-case hard (for PPT algorithms). We will be considering languages that are only defined on some input lengths (such as $\mathsf{McKTP}[t,\zeta]$). We say that a language L is *defined over inputs lengths* $s(\cdot)$ if $L \subseteq \cup_{n\in\mathbb{N}}\{0,1\}^{s(n)}$. For concreteness, note that $\mathsf{McKTP}[t,\zeta]$ is defined on input lengths $s(n) = n + \zeta(n) + \lceil \log n \rceil$.

We now turn to defining average-case hardness.

Definition 2.6. *We say that a language L defined over inputs lengths $s(\cdot)$ is $\alpha(\cdot)$* hard-on-average (α-HoA) *if for all* PPT *heuristic $\mathcal{H}$, for all sufficiently large $n \in N$,*

$$\Pr[x \leftarrow \{0,1\}^{s(n)} : \mathcal{H}(x) = L(x)] < 1 - \alpha(n)$$

In other words, there does not exist a PPT "heuristic" $\mathcal{H}$ that decides L with probability $1-\alpha(n)$ on infinitely many input lengths $n \in N$ over which L is defined.

We refer to a language L as being *mildly HoA* if there exists a polynomial $p(\cdot) > 0$ such that L is $\frac{1}{p(\cdot)}$-HoA.

3 NP-Hardness of $\mathsf{McKTP}[t,\zeta]$

In this section, we prove our main theorem: We show that there exists a reduction from the approximate γ-Bounded Set Cover Problem to $\mathsf{McKTP}[t,\zeta]$ when t,ζ are sufficiently large.

Theorem 3.1. *For all polynomial $t(n) \geq n^2$, there exists a polynomial $\zeta(n)$ such that* $\mathsf{McKTP}[t,\zeta]$ *is* NP*-hard under many-one randomized polynomial-time reductions.*

Proof: The theorem follows from Proposition 3.1 and Proposition 3.2 (stated and proved in Section 3.2), and Theorem 2.1. ■

3.1 A Reduction from the γ-Bounded Set Cover Problem to McKTP

Let γ be a constant, let $t(n) \geq n^2$ be a polynomial, and consider $\zeta(n) = (t(n))^4 n^{2\gamma}$. We will show that there exists a randomized reduction from the γ-Bounded Set Cover Problem to $\mathsf{McKTP}[t, \zeta]$.

Given an instance $(1^n, 1^\ell, \mathcal{S})$ where $\mathcal{S} = \{S_1, S_2, \ldots, S_r\}$ of the γ-Bounded Set Cover Problem, we proceed as follows:

- Let $m = n^3$; for each $i \in [n]$, sample a random string $A_i \in \{0,1\}^m$, and consider the length-$(n \times m)$ concatenation $A = A_1 || A_2 || \ldots || A_n$ of the sampled strings. Think of A_i as a randomized encoding of the element i in the Set Cover problem. See Figure 1 for an illustration of these strings.

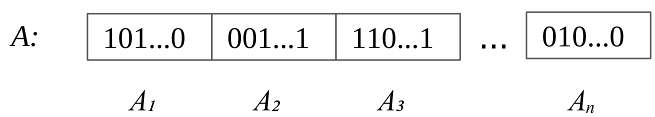

Figure 1: An illustrative example for the string A

- For each $i \in [r]$, we construct a "gadget" string $W_i \in (\{0,1\}^m)^n$ (for set S_i). We partition W_i into n blocks $W_{i,1}, W_{i,2}, \ldots, W_{i,n}$ where each block is of size m. We let $W_{i,j} = A_j$ if $j \in S_i$, and otherwise $W_{i,j} = 0^m$. In other words, W_i reveals the strings A_j for all $j \in S_i$; think of W_i as a randomized encoding of the set S_i. See Figure 2 for an illustration of there strings.

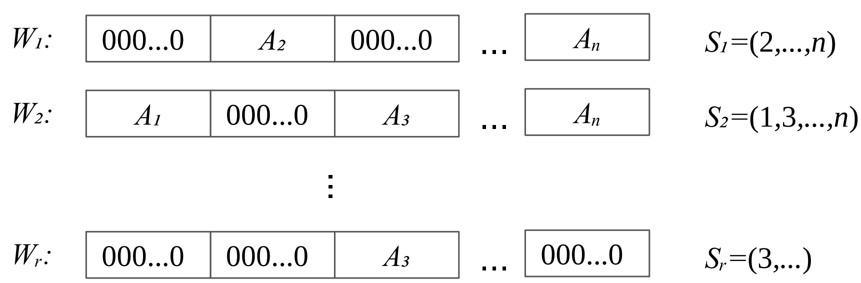

Figure 2: An illustrative example for the gadget strings W_i. Note that if we have a Set Cover $(i_1, i_2, \ldots, i_\ell)$, then the bitwise OR of the strings W_{i_1}, W_{i_1}, $\ldots W_{i_\ell}$ equals A.

- Let $\lambda = 4 \log r + 4 \log t(nm)$. For each $i \in [r]$, we sample a "key" $k_i \in \{0,1\}^\lambda$ for W_i. For simplicity, we assume that the sampled keys are distinct with each other. (If this is not the case, the reduction just aborts; since this happens only with negligible probability we may ignore this event in the analysis.)

- We are finally ready to describe the "auxiliary input" z. The idea is to hide the gadgets $\{W_i\}$ in z at random locations specified by the keys so as to ensure that the only way for a t-time bounded program to recover W_i is to essentially hard-code the key k_i as part of its description. In more detail, we consider a string z of length $2^\lambda \times n \times m$; partition z into 2^λ blocks $z_{0^\lambda}, z_{0^{\lambda-1}1}, \ldots, z_{1^{\lambda-1}0}, z_{1^\lambda}$ where for all $p \in \{0,1\}^\lambda$, $|z_p| = n \times m$. For all $p \in \{0,1\}^{2^\lambda}$, let $z_p = W_i$ if $p = k_i$ for some $i \in [r]$, and otherwise, let $z_p = 0^{n \times m}$. See Figure 3 for an illustration of there strings.

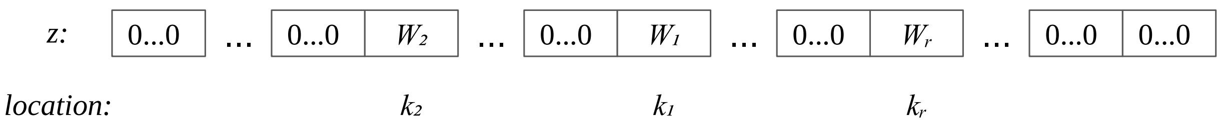

Figure 3: An illustrative example of the "auxiliary" input z

- Finally, the reduction will output YES if $K^t(A \mid z) \leq 2\lambda\ell$. Note that the length of z is upper bounded by $\zeta(|A|)$, and thus this is a syntactically valid reduction to an $\mathsf{McKTP}[t, \zeta]$ instance.

We turn to analyzing the success probability of the reduction.

3.2 Analyzing the Reduction

We will prove that the above reduction above gives us a 4-approximation of the γ-Bounded Set Cover Problem. We first show that if $[n]$ can be covered by a small number ($\leq \ell$) of sets, the time-bounded Kolmogorov complexity of A conditioned on the string z will be small ($\leq 2\lambda\ell$): the program computing A simply needs to hard-code the keys k_i corresponding to the ℓ sets in the set cover; it can look into z at the positions specified by the keys and output the bitwise OR of the content of those positions.

Proposition 3.1. *If* $\mathsf{cover}([n], \mathcal{S}) \leq \ell$ *then* $K^t(A \mid z) \leq 2\lambda\ell$.

Proof: Let $S_{i_1}, S_{i_2}, \ldots, S_{i_\ell}$ be the ℓ sets in $\mathcal{S}$ that cover $[n]$. (Since the sets are γ-Bounded, it follows that $\ell \geq n/\gamma$.) Let Π be a RAM program with n, m, λ and the keys $k_{i_1}, \ldots, k_{i_\ell}$ hardwired in it. For each $j \in [\ell]$, Π first reads $W'_{i_j} = z_{k_{i_j}}$ from the k_{i_j}-th block of the string z (where $|z_{k_{i_j}}| = n \times m$). (Recall that z is partitioned into 2^λ blocks and each block is of size $n \times m$.) Π then obtains $W'_{i_1}, \ldots, W'_{i_\ell}$ and Π simply outputs

$$W'_{i_1} \vee W'_{i_2} \vee \ldots \vee W'_{i_\ell}$$

where $\vee$ denotes the bitwise OR for binary strings.

We first show that Π indeed outputs the string A. Note that by the construction of string z, it holds that

$$(W'_{i_1}, \ldots, W'_{i_\ell}) = (z_{k_{i_1}}, \ldots, z_{k_{i_\ell}}) = (W_{i_1}, \ldots, W_{i_\ell}).$$

Recall that in the construction of the gadget string W_{i_j} (for each $j \in [\ell]$), W_{i_j} is partitioned into n blocks $W_{i_j,1}, \ldots, W_{i_j,n}$. And for each block $b \in [n]$, $W_{i_j,b} = A_b$ if $b \in S_{i_j}$, and otherwise $W_{i_j,b} = 0^m$. Since the sets $S_{i_1}, \ldots, S_{i_\ell}$ cover $[n]$, for all $b \in [n]$, there exists an index j such that the b-th block of the gadget string W_{i_j} matches A_b. Thus, $W_{i_1} \vee W_{i_2} \vee \ldots \vee W_{i_\ell} = A$.

We then show that Π can be described within $2\lambda\ell$ bits. Recall that Π contains the values n, m, λ (which takes $O(\log n)$ bits to describe), the keys $k_{i_1}, \ldots, k_{i_\ell}$ (which takes $\lambda\ell$ bits), and the code of Π (which takes $O(1)$ bits). We will provide a more fine-grained analysis in the Appendix A to show that the code of Π is of constant-bit length in the RAM model. Thus, Π can be represented using $\lambda\ell + O(\log n) \leq 2\lambda\ell$ bits.

Finally, note that Π runs in time $O(\ell n m \,\mathsf{poly} \log n) \leq (nm)^2 \leq t(|A|)$ (since in each CPU step, the CPU next-step machine takes $O(\mathsf{poly} \log n)$ time). (We refer the reader to the Appendix A for a more detailed running time analysis.) Thus, we conclude that $K^t(A \mid z) \leq 2\lambda\ell$. ■

The key part of the analysis is showing that if $K^t(A \mid z) \leq 2\lambda\ell$ then $\mathsf{cover}([n], \mathcal{S}) \leq 4\ell$:

Proposition 3.2. *With probability at least $1 - 2/n$ over the random choice of $k_1, k_2, \ldots, k_r$ (which determines z) and A, it hold that if $K^t(A \mid z) \leq 2\lambda\ell$ then $\mathsf{cover}([n], \mathcal{S}) \leq 4\ell$.*

The proof of Proposition 3.2 is provided in Section 3.3. Proposition 3.1 together with Proposition 3.2 concludes that our reduction achieves a 4-approximation.

3.3 Proof of Proposition 3.2

Let Π be a RAM program such that $|\Pi| \leq 2\lambda\ell$ and $\Pi(z)$ prints A in $\leq t = t(|A|)$ CPU steps (where t is the running time bound associated with the problem $\mathsf{McKTP}[t, \zeta]$). The existence of such Π is implied by the assumption that $K^t(A \mid z) \leq 2\lambda\ell$. We will now show how to use Π, z to extract out a Set Cover of size 4ℓ. Towards this, recall that when executing $\Pi(z)$, in each CPU step, $\Pi(z)$ will read one bit from the memory. Let

$$q_1, q_2, \ldots, q_t$$

be the memory positions that $\Pi(z)$ reads in the execution of $\Pi(z)$ (such that in CPU step i, $\Pi(z)$ reads the content of memory position q_i). Note that the string z will be stored in the memory of $\Pi(z)$, and we are interested in the memory positions where the string z is stored. So, we let d be the memory position such that z is stored from position d to position $d+|z|-1$. In addition, most of bits in z are just zeros and $z_{k_1}, z_{k_2}, \ldots, z_{k_r}$ are the only informative blocks. (Recall that z is partitioned into 2^λ blocks of size $n \times m$.) Thus, let

$$p_i = \lfloor (q_i - d)/(n \times m) \rfloor$$

be the index of the block in z from which $\Pi(z)$ reads one bit in CPU step i. When p_i matches some key k_j, $z_{p_i} = z_{k_j} = W_j$. When p_i does not match any of the keys, $z_{p_i} = 0^{n \times m}$.[8]

We say that $\Pi(z)$ makes a $\mathsf{useful\ access}$ to the string z in CPU step i if there exists $j \in [r]$ such that $p_i = k_j$ and for all $i' < i$, $p_i \neq p_{i'}$. In other words, $\Pi(z)$ makes a $\mathsf{useful\ access}$ when it first reads some bit in the block z_{k_j} for some $j \in [r]$. We say that $\Pi(z)$ hits some block z_p if in some CPU step i, $\Pi(z)$ reads one bit from z_p.

Bounding the number of useful accesses We first present an upper-bound on the number of $\mathsf{useful\ accesses}$. The following central claim shows that if the number of $\mathsf{useful\ accesses}$ is large, then the Kolmogorov complexity of Keys must be small.

Claim 2. *Let $\mathsf{Keys} = k_1||k_2||\ldots||k_r$ be the concatenation of $k_1, k_2, \ldots, k_r$. If $\Pi(z)$ makes α (or more) $\mathsf{useful\ accesses}$ to the string z, then*

$$K(\mathsf{Keys} \mid A, \mathcal{S}) \leq |\Pi| + (r - \alpha)\lambda + \alpha(\log t + \log r) + O(\log n)$$

We defer the proof of Claim 2 to Section 3.4

We observe that since Keys are picked at random, their (conditional) Kolmogorov complexity is high.

Claim 3. *For all $A \in \{0,1\}^{n \times m}$, with probability $1 - 1/n$ (over the random choice of Keys), it holds that*

$$K(\mathsf{Keys} \mid A, \mathcal{S}) \geq |\mathsf{Keys}| - \log n \geq r\lambda - \log n.$$

[8] Here we discuss the string z constructed by the reduction, instead of the one stored in the memory. (So Π can not manipulate values in z.) Thus, when p_i is out of the range (e.g., $p_i < 0$), it still holds that $z_{p_i} = 0^{n \times m}$.

Proof: Note that the total number of RAM programs with description length $< r\lambda - \log n$ is at most $2^{r\lambda - \log n} \leq \frac{2^{r\lambda}}{n}$, while the total number of the choices of Keys is $2^{r\lambda}$); thus the claim follows. ■

By combining Claim 2 and Claim 3 we get the following bound on the number of useful accesses.

Corollary 3.2. *With probability* $1 - 1/n$ *over the random choice of* Keys, *if* $|\Pi| \leq 2\lambda\ell$, *it holds that* $\Pi(z)$ *makes at most* 4ℓ useful accesses

Proof: Assume not. Then by Claim 2,

$$\begin{aligned} K(\mathsf{Keys} \mid A, \mathcal{S}) &\leq |\Pi| + (r - 4\ell)\lambda + 4\ell(\log t + \log r) + O(\log n) \\ &\leq 2\lambda\ell + r\lambda - 4\lambda\ell + 4\ell(\log t + \log r) + O(\log n) \\ &\leq r\lambda - (2\lambda\ell - 4\ell(\log t + \log r) - O(\log n)) \\ &\leq r\lambda - (8\ell(\log t + \log r) - 4\ell(\log t + \log r) - O(\log n)) \\ &\leq r\lambda - (\frac{n}{\gamma} - O(\log n)) \\ &< r\lambda - \log n \end{aligned}$$

which contradicts Claim 3. ■

Extracting a small Set Cover We now turn to showing that we can extract a Set Cover from Π, z which is bounded in size by the number of useful accesses. We first show that if $\Pi(z)$ manages to output the string A, yet does not makes useful accesses such that the union of all the blocks that are hit by $\Pi(z)$ equal A, then the Kolmogorov complexity of A must be small.

Claim 4. *Assume that*

- $\Pi(z)$ *makes* α useful accesses*;*
- $\Pi(z)$ *outputs the string* A*.*
- $z_{p_1} \vee z_{p_2} \vee \ldots \vee z_{p_t} \neq A$*;* [9]

Then,

$$K(A \mid \mathcal{S}) \leq |\Pi| + (n-1)m + \alpha(\log t + \log r) + O(\log n)$$

We defer the proof of Claim 4 to Section 3.4.

We observe that since A is a random string and its (conditional) Kolmogorov complexity must be high.

Claim 5. *With probability* $1 - 1/n$ *(over the random choice of* A*), it holds that*

$$K(A \mid \mathcal{S}) \geq |A| - \log n \geq nm - \log n.$$

Proof: Note that the total number of RAM programs with description length $< nm - \log n$ is at most $2^{nm - \log n} \leq \frac{2^{nm}}{n}$ (while the total number of the choices of A is 2^{nm}); thus the claim follows. ■

Combining Claim 4 and Claim 5, we conclude that the union of all the blocks hit by $\Pi(z)$ must equal A (provided that $\Pi(z)$ prints the string A and makes at most 4ℓ useful accesses).

[9] When $p_i < 0$ or $p_i \geq 2^{\lambda}$, we assume that z_{p_i} is an all-zero string and $z_{p_i} = 0^{n \times m}$.

Corollary 3.3. *With probability* $1 - 1/n$ *over the random choice of* A*, if* $|\Pi| \leq 2\lambda\ell$*,* $\Pi(z) = A$*, and* $\Pi(z)$ *makes at most* 4ℓ useful accesses*, it holds that* $z_{p_1} \vee z_{p_2} \vee \ldots \vee z_{p_t} = A$*.*

Proof: Assume not. Then by Claim 4,

$$\begin{aligned} K(A \mid \mathcal{S}) &\leq |\Pi| + (n-1)m + 4\ell(\log t + \log r) + O(\log n) \\ &\leq 2\lambda\ell + (n-1)m + 4\ell(\log t + \log r) + O(\log n) \\ &= nm - (m - (2\lambda\ell + 4\ell(\log t + \log r) + O(\log n))) \\ &< nm - \log n \quad (\text{since } m = n^3, \lambda \leq n, \ell \leq n) \end{aligned}$$

which contradicts to Claim 5. ■

We finally show that if the union of all the blocks hit by $\Pi(z)$ matches A, then we can extract out a Set Cover whose size is bounded by the number of useful accesses $\Pi(z)$ made.

Claim 6. *If* $\Pi(z)$ *makes at most* 4ℓ useful accesses *and* $z_{p_1} \vee z_{p_2} \vee \ldots \vee z_{p_t} = A$*, then* $\mathsf{cover}([n], \mathcal{S}) \leq 4\ell$*.*

Proof: Let α be the number of useful accesses made by $\Pi(z)$. Let

$$i_1, i_2, \ldots, i_\alpha$$

be the CPU steps when $\Pi(z)$ makes a useful access; that is, $i_1, i_2, \ldots, i_\alpha$ is a sequence of CPU step indices such that for each $l \in [\alpha]$, $\Pi(z)$ will make a useful access in CPU step i_l. (Recall that except for $z_{k_1}, \ldots, z_{k_r}$, the blocks in the string z are all-zero strings.) Recall that $\Pi(z)$ makes a useful access when it first reads some bit in the block z_{k_j} for some $j \in [r]$.) Thus, by the definition of useful access, it follows that

$$z_{p_{i_1}} \vee z_{p_{i_2}} \vee \ldots \vee z_{p_{i_\alpha}} = z_{p_1} \vee z_{p_2} \vee \ldots \vee z_{p_t} = A$$

Since $\Pi(z)$ makes a useful access in CPU step i_l, p_{i_l} must equal some key. We let $j_1, j_2, \ldots, j_\alpha \in [r]$ be a sequence of indices of the keys such that

$$(p_{i_1}, p_{i_2}, \ldots, p_{i_\alpha}) = (k_{j_1}, k_{j_2}, \ldots, k_{j_\alpha})$$

Note that (by the construction of the string z)

$$(W_{j_1}, W_{j_2}, \ldots, W_{j_\alpha}) = (z_{p_{i_1}}, z_{p_{i_2}}, \ldots, z_{p_{i_\alpha}})$$

Thus, it follows that

$$W_{j_1} \vee W_{j_2} \vee \ldots \vee W_{j_\alpha} = z_{p_{i_1}} \vee z_{p_{i_2}} \vee \ldots \vee z_{p_{i_\alpha}} = A$$

Finally, we argue that $S_{j_1}, S_{j_2}, \ldots, S_{j_\alpha}$ cover $[n]$, which concludes the proof (since $\alpha \leq 4\ell$). We recall that for each $l \in [\alpha]$, W_{j_l} is the gadget string for S_{j_l}. Furthermore, W_{j_l} is partitioned into n blocks, $W_{j_l,1}, W_{j_l,2}, \ldots, W_{j_l,n}$. For each block $b \in [n]$, $W_{j_l,b} = A_b$ if $b \in S_{j_l}$, and otherwise $W_{j_l,b} = 0^m$. Since $W_{j_1} \vee W_{j_2} \vee \ldots \vee W_{j_\alpha} = A$, it follows that for all blocks $b \in [n]$,

$$W_{j_1,b} \vee W_{j_2,b} \vee \ldots \vee W_{j_\alpha,b} = A_b.$$

Thus, for all $b \in [n]$, there must exist $l \in [\alpha]$ such that $b \in S_{j_l}$. We conclude that the sets $S_{j_1}, S_{j_2}, \ldots, S_{j_\alpha}$ indeed cover $[n]$. ■

We can now conclude the proof of Proposition 3.2:

Proof: [of Proposition 3.2] By Corollary 3.2, with probability $1 - 1/n$, $\Pi(z)$ makes at most 4ℓ useful accesses. By Corollary 3.3, with probability $1 - 1/n$, it holds that $z_{p_1} \vee z_{p_2} \vee \ldots \vee z_{p_t} = A$. Finally by Claim 6, it holds that $\mathsf{cover}([n], \mathcal{S}) \leq 4\ell$, which happens with probability at least $1 - 2/n$ (by a union bound). ■

3.4 Proof of Claim 2 and Claim 4

In both Claim 2 and Claim 4, the goal is to compress some strings (either Keys or A) provided that $\Pi(z)$ prints A. Towards doing this, we need be able to find a short representation of the information needed to perform the execution of $\Pi(z)$. Towards this, it will be helpful to track when $\Pi(z)$ makes a useful access. Furthermore, note that every useful access corresponds to some key k_j such that z_{k_j} stores the gadgets W_j of the set S_j. For each such useful access, we will also track this "key index" j. As we shall see, given Π, A and S, as well as the sequence of CPU steps and key indexes (of useful accesses), the whole execution of $\Pi(z)$ can be emulated without having access to z. In fact, as we shall formalize now, we actually do not even need the full content of A and S, but rather just the gadgets W_j corresponding to the sets hit by the useful accesses.

To formalize this, let $t = t(|A|)$ be the maximum number of CPU steps that $\Pi(z)$ can run, and let α be some integer bounded by the number of useful accesses made by $\Pi(z)$. We refer a pair of sequences of CPU steps and key indexes $\omega = ((i_1, i_2, \ldots, i_\alpha), (j_1, j_2, \ldots, j_\alpha)) \in [t]^\alpha \times [r]^\alpha$ as a configuration. We say that $\Pi(z)$ *matches* ω if the first time $\Pi(z)$ makes a useful access is in CPU step i_1 and $\Pi(z)$ reads one bit from the block $z_{k_{j_1}}$ (and recall that $z_{k_{j_1}} = W_{j_1}$), and the second time $\Pi(z)$ makes a useful access is in CPU step i_2 and $\Pi(z)$ reads one bit from the block $z_{k_{j_2}}$, and so on.

Lemma 3.3. *Let $\alpha \in \mathbb{N}$, and $\omega = ((i_1, \ldots, i_\alpha), (j_1, \ldots, j_\alpha))$ be a* configuration *in $[t]^\alpha \times [r]^\alpha$. If $\Pi(z)$ matches ω then one can emulate $\Pi(z)$ for i_α CPU steps using the code of Π, the configuration ω, and $W_{j_1}, W_{j_2}, \ldots, W_{j_\alpha}$ (without having access to z).*

Proof: We now describe how to emulate the execution of $\Pi(z)$ for i_α steps using the code of Π, the configuration ω, and $W_{j_1}, W_{j_2}, \ldots, W_{j_\alpha}$. Recall that d is the memory position where z starts at; that is, z is stored in memory positions d to $d + |z| - 1$.

Given the code of Π, we start to emulate $\Pi(z)$ with the content of memory positions $d, d + 1, \ldots, d + |z| - 1$ (which are supposed to store z) set to 0. In the simulation, we keep track of all memory positions that $\Pi(z)$ has written to. In each CPU step i, if i matches some value in $\{i_1, i_2, \ldots, i_\alpha\}$ (and suppose $i = i_l$), we proceed as follows:

- Let q_i be the memory position which Π will read from in CPU step i and proceed as follows.
- Let $p_i = \lfloor (q_i - d)/(n \times m) \rfloor$. Put the string $W_{j_l} \in \{0,1\}^{n \times m}$ into the memory from position $d + p_i \times nm$ to position $d + p_i \times nm + nm - 1$, with the following exception: If Π has ever previously written into a memory between position $d + p_i \times nm$ and $d + p_i \times nm + nm - 1$, we keep those bits unchanged.
- Finally, let Π will read the bit from the memory (just as if the string z had been there), and we continue to emulate the execution of $\Pi(z)$ in the rest of CPU step i.

If i does not appear in $\{i_1, i_2, \ldots, i_\alpha\}$, we simply emulate the execution honestly. When $i = i_\alpha$, we stop to emulate $\Pi(z)$.

We argue, by induction, that the above procedure perfectly emulates the execution of $\Pi(z)$ in the first i_α CPU steps. For the base case, we consider CPU step $i = 0$, in which $\Pi(z)$ has not started yet, so the statement is trivially true. For any $i \leq i_\alpha$, we now assume that in all the steps $\leq i - 1$, our simulation perfectly emulates $\Pi(z)$, and we will prove that also in CPU step i, the simulation does so as well. First note that if, in CPU step i, Π attempts to read from a memory position q_i that has (1) previously been written or read from, (2) the memory position is not within the range $[d, d + |z| - 1]$, or (3) the memory access to q_i is not a useful access, then the induction step directly follows from the induction hypothesis and the fact that the step is performed in exactly the same way in the simulation as in the real execution. We thus only need to consider the case when the memory access to q_i is a $\mathsf{useful\ access}$. But whenever this happens, by the induction hypothesis, the simulation will produce exactly the same content in the block of z where q_i is contained, as in the real execution of $\Pi(z)$. It thus follows that also this step is perfectly emulated.

Thus, we conclude that $\Pi(z)$ can be emulated for i_α steps using the code of Π, the $\mathsf{configuration}$ ω, and $W_{j_1}, W_{j_2}, \ldots, W_{j_\alpha}$. ■

We are now ready to prove Claim 2, which we restate for the convenience of the reader.

Claim 7 (Claim 2, restated)**.** *Let* $\mathsf{Keys} = k_1||k_2||\ldots||k_r$ *be the concatenation of* $k_1, k_2, \ldots, k_r$*. If* $\Pi(z)$ *makes* α *(or more)* $\mathsf{useful\ accesses}$ *to the string* z*, then*

$$K(\mathsf{Keys} \mid A, \mathcal{S}) \leq |\Pi| + (r-\alpha)\lambda + \alpha(\log t + \log r) + O(\log n)$$

Proof: If $\Pi(z)$ makes at least α $\mathsf{useful\ accesses}$, $\Pi(z)$ must match some $\mathsf{configuration}$

$$\omega = ((i_1, i_2, \ldots, i_\alpha), (j_1, j_2, \ldots, j_\alpha))$$

where $\omega \in [t]^\alpha \times [r]^\alpha$. We let $\{j'_1, j'_2, \ldots, j'_{r-\alpha}\} = [r] - \{j_1, j_2, \ldots, j_\alpha\}$ be the set of key indices that do *not* appear in ω.

We consider the following program Π' that prints the string $\mathsf{Keys} = k_1||k_2||\ldots||k_r$ with the string A and the collection of sets $\mathcal{S}$ as auxiliary information. Π' has the values n, m, λ, α, t, r hardwired in it, and the code of Π' also includes the $\mathsf{configuration}$ ω, the code of Π, and the $r-\alpha$ keys $k_{j'_1}, k_{j'_2}, \ldots, k_{j'_{r-\alpha}}$. Π' first computes $W_{j_1}, W_{j_2}, \ldots, W_{j_\alpha}$ from A and $\mathcal{S}$. Π' then emulates the execution of $\Pi(z)$ (using the code of Π, the $\mathsf{configuration}$ ω, and $W_{j_1}, W_{j_2}, \ldots, W_{j_\alpha}$, using the method described in Lemma 3.3) for i_α CPU steps (recall that i_α is the CPU step when $\Pi(z)$ makes its α'th $\mathsf{useful\ access}$). Let d be the index such that z is initially stored in the memory from position d to the position $d + |z| - 1$. Let

$$q_1, q_2, \ldots, q_{i_\alpha}$$

be the memory positions that $\Pi(z)$ reads (such that in CPU step i, $\Pi(z)$ reads one bit from memory position q_i) in the first i_α CPU steps. We will decode $k_{j_1}, k_{j_2}, \ldots, k_{j_\alpha}$ from $q_1, q_2, \ldots, q_{i_\alpha}$ as follows: For each $i \leq i_\alpha$, let

$$p_i = \lfloor (q_i - d)/(n \times m) \rfloor$$

Since $\Pi(z)$ matches ω, it follows that

$$p_{i_1} = k_{j_1}, p_{i_2} = k_{j_2}, \ldots, p_{i_\alpha} = k_{j_\alpha}$$

Thus, Π' has access to $k_{j'_1}, k_{j'_2}, \ldots, k_{j'_{r-\alpha}}$ (hardwired) and can compute $k_{j_1}, k_{j_2}, \ldots, k_{j_\alpha}$ as specified above. Thus, Π' can recover and output the string $\mathsf{Key} = k_1||k_2||\ldots||k_r$.

Finally, we show that the description length of Π' is at most $|\Pi| + (r-\alpha)\lambda + \alpha(\log t + \log r) + O(\log n)$. To describe Π', we require:

- $|\Pi|$ bits to store the code of Π;
- $(r-\alpha)\lambda$ bits to store the $r-\alpha$ keys $k_{j'_1}, k_{j'_2}, \ldots, k_{j'_{r-\alpha}}$;
- $\alpha(\log t + \log r)$ bits to store the configuration ω.
- $O(\log n)$ bits to store the values n, m, λ, α, t, r.
- $O(1)$ bits to describe the CPU next-step machine.

Thus, the description length of Π' is at most $|\Pi| + (r-\alpha)\lambda + \alpha(\log t + \log r) + O(\log n)$, and from this we conclude that

$$K(\mathsf{Keys} \mid A, \mathcal{S}) \leq |\Pi| + (r-\alpha)\lambda + \alpha(\log t + \log r) + O(\log n)$$

which completes the proof. ■

We next proceed to prove Claim 4, which we first restate:

Claim 8 (Claim 4, restated)**.** *Assume that*

- $\Pi(z)$ *makes* α useful accesses*;*
- $\Pi(z)$ *outputs the string* A*.*
- $z_{p_1} \vee z_{p_2} \vee \ldots \vee z_{p_t} \neq A$*;* [10]

Then,

$$K(A \mid \mathcal{S}) \leq |\Pi| + (n-1)m + \alpha(\log t + \log r) + O(\log n)$$

Proof: Consider some Π, z satisfying the pre-conditions of the claim. Since $\Pi(z)$ has the property that

$$z_{p_1} \vee z_{p_2} \vee \ldots \vee z_{p_t} \neq A,$$

and recalling that each z_{p_i} is divided n m-size blocks, $z_{p_i,1}, \ldots, z_{p_i,n}$, it follows that there exists a block index $b \in [n]$ such that for each block $z_{p_i} \in \{0,1\}^{n\times m}$ that $\Pi(z)$ reads, $z_{p_i,b} = 0^m$. In addition, note that $\Pi(z)$ makes α useful accesses, so $\Pi(z)$ must match some configuration

$$\omega = ((i_1, i_2, \ldots, i_\alpha), (j_1, j_2, \ldots, j_\alpha))$$

where $\omega \in [t]^\alpha \times [r]^\alpha$. Since $\Pi(z)$ matches ω, we know that

$$(W_{j_1}, W_{j_2}, \ldots, W_{j_\alpha}) = (z_{p_{i_1}}, z_{p_{i_2}}, \ldots, z_{p_{i_\alpha}})$$

Thus,

$$W_{j_1} \vee W_{j_2} \vee \ldots \vee W_{j_\alpha} \neq A$$

It follows that for all $l \in [\alpha]$, $W_{j_l,b} = 0^m$. From this, we can conclude that the gadget strings $W_{j_1}, W_{j_2}, \ldots, W_{j_\alpha}$ can be constructed from $\mathcal{S}$ and all randomized encodings $A_1, \ldots, A_{b-1}, A_{b+1}, \ldots, A_n$ *excluding* A_b.

Based on this observation, let us show how to construct a program Π' that outputs the string A given $\mathcal{S}$ as auxiliary information. The program Π' embeds the values n, m, λ, α, r, t, the value of b, the code of Π, the configuration ω, and strings $A_1, \ldots, A_{b-1}, A_{b+1}, \ldots, A_n$ into its code. Π' first computes $W_{j_1}, W_{j_2}, \ldots, W_{j_\alpha}$ from $A_1, \ldots, A_{b-1}, A_{b+1}, \ldots, A_n$ and $\mathcal{S}$. Π' then simulates the execution of $\Pi(z)$ using the code of Π, the configuration ω, and the gadget strings $W_{j_1}, W_{j_2}, \ldots, W_{j_\alpha}$ (making use of Lemma 3.3), and finally outputs whatever $\Pi(z)$ outputs. Note that since $\Pi(z)$ makes exactly α useful accesses, Π' can emulate $\Pi(z)$ all the way until it terminates. Furthemore, recall that by assumption $\Pi(z)$ outputs A, so Π' will do so as well.

We finally show that the description length of Π' is at most $|\Pi| + (n-1)m + \alpha(\log t + \log r) + O(\log n)$. To see this, note that to specify Π', we require:

- $|\Pi|$ bits to include the code of Π;

[10] When $p_i < 0$ or $p_i \geq 2^\lambda$, we assume that z_{p_i} is an all-zero string and $z_{p_i} = 0^{n\times m}$.

- $(n-1)m$ bits to store strings $A_1, \ldots, A_{b-1}, A_{b+1}, \ldots, A_n$;
- $\alpha(\log t + \log r)$ bits to save the configuration ω.
- $O(\log n)$ bits to strore the values n, m, λ, α, r, t, b
- $O(1)$ bits to implement the CPU next-step machine:

Thus, we have that the description length of Π' is at most $|\Pi| + (n-1)m + \alpha(\log t + \log r) + O(\log n)$. From this we conclude that

$$K(A \mid \mathcal{S}) \leq |\Pi| + (n-1)m + \alpha(\log t + \log r) + O(\log n).$$

which proves the claim. ■

4 OWFs from Mild Avg-case Hardness of $\mathsf{McKTP}[t, \zeta]$

We here show that for all polynomials $t(n) > 0, \zeta(n) \geq 0$, mild average-case hardness of $\mathsf{McKTP}[t, \zeta]$ implies the existence of OWFs. The proof closely follows the proof in [LP20]; for the reader familiar with the construction in [LP20], the only modification is that the OWF construction now interprets part of its input as the "auxiliary string" z and simply ouputs it.

Theorem 4.1. *Assume that there exist polynomials $t(n) > 0, \zeta(n) \geq 0, p(n) > 0$ such that $\mathsf{McKTP}[t, \zeta]$ is mildly HoA. Then, there exists a weak OWF f (and thus also a OWF).*

Proof: We start with the assumption that $\mathsf{McKTP}[t, \zeta]$ is mildly HoA; that is, there exists a polynomial $p(\cdot)$ such that for all PPT heuristics $\mathcal{H}$ and all sufficiently large n,

$$\Pr[x \leftarrow \{0,1\}^n, z \leftarrow \{0,1\}^{\zeta(n)}, k \leftarrow \{0,1\}^{\lceil \log n \rceil} : \mathcal{H}(x, z, k) = \mathsf{McKTP}[t, \zeta](x, z, k)] < 1 - \frac{1}{p(n)}.$$

Let c be the constant from Fact 2.1. Consider the function $f : \{0,1\}^{n+c+\lceil \log(n+c) \rceil + \zeta(n)} \rightarrow \{0,1\}^*$, which given an input $\ell || \Pi' || z$ where $|\ell| = \lceil \log(n + c) \rceil$, $|\Pi'| = n + c$, and $|z| = \zeta(n)$, outputs $\ell || U(\Pi(z), 1^{t(n)}) || z$ where $\Pi = [\Pi']_\ell$ is the ℓ-bit prefix of Π'. That is,

$$f(\ell || \Pi' || z) = \ell || U(\Pi(z), 1^{t(n)}) || z.$$

Observe that f is only defined over some input lengths, but by an easy padding trick, it can be transformed into a function f' defined over all input lengths, such that if f is (weakly) one-way (over the restricted input lengths), then f' will be (weakly) one-way (over all input lengths): $f'(x')$ simply truncates its input x' (as little as possible) so that the (truncated) input x now becomes of length $m = n + c + \lceil \log(n + c) \rceil + \zeta(n)$ for some n and outputs $f(x)$.

We now show that f is a $\frac{1}{q(\cdot)}$-weak OWF, where $q(n) = 2^{2c+3} n p(n)^2$, which concludes the proof of the theorem. The remaining proof follows exactly the same structure as the proof [LP20] with only minor adjustments to deal with the fact that we now consider conditional Kolmogorov complexity.

Assume for contradiction that f is not a $\frac{1}{q(\cdot)}$-weak OWF. That is, there exists some PPT attacker $\mathcal{A}$ that inverts f with probability at least $1 - \frac{1}{q(n)} \leq 1 - \frac{1}{q(m)}$ for infinitely many $m = n + c + \lceil \log(n + c) \rceil + \zeta(n)$. Fix some such $m, n > 2$. We first claim that we can use $\mathcal{A}$ to construct a PPT heuristic $\mathcal{H}^*$ such that

$$\Pr[x \leftarrow \{0,1\}^n, z \leftarrow \{0,1\}^{\zeta(n)} : \mathcal{H}^*(x, z) = K^t(x \mid z)] \geq 1 - \frac{1}{p(n)}.$$

If this is true, consider the heuristic $\mathcal{H}$ which given a string $x \in \{0,1\}^n$, a string $z \in \{0,1\}^{\zeta(n)}$, and a size parameter $k \in \{0,1\}^{\lceil \log n \rceil}$, outputs 1 if $\mathcal{H}^*(x) \le k$, and outputs 0 otherwise. Note that if $\mathcal{H}^*$ succeeds on some string x, $\mathcal{H}$ will also succeed. Thus,

$$\Pr[x \leftarrow \{0,1\}^n, z \leftarrow \{0,1\}^{\zeta(n)}, k \leftarrow \{0,1\}^{\lceil \log n \rceil} : \mathcal{H}(x,z,k) = \mathsf{McKTP}[t,\zeta](x,z,k)] \ge 1 - \frac{1}{p(n)},$$

which is a contradiction.

It remains to construct the heuristic $\mathcal{H}^*$ that computes $K^t(x \mid z)$ with high probability over random inputs $x \in \{0,1\}^n, z \in \{0,1\}^{\zeta(n)}$, using $\mathcal{A}$. By an averaging argument, except for a fraction $\frac{1}{2p(n)}$ of random tapes r for $\mathcal{A}$, the *deterministic* machine $\mathcal{A}_r$ (i.e., machine $\mathcal{A}$ with randomness fixed to r) fails to invert f with probability at most $\frac{2p(n)}{q(n)}$. Fix some such "good" randomness r for which $\mathcal{A}_r$ succeeds to invert f with probability $1 - \frac{2p(n)}{q(n)}$.

On input $x \in \{0,1\}^n, z \in \{0,1\}^{\zeta(n)}$, our heuristic $\mathcal{H}_r^*(x,z)$ runs $\mathcal{A}_r(i||x||z)$ for all $i \in [n+c]$ where i is represented as a $\lceil \log(n+c) \rceil$ bit string, and outputs the length of the smallest RAM program Π output by $\mathcal{A}_r$ such that $\Pi(z)$ produces the string x within $t(n)$ steps. Let S be the set of pairs $(x,z) \in \{0,1\}^n \times \{0,1\}^{\zeta(n)}$ for which $\mathcal{H}_r^*(x,z)$ fails to compute $K^t(x \mid z)$. Note that $\mathcal{H}_r^*$ thus fails with probability

$$fail_r = \frac{|S|}{2^{n+\zeta(n)}}.$$

Consider any pair $(x,z) \in S$ and let $w = K^t(x \mid z)$ be its conditional K^t-complexity. By Fact 2.1, we have that $w \le n+c$. Since $\mathcal{H}_r^*(x,z)$ fails to compute $K^t(x \mid z)$, $\mathcal{A}_r$ must fail to invert $(w||x||z)$. But, since $w \le n+c$, the output $(w||x||z)$ is sampled with probability

$$\frac{1}{n+c} \cdot \frac{1}{2^w 2^{\zeta(n)}} \ge \frac{1}{(n+c)} \frac{1}{2^{n+c+\zeta(n)}} \ge \frac{1}{n2^{2c+1}} \cdot \frac{1}{2^{n+\zeta(n)}}$$

in the one-way function experiment, so $\mathcal{A}_r$ must fail with probability at least

$$|S| \cdot \frac{1}{n2^{2c+1}} \cdot \frac{1}{2^{n+\zeta(n)}} = \frac{1}{n2^{2c+1}} \cdot \frac{|S|}{2^{n+\zeta(n)}} = \frac{fail_r}{n2^{2c+1}}$$

which by assumption (that $\mathcal{A}_r$ is a good inverter) is at most that $\frac{2p(n)}{q(n)}$. We thus conclude that

$$fail_r \le \frac{2^{2c+2} np(n)}{q(n)}$$

Finally, by a union bound, we have that $\mathcal{H}^*$ (using a uniform random tape r) fails in computing $K^t(x \mid z)$ with probability at most

$$\frac{1}{2p(n)} + \frac{2^{2c+2}np(n)}{q(n)} = \frac{1}{2p(n)} + \frac{2^{2c+2}np(n)}{2^{c+3}np(n)^2} = \frac{1}{p(n)}.$$

Thus, $\mathcal{H}^*$ computes $K^t(x \mid z)$ with probability $1 - \frac{1}{p(n)}$ for infinitely many $n \in \mathbb{N}$ (and therefore $\mathcal{H}$ decides $\mathsf{McKTP}[t,\zeta]$ with probability $1 - \frac{1}{p(n)}$ for infintely many n), which contradicts the assumption that $\mathsf{McKTP}[t,\zeta]$ is $\frac{1}{p(\cdot)}$-HoA. ■

5 Mild Avg-case Hardness of $\mathsf{McKTP}[t,\zeta]$ from OWFs

We here show that that for all polynomial ζ and every polynomial $t(n) \geq 1.1n$, the existence of OWFs implies mild average-case hardness of $\mathsf{McKTP}[t,\zeta]$. Again, our proof closely follows the proof in [LP20] with only minor modifications to deal with the fact that we now consider conditional Kolmogorov complexity.

Theorem 5.1. *If one-way functions exist, then for every constant $\epsilon > 0$, all $t(n) \geq (1+\epsilon)n$, for all polynomial $\zeta(n) \geq 0$, $\mathsf{McKTP}[t,\zeta]$ is mildly HoA.*

Proof: The theorem follows immediately from Theorem 5.4 and Theorem 5.5, which will be stated and proved below. ■

5.1 Some additional preliminaries

Let us first recall some additional standard preliminaries.

Computational Indistinguishability We recall the definition of (computational) indistinguishability [GM84].

Definition 5.2. *Two ensembles $\{A_n\}_{n\in\mathbb{N}}$ and $\{B_n\}_{n\in\mathbb{N}}$ are said to be $\mu(\cdot)$-*indistinguishable*, if for every probabilistic machine D (the "distinguisher") whose running time is polynomial in the length of its first input, there exists some $n_0 \in \mathbb{N}$ so that for every $n \geq n_0$:*

$$|\Pr[D(1^n, A_n) = 1] - \Pr[D(1^n, B_n) = 1]| < \mu(n)$$

We say that are $\{A_n\}_{n\in\mathbb{N}}$ and $\{B_n\}_{n\in\mathbb{N}}$ simply indistinguishable *if they are $\frac{1}{p(\cdot)}$-indistinguishable for every polynomial $p(\cdot)$.*

Statistical Distance and Entropy For any two random variables X and Y defined over some set $\mathcal{V}$, we let $\mathsf{SD}(X,Y) = \frac{1}{2}\sum_{v\in\mathcal{V}} |\Pr[X = v] - \Pr[Y = v]|$ denote the *statistical distance* between X and Y. For a random variable X, let $H(X) = \mathsf{E}[\log \frac{1}{\Pr[X=x]}]$ denote the (Shannon) entropy of X, and let $H_\infty(X) = \min_{x\in Supp(X)} \log \frac{1}{\Pr[X=x]}$ denote the *min-entropy* of X.

5.2 Entropy-preserving PRGs

Liu and Pass [LP20] defined a notion of a *conditionally-secure entropy-preserving pseudorandom generator (cond EP-PRG)*. Roughly speaking, a cond-EP-PRG is a function where the output is indistinguishable from the uniform distribution and also preserves the entropy in the input only when *conditioned on* some event E.

Definition 5.3. *An efficiently computable function $G : \{0,1\}^n \to \{0,1\}^{n+\gamma \log n}$ is a $\mu(\cdot)$-*conditionally secure entropy-preserving pseudorandom generator (μ-condEP-PRG) *if there exist a sequence of events $= \{E_n\}_{n\in\mathbb{N}}$ and a constant α (referred to as the* entropy-loss constant*) such that the following conditions hold:*

- **(pseudorandomness):** *$\{G(\mathcal{U}_n \mid E_n)\}_{n\in\mathbb{N}}$ and $\{\mathcal{U}_{n+\gamma\log n}\}_{n\in\mathbb{N}}$ are $\mu(n)$-indistinguishable;*
- **(entropy-preserving):** *For all sufficiently large $n \in \mathbb{N}$, $H(G(\mathcal{U}_n \mid E_n)) \geq n - \alpha \log n$.*

We say that G has *rate-1 efficiency* if its running time on inputs of length n is bounded by $n + O(n^\varepsilon)$ for some constant $\varepsilon < 1$. When defining running-time, we here mean running-time in terms of a RAM-program computation. [LP20] showed the existence of rate-1 efficient cond EP-PRG; in [LP20] running-time was counted in terms of execution on Turing machine, but we note that identically the same proof shows that the PRG us rate-1 efficient also when run on a RAM.

Theorem 5.4 ([LP20])**.** *Assume that OWFs exist. Then, for every $\gamma > 1$, there exists a rate-1 efficient μ-cond EP-PRG $G_\gamma : \{0,1\}^n \to \{0,1\}^{n+\gamma \log n}$, where $\mu = \frac{1}{n^2}$.*

5.3 Mild Avg-case Hardness of $\mathsf{McKTP}[t, \zeta]$ from Cond EP-PRGs

Theorem 5.5. *Assume that for some $\gamma \geq 4$, there exists a rate-1 efficient μ-condEP-PRG $G : \{0,1\}^n \to \{0,1\}^{n+\gamma \log n}$ where $\mu(n) = 1/n^2$. Then, for every $\epsilon > 0$, all $t(n) \geq (1+\epsilon)n$, for all polynomial $\zeta(n) \geq 0$, $\mathsf{McKTP}[t, \zeta]$ is mildly HoA.*

Proof: The proof follows exactly the same structure as the proof [LP20] with only minor adjustments to deal with the fact that we now consider conditional Kolmogorov complexity. Essentially, the key observation is that random strings have high Kolmogorov complexity also conditionned on any arbitrary string, and due to this observation, essentially the proof in [LP20] can still be applied. We proceed to the full details.

Let $\gamma \geq 4$, and let $G' : \{0,1\}^n \to \{0,1\}^{m'(n)}$ where $m'(n) = n + \gamma \log n$ be a rate-1 efficient μ-condEP-PRG, where $\mu = 1/n^2$. For any constant c, let $G^c(x)$ be a function that computes $G'(x)$ and truncates the last c bits. It directly follows that G^c is also a rate-1 efficient μ-condEP-PRG (since G' is so). Consider any $\varepsilon > 0$ and any polynomial $t(n) \geq (1+\varepsilon)n$ and let $p(n) = 2n^{2(\alpha+\gamma+1)}$.

Assume for contradiction that there exists some PPT $\mathcal{H}(x, z, k)$ that decides $\mathsf{McKTP}[t, \zeta]$ with probability $1 - \frac{1}{p(m)}$ for infinitely many $m \in \mathbb{N}$. Since $m'(n+1) - m'(n) \leq \gamma + 1$, there must exist some constant $c \leq \gamma + 1$ such that $\mathcal{H}$ succeeds (to decide $\mathsf{McKTP}[t, \zeta]$) with probability $1 - \frac{1}{p(m)}$ for infinitely many m of the form $m = m(n) = n + \gamma \log n - c$. Let $G(x) = G^c(x)$; recall that G is a rate-1 efficient μ-condEP-PRG (trivially, since G^c is so), and let $\alpha, \{E_n\}$, respectively, be the entropy loss constant and sequence of events, associated with it.

We next show that $\mathcal{H}$ can be used to break the condEP-PRG G. Towards this, note that a random string still has high K^t-complexity with high probability even if conditioned on a random string: for $m = m(n)$, we have,

$$\Pr_{x \in \{0,1\}^m, z \in \{0,1\}^{\zeta(m)}}[K^t(x \mid z) > m - \frac{\gamma}{2}\log n] \geq \frac{2^m - 2^{m - \frac{\gamma}{2}\log n}}{2^m} = 1 - \frac{1}{n^{\gamma/2}}, \tag{1}$$

since the total number of RAM programs Π with length smaller than $m - \frac{\gamma}{2}\log n$ is only $2^{m-\frac{\gamma}{2}\log n}$, and fix an auxiliary string z, $\Pi(z)$ could output a single string. However, any string output by G, must have "low" K^t complexity no matter what string is conditioned on: For every sufficiently large $n, m = m(n)$, we have that,

$$\Pr_{x \in \{0,1\}^n, z \in \{0,1\}^{\zeta(m)}}[K^t(G(x) \mid z) > m - \frac{\gamma}{2}\log n] = 0, \tag{2}$$

since $G(x)$ can be produced by a RAM program Π with the seed x of length n and the code of G (of constant length) hardwired in it (and the string z is skipped). The running time of Π is bounded by $t(n)$ for all sufficiently large n (since G is rate-1 efficient in the RAM model) , so $K^t(G(x) \mid z) = n + O(1) \leq m - \gamma/2 \log n$ for sufficiently large n (since recall that $\gamma \geq 4$).

Based on these observations, we now construct a PPT distinguisher $\mathcal{A}$ breaking G. On input $1^n, x$, where $x \in \{0,1\}^{m(n)}$, $\mathcal{A}(1^n, x)$ samples $z \leftarrow \{0,1\}^{\zeta(m)}$ and picks $k = m - \frac{\gamma}{2}\log n$. $\mathcal{A}$ outputs

1 if $\mathcal{H}(x, z, k)$ outputs 1 and 0 otherwise. Fix some n, $m = m(n)$, $m'(n) = m + \zeta(m) + \lceil \log m \rceil$ for which $\mathcal{H}$ succeeds to decide $\mathsf{McKTP}[t, \zeta]$ with probability $\frac{1}{p(m)}$. The following two claims conclude that $\mathcal{A}$ distinguishes $\mathcal{U}_{m(n)}$ and $G(\mathcal{U}_n \mid E_n)$ with probability at least $\frac{1}{n^2}$.

Claim 9. *$\mathcal{A}(1^n, \mathcal{U}_m)$ outputs 0 with probability at least $1 - \frac{2}{n^{\gamma/2}}$.*

Proof: Note that $\mathcal{A}(1^n, x)$ will output 0 if (1) x is a string with K^t-complexity larger than $m - \gamma/2 \log n$ conditioned on a randomly sampled string z and (2) $\mathcal{H}$ succeeds on input (x, z, k). Thus,

$$\begin{aligned}
&\Pr[\mathcal{A}(1^n, x) = 0] \\
&\geq \Pr[K^t(x \mid z) > m - \gamma/2 \log n \wedge \mathcal{H} \text{ succeeds on } (x, z, k)] \\
&\geq 1 - \Pr[K^t(x \mid z) \leq m - \gamma/2 \log n] - \Pr[\mathcal{H} \text{ fails on } (x, z, k)] \\
&\geq 1 - \frac{1}{n^{\gamma/2}} - \frac{1}{p(m)} \\
&\geq 1 - \frac{2}{n^{\gamma/2}}.
\end{aligned}$$

where the probability is over a random $x \leftarrow \mathcal{U}_m$, $z \leftarrow \mathcal{U}_{\zeta(m)}$, $k \leftarrow \lceil \log m \rceil$ and the randomness of $\mathcal{A}$ and $\mathcal{H}$. ■

Claim 10. *$\mathcal{A}(1^n, G(\mathcal{U}_n \mid E_n))$ outputs 0 with probability at most $1 - \frac{1}{n} + \frac{3}{n^2}$*

Proof: Recall that by assumption, $\mathcal{H}(x, z, k)$ fails to decide whether $(x, z, k) \in \mathsf{McKTP}[t, \zeta]$ for a random $x \in \{0,1\}^m, z \in \{0,1\}^{\zeta(m)}, k \in \{0,1\}^{\lceil \log m \rceil}$ with probability at most $\frac{1}{p(m)}$.

By an averaging argument, for at least a $1 - \frac{1}{n^2}$ fraction of random tapes r for $\mathcal{H}$ (resp a $1 - \frac{1}{n^2}$ fraction of random choices of z), the deterministic machine $\mathcal{H}_r$ (resp the machine $\mathcal{H}$ with part of input fixed to z) fails to decide $\mathsf{McKTP}[t, \zeta]$ with probability at most $\frac{n^2}{p(m)}$. Fix some "good" randomness r and "good" string z such that $\mathcal{H}_{r,z}$ decides $\mathsf{McKTP}[t, \zeta](\cdot, z, \cdot)$ with probability at least $1 - \frac{n^2}{p(m)}$.

We next analyze the success probability of $\mathcal{A}_{r,z}$. Assume for contradiction that $A_{r,z}$ outputs 1 with probability at least $1 - \frac{1}{n} + \frac{1}{n^{\alpha+\gamma}}$ on input $G(\mathcal{U}_n \mid E_n)$. Recall that (1) the entropy of $G(\mathcal{U}_n \mid E_n)$ is at least $n - \alpha \log n$ and (2) the quantity $-\log \Pr[G(\mathcal{U}_n \mid E_n) = y]$ is upper bounded by n for all $y \in G(\mathcal{U}_n \mid E_n)$. By an averaging argument, with probability at least $\frac{1}{n}$, a random $y \in G(\mathcal{U}_n \mid E_n)$ will satisfy

$$-\log \Pr[G(\mathcal{U}_n \mid E_n) = y] \geq (n - \alpha \log n) - 1.$$

We refer to an output y satisfying the above condition as being "good" and other y's as being "bad". Let $S = \{y \in G(\mathcal{U}_n \mid E_n) : \mathcal{A}_{r,z}(1^n, y) = 0 \wedge y \text{ is good}\}$, and let $S' = \{y \in G(\mathcal{U}_n \mid E_n) : \mathcal{A}_{r,z}(1^n, y) = 0 \wedge y \text{ is bad}\}$. Since

$$\Pr[\mathcal{A}_{r,z}(1^n, G(\mathcal{U}_n \mid E_n)) = 0] = \Pr[G(\mathcal{U}_n \mid E_n) \in S] + \Pr[G(\mathcal{U}_n \mid E_n) \in S'],$$

and $\Pr[G(\mathcal{U}_n \mid E_n) \in S']$ is at most the probability that $G(\mathcal{U}_n \mid E_n)$ is "bad" (which as argued above is at most $1 - \frac{1}{n}$), we have that

$$\Pr[G(\mathcal{U}_n \mid E_n) \in S] \geq \left(1 - \frac{1}{n} + \frac{1}{n^{\alpha+\gamma}}\right) - \left(1 - \frac{1}{n}\right) = \frac{1}{n^{\alpha+\gamma}}.$$

Furthermore, since for every $y \in S$, $\Pr[G(\mathcal{U}_n \mid E_n) = y] \leq 2^{-n+\alpha \log n + 1}$, we also have,

$$\Pr[G(\mathcal{U}_n \mid E_n) \in S] \leq |S| 2^{-n+\alpha \log n + 1}$$

So,

$$|S| \geq \frac{2^{n-\alpha \log n - 1}}{n^{\alpha+\gamma}} = 2^{n-(2\alpha+\gamma)\log n - 1}$$

However, for any $y \in G(\mathcal{U}_n \mid E_n)$, if $\mathcal{A}_{r,z}(1^n, y)$ outputs 0, then by Equation 2, $K^t(y \mid z) \leq m - \gamma/2 \log n = k$, so $\mathcal{H}_{r,z}$ fails to decide $\mathsf{McKTP}[t, \zeta]$ on input $(y, z, m - \gamma/2 \log n)$.

Thus, the probability that $\mathcal{H}_{r,z}$ fails (to decide $\mathsf{McKTP}[t, \zeta]$) on a random input (y, z, k) (where y and k are uniformly sampled in $\{0,1\}^m$ and $\{0,1\}^{\lceil \log m \rceil}$, z is a fixed string) is at least

$$|S|/2^{m+\lceil \log m \rceil} = \frac{2^{n-(2\alpha+\gamma)\log n - 1}}{2^{n+\gamma \log n + \lceil \log m \rceil}} \geq \frac{2^{-(2\alpha+2\gamma)\log n - 1}}{2^{\lceil \log m \rceil}} \geq 2^{-2(\alpha+\gamma+1)\log n - 1} = \frac{1}{2n^{2(\alpha+\gamma+1)}}$$

which contradicts the fact that $\mathcal{H}_{r,z}$ fails to decide $\mathsf{McKTP}[t, \zeta]$ with probability at most $\frac{n^2}{p(m)} < \frac{1}{2n^{2(\alpha+\gamma+1)}}$ (since $n < m$).

We conclude that for every good randomness r, every good choice of string z, $\mathcal{A}_{r,z}$ outputs 0 with probability at most $1 - \frac{1}{n} + \frac{1}{n^{\alpha+\gamma}}$. Finally, by union bound (and since a random tape is bad with probability $\leq \frac{1}{n^2}$ and a random choice of z is bad with probability $\leq \frac{1}{n^2}$), we have that the probability that $\mathcal{A}(G(\mathcal{U}_n \mid E_n))$ outputs 1 is at most

$$\frac{2}{n^2} + \left(1 - \frac{1}{n} + \frac{1}{n^{\alpha+\gamma}}\right) \leq 1 - \frac{1}{n} + \frac{3}{n^2},$$

since $\gamma \geq 2$. ■

We conclude, recalling that $\gamma \geq 4$, that $\mathcal{A}$ distinguishes $\mathcal{U}_m$ and $G(\mathcal{U}_n \mid E_n)$ with probability of at least

$$\left(1 - \frac{2}{n^{\gamma/2}}\right) - \left(1 - \frac{1}{n} + \frac{2}{n^2}\right) \geq \left(1 - \frac{2}{n^2}\right) - \left(1 - \frac{1}{n} + \frac{2}{n^2}\right) = \frac{1}{n} - \frac{4}{n^2} \geq \frac{1}{n^2}$$

for infinitely many $n \in \mathbb{N}$.

■

6 Acknowledgements

We are very grateful to Vinod Vaikuntanathan and Rahul Ilango for helpful comments on an earlier version of this paper.

References

[ABK+06] Eric Allender, Harry Buhrman, Michal Kouckỳ, Dieter Van Melkebeek, and Detlef Ronneburger. Power from random strings. *SIAM Journal on Computing*, 35(6):1467–1493, 2006.

[ACM+21a] Eric Allender, Mahdi Cheraghchi, Dimitrios Myrisiotis, Harsha Tirumala, and Ilya Volkovich. One-way functions and a conditional variant of mktp. *manuscript*, 2021.

[ACM+21b] Eric Allender, Mahdi Cheraghchi, Dimitrios Myrisiotis, Harsha Tirumala, and Ilya Volkovich. One-way functions and partial MCSP. *Electron. Colloquium Comput. Complex.*, 28:9, 2021.

[AD97] Miklós Ajtai and Cynthia Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In Frank Thomson Leighton and Peter W. Shor, editors, *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 284–293. ACM, 1997.

[AGGM06] Adi Akavia, Oded Goldreich, Shafi Goldwasser, and Dana Moshkovitz. On basing one-way functions on NP-hardness. In *STOC '06*, pages 701–710, 2006.

[AHM+08] Eric Allender, Lisa Hellerstein, Paul McCabe, Toniann Pitassi, and Michael E. Saks. Minimizing disjunctive normal form formulas and ac^0 circuits given a truth table. *SIAM J. Comput.*, 38(1):63–84, 2008.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 99–108. ACM, 1996.

[BB15] Andrej Bogdanov and Christina Brzuska. On basing size-verifiable one-way functions on NP-hardness. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part I*, volume 9014 of *Lecture Notes in Computer Science*, pages 1–6. Springer, 2015.

[Blu82] Manuel Blum. Coin flipping by telephone - A protocol for solving impossible problems. In *COMPCON'82, Digest of Papers, Twenty-Fourth IEEE Computer Society International Conference, San Francisco, California, USA, February 22-25, 1982*, pages 133–137. IEEE Computer Society, 1982.

[BM88] László Babai and Shlomo Moran. Arthur-merlin games: A randomized proof system, and a hierarchy of complexity classes. *J. Comput. Syst. Sci.*, 36(2):254–276, 1988.

[Bra83] Gilles Brassard. Relativized cryptography. *IEEE Transactions on Information Theory*, 29(6):877–893, 1983.

[BT03] Andrej Bogdanov and Luca Trevisan. On worst-case to average-case reductions for np problems. In *FOCS '03*, pages 308–317, 2003.

[Cha69] Gregory J. Chaitin. On the simplicity and speed of programs for computing infinite sets of natural numbers. *J. ACM*, 16(3):407–422, 1969.

[DH76] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[FS90] Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *STOC '90*, pages 416–426, 1990.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. On the cryptographic applications of random functions. In *CRYPTO*, pages 276–288, 1984.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[GT00] Rosario Gennaro and Luca Trevisan. Lower bounds on the efficiency of generic cryptographic constructions. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 305–313. IEEE Computer Society, 2000.

[GWXY10] S. Dov Gordon, Hoeteck Wee, David Xiao, and Arkady Yerukhimovich. On the round complexity of zero-knowledge proofs based on one-way permutations. In *LATINCRYPT*, pages 189–204, 2010.

[Har83] J. Hartmanis. Generalized kolmogorov complexity and the structure of feasible computations. In *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*, pages 439–445, Nov 1983.

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

[Hir20] Shuichi Hirahara. Unexpected hardness results for kolmogorov complexity under uniform reductions. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1038–1051. ACM, 2020.

[HMX10] Iftach Haitner, Mohammad Mahmoody, and David Xiao. A new sampling protocol and applications to basing cryptographic primitives on the hardness of NP. In *IEEE Conference on Computational Complexity*, pages 76–87, 2010.

[HOS18] Shuichi Hirahara, Igor Carboni Oliveira, and Rahul Santhanam. Np-hardness of minimum circuit size problem for OR-AND-MOD circuits. In Rocco A. Servedio, editor, *33rd Computational Complexity Conference, CCC 2018, June 22-24, 2018, San Diego, CA, USA*, volume 102 of *LIPIcs*, pages 5:1–5:31. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[IL89] Russell Impagliazzo and Michael Luby. One-way functions are essential for complexity based cryptography (extended abstract). In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*, pages 230–235, 1989.

[Ila19] Rahul Ilango. $ac^0[p]$ lower bounds and np-hardness for variants of MCSP. *Electron. Colloquium Comput. Complex.*, 26:21, 2019.

[Ila20] Rahul Ilango. Approaching MCSP from above and below: Hardness for a conditional variant and ac^0[p]. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPIcs*, pages 34:1–34:26. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[Ila21] Rahul Ilango. Personal communication, 2021.

[ILO20] Rahul Ilango, Bruno Loff, and Igor Carboni Oliveira. Np-hardness of circuit minimization for multi-output functions. In Shubhangi Saraf, editor, *35th Computational Complexity Conference, CCC 2020, July 28-31, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 169 of *LIPIcs*, pages 22:1–22:36. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[Imp95] Russell Impagliazzo. A personal view of average-case complexity. In *Structure in Complexity Theory '95*, pages 134–147, 1995.

[IN89] Russell Impagliazzo and Moni Naor. Efficient cryptographic schemes provably as secure as subset sum. In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*, pages 236–241. IEEE Computer Society, 1989.

[KC00] Valentine Kabanets and Jin-yi Cai. Circuit minimization problem. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 73–79, 2000.

[Ko86] Ker-I Ko. On the notion of infinite pseudorandom sequences. *Theor. Comput. Sci.*, 48(3):9–33, 1986.

[Ko91] Ker-I Ko. On the complexity of learning minimum time-bounded turing machines. *SIAM J. Comput.*, 20(5):962–986, 1991.

[Kol68] A. N. Kolmogorov. Three approaches to the quantitative definition of information. *International Journal of Computer Mathematics*, 2(1-4):157–168, 1968.

[Lev73] Leonid A. Levin. Universal search problems (russian), translated into english by ba trakhtenbrot in [Tra84]. *Problems of Information Transmission*, 9(3):265–266, 1973.

[Liv10] Noam Livne. On the construction of one-way functions from average case hardness. In *ICS*, pages 301–309. Citeseer, 2010.

[LM91] 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.

[LP20] Yanyi Liu and Rafael Pass. On one-way functions and kolmogorov complexity. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1243–1254. IEEE, 2020.

[LP21] Yanyi Liu and Rafael Pass. On the possibility of basing cryptography on exp neq bpp. Manuscript in preparation, 2021.

[MH78] R. Merkle and M. Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions on Information Theory*, 24(5):525–530, 1978.

[Nao91] Moni Naor. Bit commitment using pseudorandomness. *J. Cryptology*, 4(2):151–158, 1991.

[Odl90] A. M. Odlyzko. The rise and fall of knapsack cryptosystems. In *In Cryptology and Computational Number Theory*, pages 75–88. A.M.S, 1990.

[Pas06] Rafael Pass. Parallel repetition of zero-knowledge proofs and the possibility of basing cryptography on np-hardness. In *21st Annual IEEE Conference on Computational Complexity (CCC 2006), 16-20 July 2006, Prague, Czech Republic*, pages 96–110. IEEE Computer Society, 2006.

[Rom90] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *STOC*, pages 387–394, 1990.

[RSA83] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems (reprint). *Commun. ACM*, 26(1):96–99, 1983.

[Sip83] 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.

[Sol64] R.J. Solomonoff. A formal theory of inductive inference. part i. *Information and Control*, 7(1):1 – 22, 1964.

[Tra84] 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.

[Tre01] Luca Trevisan. Non-approximability results for optimization problems on bounded degree instances. In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 453–461. ACM, 2001.

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

[ZL70] 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.

A Proof of Lemma 3.1: Implementing Π in the RAM model

Recall that in Lemma 3.1, we aim at constructing a machine Π that prints $A \in \{0,1\}^{n\times m}$ with running time $\leq t(|A|)$. The machine Π has the values n, m, λ, ℓ, the keys $k_{i_1}, k_{i_2}, \ldots, k_{i_\ell}$ (where each of the keys is of length $\lambda \in O(\log n)$), and the string $z \in (\{0,1\}^{n\times m})^{2^\lambda}$ in its memory. It is guaranteed that

$$z_{k_{i_1}} \vee z_{k_{i_2}} \vee \ldots \vee z_{k_{i_\ell}} = A.$$

We note that the machine Π only has to read $z_{k_{i_1}}, z_{k_{i_2}}, \ldots, z_{k_{i_\ell}}$ from the string z using the keys $k_{i_1}, k_{i_2}, \ldots, k_{i_\ell}$, and outputs the bit-wise or of those strings. (Note that the length of z could be large than $t(|A|)$, Π's running time bound, and thus a Turing machine cannot finish the algorithm in time $t(|A|)$.)

We show that there exists a CPU "next-step" machine M receiving a state with $O(\log n)$ bits that implements the above algorithm. Recall that in each CPU step, M receives a state state and some "read bit" b^{read} as input, and outputs a new state state', a read position i^{read}, a write position i^{write}, and some bit b^{write}. The execution of Π will replace state with the new state state', b^{read} with the content of memory position i^{read}, and replace the content of memory position i^{write} by b^{write}.

When designing a CPU next-step machine M, it is instructive to view the state state as a "snapshot" of some registers of Π, and each register could store a $O(\log n)$ bit string. Thus, we can view the machine M as a machine that receives the values in the registers as input, and outputs some new value for each register. Thus, we are equipped with some registers, and our implementation is as follows.

- M loads n, m, λ into some registers.
- M creates a new register j, initialized with 1. M makes a loop with j going from 1 to ℓ. In the j-th iteration, M loads k_{i_j} into a register.
- In the j-th iteration, M creates two new register p and L, and L is set to be a large enough integer such that the contents in memory positions $\geq L$ are $\perp$.
- M makes a loop with p going from 1 to $n \times m$. M loads $z_{k_{i_j},p}$ into a register.
- In the p-th iteration, if the memory is empty in position $L+p$, M moves $z_{k_{i_j},p}$ to the memory position $L+p$. Otherwise, M replace the content of memory position $L+p$ with its binary-or with $z_{k_{i_j},p}$.
- The two loops end here.
- Finally, M outputs the string saved from memory position $L+1$ to memory position $L+nm$.

Note that the above procedure (as a RAM program) takes $O(\ell nm)$ CPU steps, but in each CPU step, the machine M is only in charged of computing the new values in the registers (and the values of i^{read}, i^{write}, b^{write}) from the old values in the registers (and the value of b^{read}). And we only have constant number of registers, and each of them is of size $O(\log n)$. Thus, in each CPU step, the computation of M is polynomial in its input size ($O(\log n)$), and the total running time of the RAM program is $O(\ell nm \mathsf{poly} \log n)$.