Electronic Colloquium on Computational Complexity, Report No. 175 (2023)

The Non-Uniform Perebor Conjecture for Time-Bounded Kolmogorov Complexity is False[*]

Noam Mazor [†] Rafael Pass [‡]

November 15, 2023

Abstract

The *Perebor* (Russian for "brute-force search") conjectures, which date back to the 1950s and 1960s are some of the oldest conjectures in complexity theory. The conjectures are a stronger form of the $\mathbf{NP} \neq \mathbf{P}$ conjecture (which they predate) and state that for "meta-complexity" problems, such as the *Time-bounded Kolmogorov complexity Problem*, and the *Minimum Circuit Size Problem*, there are no better algorithms than brute force search.

In this paper, we disprove the *non-uniform* version of the Perebor conjecture for the Time-Bounded Kolmogorov complexity problem. We demonstrate that for every polynomial $t(\cdot)$, there exists of a circuit of size $2^{4n/5+o(n)}$ that solves the $t(\cdot)$-bounded Kolmogorov complexity problem on *every* instance.

Our algorithm is *black-box* in the description of the Universal Turing Machine employed in the definition of Kolmogorov Complexity, and leverages the characterization of one-way functions through the hardness of the time-bounded Kolmogorov complexity problem of Liu and Pass (FOCS'20), and the time-space trade-off for one-way functions of Fiat and Naor (STOC'91). We additionally demonstrate that no such black-box algorithm can have sub-exponential circuit size.

Along the way (and of independent interest), we extend the result of Fiat and Naor and demonstrate that any efficiently computable function can be inverted (with probability 1) by a circuit of size $2^{4n/5+o(n)}$; as far as we know, this yields the first formal proof that a non-trivial circuit can invert any efficient function.

[*]This paper appears in ITCS 2024.

[†]Cornell Tech. E-mail: `noammaz@gmail.com`. Research partly supported by NSF CNS-2149305 and NSF CNS-2128519.

[‡]Tel-Aviv University and Cornell Tech. E-mail: `rafaelp@tau.ac.il`. Supported in part by NSF Award CNS 2149305, AFOSR Award FA9550-18-1-0267, AFOSR Award FA9550-23-1-0387 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, DARPA or the AFOSR.

ISSN 1433-8092

1 Introduction

In his historical account, Thaktenbrot [Tra84], describes efforts in the 1950s and 1960s in the Russian Cybernetics program to understand problem that requiring *brute-force search* to solve. [Tra84; Yab59a; Yab59b]. The so-called *Perebor* (Russian for brute-force search) conjectures refer to the conjectures that certain types of, what today are referred to as "meta-complexity", problems require brute-force search to be solve. These include (a) the *Minimimum Circuit Size problem* (MCSP) [KC00; Tra84]—finding the smallest Boolean circuit that computes a given function x, and (b) the *Time-Bounded Kolmogorov Complexity Problem* [Kol68; Sol64; Cha69; Ko86; Har83; Sip83]—computing the length, denoted $\mathrm{K}^t(x)$ of shortest program (evaluated on some particular Universal Turing machine U) that generates a given string x, within time $t(|x|)$, where t is a polynomial. The Perebor conjectures state that solving these problems require an algorithms with running time (in the *uniform* regime), or circuit size (in the *non-uniform* regime) close to 2^n where $n = |x|$ is the size of the given instance x.

The Perebor conjecture can be viewed as an early precursor, and stronger form, of the $\mathbf{NP} \neq \mathbf{P}$ conjecture (as these meta-complexity problems reside in $\mathbf{NP}$). In this work, we focus on the Time-Bounded Kolmogorov Complexity Problem, and the stronger *non-uniform* version of the Perebor conjecture for the time-bounded Kolmogorov Complexity problem.

1.1 Our Results

Circuit Complexity Upperbounds for Solving K^t Our main result disproves the non-uniform Perebor conjecture for Time-bounded Kolmogorov Complexity:

Theorem 1.1 (Main theorem). *For every $t \in \text{poly}$, there exists a circuit family $\mathcal{C} = \{C_n\}_{n\in\mathbb{N}}$ of size $\tilde{O}(2^{4n/5})$ such that, for every $n \in \mathbb{N}$ and for every $x \in \{0,1\}^n$, $C_n(x) = \mathrm{K}^t(x)$ (That is, the circuits computes $\mathrm{K}^t(x)$ on every instance x.)*

In fact, our theorem is stronger than stated: we not only show how to compute K^t but also to find a so-called K^t-witness (i.e., a shortest-length program generated the string x)—this was referred to as the *constructive* version of the problem in [Tra84].

We highlight that the fact that a non-uniform Perebor conjecture may be false may perhaps not be shocking to experts; indeed, Ren and Santhanam [RS21] recently considered an *average-case strengthening* of the non-uniform Perebor conjecture for the Time-bounded Kolmogorov complexity and noted that by the recent connection [LP20] between average-case hardness of the time-bounded Kolmogorov complexity problem and the cryptographic notion of *one-way functions* [DH76], there is evidence pointing to this conjecture being false:

> *However, building on Hellman [Hel80], Fiat and Naor [FN00] showed that no such one-way function exists in the non-uniform RAM model. In particular, for any function $f : \{0,1\}^n \to \{0,1\}^n$, there is an algorithm that runs in $2^{3n/4}$ time, with random access to an advice tape of length $2^{3n/4}$, and inverts f at any point. It is conceivable that a similar attack could also be implemented in circuits, i.e. every function f could be inverted by a circuit of size $2^{99n/100}$ in the worst-case. This gives strong evidence that the non-uniform version of Hypothesis 5.18 is false.*

As far as we know, such a circuit implementation of [FN00] is not known—indeed, as we discuss below, the natural implementation of their algorithm as a circuit leads to a "trivial" size circuit, and does not beat Perebor.

Our main technical result is a non-trivial implementation of the Fiat-Naor algorithm using a circuit of size $\tilde{O}(2^{4n/5})$ (note that this is larger than the running-time/advice size of $\tilde{O}(2^{3n/4})$ acheived by Fiat and Naor):

Theorem 1.2. *For every efficiently computable function $f : \{0,1\}^n \to \{0,1\}^n$, there exists a circuit family $\mathcal{C} = \{C_n\}_{n\in\mathbb{N}}$ of size $\tilde{O}(2^{4n/5})$ such that, for every $n \in \mathbb{N}$ and for every $x \in \{0,1\}^n$, $f(C_n(f(x)) = f(x)$ (That is, the circuits inverts f on every point in the range of f).*

We next combine Theorem 1.2 with the one-way function construction of [LP20] to conclude Theorem 1.1—leveraging the fact that the one-way function construction is length-preserving and thus there is little loss in the parameters.

Circuit Complexity Lowerbounds for *Black-box* Solving K^t. We highlight that our K^t solving circuit is "black-box" in the description of the Universal Turing machine U employed in the definition of Kolmogorov complexity (or more precisely, the circuit can be implemented having simply oracle access to U)—we refer to such circuits as being a *black-box* K^t *solver*. We next demonstrate that subexponential-size black-box K^t solvers cannot exists, and as such our non-uniform K^t solver is "optimal" up to a constant in the exponent w.r.t. such black-box algorithms.

Theorem 1.3 (Informal)**.** *Every black-box* K^t*-solver must have circuit size at least* $2^{n/2-o(n)}$.

1.2 Proof Outline

We here provide a proof outline of Theorems 1.1 and 1.2.

1.2.1 Constructing a One-way Function from K^t

As mentioned above, our starting point is the one-way function construction of [LP20] from the Time-bounded Kolmogorov Complexity problem. We note that this construction trivially works also in the worst-case regime (i.e., any worst-case inverted for the one-way function also solves K^t in the worst-case). But most importantly, this construction is *length preserving*: any algorithm breaking the one-way function of inputs of length $n + O(\log n)$ solves the K^t problem on inputs of length n. This observation will be crucial as it means that the upperbound that we get for inverting the one-way function will directly translate to (essentially) the same bound for K^t.

1.2.2 The Hellman/Fiat-Naor Algorithm

The Hellman Algorithm Before describing the Fiat-Naor algorithm for inverting arbitrary functions, we describe the algorithm of Hellman [Hel80] for inverting a permutation. The main idea of Hellman is that for every permutation $f\colon [2^n] \to [2^n]$, and every image $y \in [2^n]$, if we compute the chain $(y, f(y), f(f(y)), \dots)$, at some point we must find a pre-image of y. Namely, there exists some $k \geq 1$ such that $f^k(y) = y$, and thus $f^{k-1}(y) = f^{-1}(y)$. Of course, this k can be large, and it could even be that $k = 2^n$. In this case, computing the chain $(y, f(y), f(f(y)), \dots)$ takes the same time as brute-force search.

To overcome this, Hellman's algorithm uses an advice string. For a parameter T, this advice string contains S entries of the form $(x, f^T(x))$. Then, to find a pre-image of y, the algorithm computes the chain of length T, $(y, f(y), \ldots, f^T(y))$. If one of the points in the chain is a pre-image of y, the algorithm finishes. Otherwise, the algorithm looks for an entry $(x_i, f^T(x_i))$ in the advice string, such that $f^T(x_i)$ appears in the chain $(y, f(y), \ldots, f^T(y))$. If it finds such an entry, then it holds that $f^k(y) = f^T(x_i)$ for some $k \leq T$, or equivalently, $y = f^{T-k}(x_i)$. The algorithm then can find the pre-image of y by computing $f^{T-k-1}(x_i)$.

Every such entry $(x_i, f^T(x_i))$ in the advice can be used to invert the T images $(f(x_i), \ldots f^T(x_i))$ using $O(T)$ calls to f. Hellman showed that we can find at most $S = N/T$ points $x_1, \ldots x_S$, such that the advice $((x_1, f^T(x_1)), \ldots, (x_S, f^T(x_S)))$ can be used to invert *any* image with $O(T)$ calls to f. This allows to invert any such permutation in time $\tilde{O}(2^{n/2})$ using an advice of length $\tilde{O}(2^{n/2})$.

When $f\colon [2^n] \to [2^n]$ is an arbitrary function, it is no longer true that $y = f^{T-k}(x_i)$ if $f^k(y) = f^T(x_i)$. Moreover, it might be impossible to find a non-trivial number of points $x_1, \ldots, x_S$ such that the T-length chains starting from these points cover all the images of the function. However, Hellman showed under some assumptions on the structure of f (that is, that f is "random enough"), that a similar algorithm to the above can work with slightly worse parameters. Fiat and Naor later generalized Hellman's result to work with any function f.

The Fiat-Naor Algorithm. Let $f\colon [2^n] \to [2^n]$ be an arbitrary function. The first step in the Fiat-Naor algorithm is to remove from f any image with too many pre-images. To do so, for a parameter $U \in \mathbb{N}$, let $\mathcal{A} = \{y_1, \ldots, y_r\}$ be the set of all images y_i with more than U pre-images. Observe that the size of $\mathcal{A}$, r, is at most $2^n/U$. The Fiat-Naor algorithm adds to the advice a pre-image x_i of every $y_i \in \mathcal{A}$. By saving $(x_i, y_i)_{y_i \in \mathcal{A}}$ in the advice, we can trivially invert any image y_i. Thus, we are left to invert f over the domain $\mathcal{D} = \{x\colon f(x) \notin \mathcal{A}\}$ of all inputs with a small number of "siblings". Note that over the domain D, every image of f has at most U pre-images. In the following, we restrict attention to functions with a small number of heavy images. That is, we assume that $|\mathcal{D}| \geq 2^n/2$. (Fiat and Naor deal with this issue later using a different parametrization, which would lead to further complications in our setting. Nevertheless, as we shall explain in more detail below, we shall observe that focusing on the above "simplified" case actually is without loss of generality.)

Next, [FN00] proceed by presenting an algorithm A with advice of length $\tilde{O}(m)$ that makes $\tilde{O}(t)$ calls to f. For some choices of the parameters m and t, [FN00] presented a distribution over advices, such that for every image $y \in f(\mathcal{D})$ and for a random advice, A successfully invert y with probability roughly mt/N. By running the algorithm $\ell = \tilde{O}(N/mt)$ times with ℓ random advices, [FN00] get an algorithm that inverts *all* images $y \in f(\mathcal{D})$ simultaneously with high probability. The resulting algorithm has advice of length $S = \ell \cdot m + |\mathcal{A}|$, and it makes $T = \ell \cdot t$ calls to the function f.

We next describe the algorithm A. To make the function f behave more randomly, Fiat-Naor uses a hash function $g\colon [2^n] \to [\mathcal{D}]$. Then, the algorithm actually tries to invert the function $h = g \circ f$. Choosing the function g such that it will have a succinct description and yet will be easy to evaluate is an important part of the Fiat-Naor algorithm. But for now, we can think of g as a random function. As in Hellman's algorithm, the advice contains $(x_i, h^t(x_i))$ for m randomly chosen points $x_1, \ldots, x_m$.

Given the advice $(x_i, h^t(x_i))_{i \in [m]}$, the function g and an input $y = f(x)$ to invert, the algorithm A starts with computing $g(y) = h(x)$. Next, the algorithm proceed by computing the t-length chain $(h(x), h^2(x), \ldots, h^t(x))$. Then, for every $i \in [m]$ such that $h^t(x_i) = h^k(x)$ (that is, $h^t(x_i)$ appears in the chain $(h(x), h^2(x), \ldots, h^t(x))$), A computes $x' = h^{t-k}(x_i)$ and checks if $f(x') = y$. Importantly, when f is not a permutation, it is possible that $f(x') \neq y$. In this case, we say that i is a false-positive index.

The Correctness Proof. Fiat and Naor showed that for every $y \in f(\mathcal{D})$, when the parameters are chosen such that $mt^2U \approx 2^n$, and over a random choice of the function g and the points $x_1, \ldots, x_m \leftarrow [2^n]$, the algorithm succeeds in finding a pre-image of y with probability roughly mt/N. Moreover, to bound the running time of the algorithm Fiat and Naor showed that the expected number of false-positive indexes is constant. For the convenience of the interested reader, we give a high level sketch of the proof in Appendix A (but this will not be relevant for the rest of the paper—we will use this part of Fiat-Naor in a black-box way).

The Function g. We next describe how to choose the function g used by A. As mentioned above, we want g to have a succinct description and we need to be able to evaluate it efficiently. Recall we want g to be from $[2^n]$ to $\mathcal{D}$. However, the range $\mathcal{D}$ is defined by the function f, and it thus does not have an efficient representation. However, given the set $\mathcal{A}$, and the function f, it is easy to check if a point x is in $\mathcal{D}$: simply check if $f(x)$ is not in $\mathcal{A}$. Using this fact, to construct the function g, Fiat and Naor [FN00] constructed first a function $g' \colon [2^n] \times [2^n] \to [2^n]$. Then, they defined $g(x)$ to be $g'(x, z)$ for the minimal $z \in [2^n]$ for which $g'(x, z) \in \mathcal{D}$. By our assumption that $\mathcal{D}$ is large (at least $2^n/2$), if g' is a random function (or $\Omega(1)$-wise independent), the expectation of this z is $O(1)$. Thus, we will not need to evaluate g' too many times to evaluate g. Moreover, in the RAM model, checking if $g'(x, z) \in \mathcal{D}$ can be done efficiently (using hash tables or binary search), and thus evaluating g has roughly the same cost as evaluating g'.

For the function g', [FN00] used a random $O(t)$-degree polynomial. Such a polynomial has a (relatively) succinct description, and the resulting function g' is $O(t)$-wise independent, which is enough for their analysis to go through. However, evaluating g' on a single point takes $O(t)$ time, which is more than we can afford.

[FN00] solved this issue by exploiting the fact that we run the algorithm A ℓ times in parallel, and by using the FFT algorithm. Specifically, as we actually need to construct ℓ functions, $g'_1, \ldots, g'_\ell$. [FN00] constructed these polynomials in a correlated way, such that the *amortized* cost of evaluating each polynomial g'_i on a point x_i will be $\tilde{O}(1)$. In other words, computing $g'_1(x_1), \ldots, g'_\ell(x_\ell)$ simultaneously can be done in time $\tilde{O}(\ell)$.

The Time and Space Complexity. As mentioned above, the Fiat-Naor algorithms has time $T = \tilde{O}(\ell \cdot t)$ and space $S = \tilde{O}(\ell \cdot m + |\mathcal{A}|)$, where $|\mathcal{A}| \leq 2^n/U$ and $\ell = 2^n/mt$. That is, $T = \tilde{O}(2^n/m)$ and space $S = \tilde{O}(2^n/t + 2^n/U)$. Thus, [FN00] can choose parameters such that $U = t$. Moreover, by the analysis above we need to choose parameters such that $mt^2U < 2^n$. To minimize $T + S$, we want to choose parameters such that $2^n/m = 2^n/t = 2^n/U$, which imply that $m = t = U = 2^{n/4}$. In this case, we get that $S = T = 2^{3n/4}$.

On Implementing the Attack as a Circuit. When trying to implement the Fiat-Naor algorithm as a circuit, the main difference from the above is in the evaluation of the functions $g_1, \ldots, g_\ell$. Specifically, to evaluate g_i on a point x, we need to evaluate the function g'_i on x, z for multiple values of z, and for each one, to check if $f(g'_i(x, z))$ appears in the list $\mathcal{A}$. While in the RAM model checking if $f(g_i'(x,z))$ appears in the list $\mathcal{A}$ can be done in $\log(|\mathcal{A}|)$ time, with a circuit we must use a circuit of linear size. Since we evaluate $g_1, \dots, g_\ell$ overall $\Omega(t \cdot \ell)$ times , the total cost of the above lookup is $\Omega(|\mathcal{A}| \cdot t \cdot \ell) = \Omega(2^n/U \cdot 2^n/m)$. By the constraint that $mt^2U < 2^n$, we get that the resulting circuit size is at least $2^n \cdot t^2$, which is more than the trivial size.

1.3 Our Circuit Implementation

To solve the above issues, we first construct a new primitive that we call *batched look-up tables*, and show that it is possible to look-up for multiple values in the list $\mathcal{A}$ with a small amortized cost.

Then, we show that by choosing different parameters then Fiat and Naor [FN00], we can obtain a non-trivial circuit.

Batched Look-Up Tables Informally, Batched Look-Up circuits allow to search multiple entries in a list, with a small amortized cost. In more detail, let $\mathcal{T} = (a_1, \dots, a_r)$ be a target list, and let $\mathcal{B} = (x_1, \dots, x_\ell)$ be the inputs. We want to check for each x_i if it appears in $\mathcal{T}$. The batch look-up circuit returns a list of bits $(b_1, \dots, b_\ell)$ such that b_i indicates if $x_i \in \mathcal{T}$.

We show that there is a look-up circuit of size $\tilde{O}(|\mathcal{T}| + |\mathcal{B}|)$. It follows that, when $\mathcal{B}$ is long enough, the amortized cost of each search is $\tilde{O}(1)$. We implement this circuit in three steps:

First, we modify the list $\mathcal{B}$ such that each element in the list contains the index. That is, we let $\mathcal{B}' = ((x_1, 1), \dots, (x_\ell, \ell))$. We also let $\mathcal{T}' = ((a_1, 0), \dots, (a_t, 0))$. Then, we use a sorting circuit to sort the union of the lists (according to the first entry of each element, and then the second). Let $\mathcal{R}_1$ be the output of this step.

Next, we check for each element (x, i) in $\mathcal{R}'$ if there exists an element of the form $(x, 0)$ earlier in the list. If there is, we change the entry to $(1, i)$. If there is no such element, we change the entry to $(0, i)$. Since the list is sorted, this can be done with an efficient algorithm that reads the list in one pass, and thus can be implemented with a small circuit. After this step, the list contains one entry of the form (b, i) for every $i \in [\ell]$, where the value of b is 1 if and only if x_i appears in $\mathcal{T}$.

Lastly, we sort the list again, this time according to the indexes, and output the first entry of each of the last ℓ elements in the sorted list.

Changing the Parametrization Using the batched look-up circuit, we can now compute the functions $g_1, \dots, g_\ell$ in a small amortized cost in parallel. Specifically, we can evaluate all the ℓ functions, each on one point, in cost of $\tilde{O}(\ell + |\mathcal{A}|)$. However, to compute a chain of length t using the function g_i, the algorithm needs to evaluate g_i sequentially. Recall that to evaluate g_i on a point x, we need to find z such that $f(g_i'(x,z)) \notin \mathcal{A}$. Thus the look-up step must be performed between every two consecutive calls g_i. So while we can batch calls across chains, in each chain we must perform the look-up separately for each call. This implies that we need to apply the batch look-up circuit t times, each time on ℓ calls. Thus, we need to pay $\tilde{O}((r + \ell)t)$ for this step. For the parameter choice of Fiat and Naor [FN00], when $r = S = \ell \cdot m$, we get that $t \cdot r = 2^n$, and thus the circuit size is again trivial.

Fortunately, we can choose the parameters differently to get a better circuit size. Specifically, we choose r to be much smaller than $\ell \cdot m$. In more detail, our final circuit size is $\tilde{O}((r + \ell)t + m \cdot \ell)$.[1] Recall that $\ell = 2^n/mt$, $r = 2^n/U$, and we need to choose parameters such that $mt^2U < 2^n$, or

[1]The term $m \cdot \ell$ is since in the Fiat-Naor algorithm, we need to find all the indexes j such that $(x_j, h^t(x_j))$ is in the advice and $h^t(x_j)$ appears in the chain starting with y. For this step we also use a batch look-up circuit.

equivalently $mt^2 < r$. Taking $m = t$ and $r = \ell = 2^n/t^2$, we get that for $t < 2^{n/5}$ the above constrain holds. In this case, the circuit size is $\tilde{O}(2^{4n/5})$.

Other Issues We note that the above outline oversimplifies a bit, and we are also require to reanalyze certain aspect of the Fiat-Naor algorithm (on top of the above changes).

Inverting General Functions The above approach only work when most points in the range of the function has a small number of pre-images. Fiat and Naor show how to deal also with the more general case, but this requires a more subtle choice of parameters and this would complicate things for us. To overcome this issue (and of independent interest), we observe that any function $f\colon [2^n] \to [2^n]$ can be converted into a function $f'\colon [2 \cdot 2^n] \to [2 \cdot 2^n]$ satisfying the small-image requirement, in a way that inverting f can be reduced to inverting f'. Indeed, we can simply let $f'(x) = f(x)$ for every $x \in [2^n]$, and $f'(x) = x$ otherwise. As a result, get that every function f can be inverted with a circuit of size $\tilde{O}(2^{4n/5})$.

2 Preliminaries

2.1 Notations

All logarithms are taken in base 2. We use calligraphic letters to denote sets and distributions, uppercase for random variables, and lowercase for values and functions. Let poly stand for the set of all polynomials. Given a vector $v \in \Sigma^n$, let v_i denote its i^{th} entry, let $v_{<i} = (v_1, \ldots, v_{i-1})$ and $v_{\leq i} = (v_1, \ldots, v_i)$. Similarly, for a set $\mathcal{I} \subseteq [n]$, let $v_{\mathcal{I}}$ be the ordered sequence $(v_i)_{i \in \mathcal{I}}$. For a function $f\colon \mathcal{D} \to \mathcal{R}$, and a set $\mathcal{S} \subseteq \mathcal{D}$, we let $f(\mathcal{S}) = \{f(x)\colon x \in \mathcal{S}\}$.

2.2 Distributions and Random Variables

When unambiguous, we will naturally view a random variable as its marginal distribution. The support of a finite distribution $\mathcal{P}$ is defined by $\mathrm{Supp}(\mathcal{P}) := \{x\colon \Pr_{\mathcal{P}}[x] > 0\}$. For a (discrete) distribution $\mathcal{P}$, let $x \leftarrow \mathcal{P}$ denote that x was sampled according to $\mathcal{P}$. Similarly, for a set $\mathcal{S}$, let $x \leftarrow \mathcal{S}$ denote that x is drawn uniformly from $\mathcal{S}$.

We will use k-wise independent functions in the construction.

Definition 2.1 (k-wise independent). *For a set $\mathcal{R}$, n random variables $Y_1, \ldots, Y_n$ over Ω are* k-wise independent *if for every indexes $i_1, \ldots, i_k$, the joint distribution $Y_{i_1}, \ldots, Y_{i_k}$ is the uniform distribution over $\mathcal{R}^k$.*

A function family $G = \{g\colon [N] \to \mathcal{R}\}$ is k-wise independent if the distribution of $g(1), \ldots, g(N)$, for $g \leftarrow G$ is k-wise independent.

2.3 Kolmogorov Complexity

Roughly speaking, the *t-time-bounded Kolmogorov complexity*, $\mathrm{K}^t(x)$, of a string $x \in \{0,1\}^*$ is the length of the shortest program $\Pi = (M, y)$ such that, when simulated by an universal Turing machine, Π outputs x in $t(|x|)$ steps. Here, a program Π is simply a pair of a Turing Machine M and an input y, where the output of P is defined as the output of $M(y)$. When there is no running time bound (i.e., the program can run in an arbitrary number of steps), we obtain the notion of Kolmogorov complexity.

In the following, let $\mathsf{U}(\Pi, 1^t)$ denote the output of Π when emulated on U for t steps. We now define the notion of Kolmogorov complexity with respect to the universal TM U.

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

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

where $|\Pi|$ is referred to as the description length *of* Π.

We will use the following bound on the Kolmogorov complexity of strings sampled from the uniform distribution.

Lemma 2.3. *For any universal TM U and every $n \in \mathbb{N}$, it holds that*

$$\Pr_{x \leftarrow \{0,1\}^n}\left[\mathrm{K}^t_{\mathsf{U}}(x) \geq n - i\right] \geq 1 - 2^{-i}.$$

In this paper, unless otherwise stated, we fix some universal Turing machine U that can emulate any program Π with polynomial overhead, and let $\mathrm{K}^t = \mathrm{K}^t_{\mathsf{U}}$.

2.4 Circuits

In this paper we consider circuits over the De-Morgan Basis, which contains the following gates: $\wedge$ ("and" gate with fan-in 2), $\vee$ ("or" gate with fan-in 2), and $\neg$ ("not" gate with fan-in one). The size of a circuit C, denoted by $|C|$ is the number of gates in C. We will sometime allow to use also f-gates in the circuits, for a function $f\colon \{0,1\}^n \to \{0,1\}^m$.

Definition 2.4 (Circuit with f-gates)**.** *For a function $f\colon \{0,1\}^n \to \{0,1\}^m$, a* circuit with f-gates *is a circuit which have, in addition to $\wedge, \vee$ and $\neg$ gates, a special f gates. Each f-gate has fan-in n, where each in-wire is label (uniquely) by a number in $1, \ldots, n$. Each out-wire is labeled by a number in $1, \ldots, m$. The value of a out-wire labeled by i is the i-th bit of $f(w_1, \ldots, w_n)$, where w_j is the value of the in-wire labeled by j.*

The following lemma is a non-uniform version of the fast evaluation algorithm used by [FN00] (see for example [Bav12]), stating that it is possible to efficiently evaluate a polynomial on multiple points. This lemma will help us to get better parameters in our construction.

Lemma 2.5 (Evaluating a polynomial on multiplied points.)**.** *There exists a constant c such that the following holds. Let p be an arbitrary degree d polynomial in a field of size 2^{2n}, for $n \geq 1$. Then there exists a circuit C of size $d \cdot (n + \log d)^c$ such that $C(x_1, \ldots, x_d) = (p(x_1), \ldots, p(x_d))$ for every d field elements $x_1, \ldots, x_d$.*

The next lemma states that it is possible to sort a list of integers by an nearly linear size circuits.

Lemma 2.6 (Sorting circuits [AKS83])**.** *There exists a circuit family of size $O(n \cdot \mathrm{poly}(\log n, k))$ that sorts n numbers of k-bits each.*

We will also use the following simple lemma, that states that any computation on a list that can be done using efficient, one-pass algorithm, can be done also with a small circuit.

Lemma 2.7 (Circuit of an one-pass algorithm.)**.** *There exists a universal constant c such that the following holds. Let A be an algorithm that gets as input a list $\mathcal{L}$ of length ℓ of n-bits elements, and output a list $\mathcal{L}'$ of length ℓ of n-bits elements. Furthermore, assume that A reads $\mathcal{L}$ in one pass: for $i \in [\ell]$, in the i-th step of the algorithm, A reads the i-th entry of $\mathcal{L}$, and apply a s-sized circuit on the entry and its own state to compute the i-th entry of $\mathcal{L}'$ together with its new state. Then the function computed by A can be computed by a $c\ell \cdot s$-size circuit.*

3 The Fiat-Naor Algorithm

In this part we describe the algorithm of Fiat and Naor [FN00], and show how to implement it using a small circuit. For this implementation, we need to choose different parameters than the one used in [FN00]. For a function $f\colon [2^n] \to [2^n]$, our goal is to construct a small circuit that inverts *any* image of f. We prove the following theorem.

Theorem 3.1. *There exists a constant c such that the following holds. Let $n \in \mathbb{N}$ be a number, and let $f\colon [2^n] \to [2^n]$ be a function. Then there exists a circuit C of size $c \cdot 2^{4n/5} \cdot n^c$ with f-gates, such that*

$$\Pr_{x \leftarrow [2^n], y := f(x)}\left[C(y) \in f^{-1}(y)\right] = 1.$$

Following Fiat and Naor [FN00], we start with focusing on functions which are not *trivial to invert*. Informally, for a parameter $U \in \mathbb{N}$, a function f is said to be U-trivial to invert if there are many images with more than U pre-images.

Definition 3.2 (Trivial to invert function, [FN00])**.** *Let $U, n \in \mathbb{N}$ be two numbers. A function $f\colon [2^n] \to [2^n]$ is U-trivial to invert if*

$$\left|\left\{x\colon \left|f^{-1}(f(x))\right| < U\right\}\right| < 2^n/2.$$

Notice that if f is U-trivial to invert, it also U'-trivial to invert, for every $U' < U$. Such a function f which is U-trivial to invert, can inverted with probability $1/2$ (over a random input) using a relatively short advice, by keeping one pre-image for every image with more than U pre-images. As a simplification of independent interest[2], we observe that any function $f\colon [2^n] \to [2^n]$ can be converted to a function $f'\colon [2 \cdot 2^n] \to [2 \cdot 2^n]$ which is not U-trivial to invert for any $U > 1$. Indeed, define $f'(x)$ be equal to $f(x)$ if $x \in [2^n]$, or equal to x otherwise. It immediately follows that f is injective on at least $1/2$ of the domain. Moreover, inverting f is equivalent to inverting f'. Thus, the assumption that f is not trivial to invert is without loss of generality.

In the first step, following [FN00], we construct a *distribution* $\mathcal{P}$ over circuits of size $\tilde{O}(2^{4n/5})$, such that, for every $y \in f([2^n])$, it holds that $\Pr_{C \leftarrow \mathcal{P}}\left[C(y) \in f^{-1}(y)\right] \geq 1/10$. Then, by sampling $O(\log n)$ circuits from $\mathcal{P}$ and combining them together, with high probability we get a single circuit that inverts *any* image y. The main lemma and the derivation of Theorem 3.1 are formally stated next.

Lemma 3.3. *There exists a constant c such that the following holds. Let $n \in \mathbb{N}$ be a number, and let $f\colon [2^n] \to [2^n]$ be a function which is not $2^{n/5}$-trivial to invert. There exists a distribution $\mathcal{P}$ over circuits of size at most $c \cdot 2^{4n/5} \cdot n^c$ with f-gates, such that for every $y \in f([2^n])$,*

$$\Pr_{C \leftarrow \mathcal{P}}\left[C(y) \in f^{-1}(y)\right] \geq 1/10.$$

Theorem 3.1 follows easily by Lemma 3.3.

Proof of Theorem 3.1. Let $n' = n + 1$ and $f'\colon [2^{n'}] \to [2^{n'}]$ be as defined above. Let $\mathcal{P}$ be the distribution promised by Lemma 3.3 with respect to f'. Consider the following distribution over circuits $\mathcal{P}'$. To sample a circuit C from $\mathcal{P}'$, we start by sampling $15n'$ circuits from $\mathcal{P}$, $C_1, \ldots, C_{15n'}$.

[2]As mentioned in the introduction, Fiat and Naor, required a more subtle analysis to deal with also U-trivial functions.

Then, we let C be the circuit that given y, computes $x_i = C_i(y)$ for every $i \in [5n']$, and checks if, for some i, $f'(x_i) = y$. If there exists some i, C outputs x_i (for the minimal such i), or arbitrary value otherwise. It is not hard to see that C can be implemented with size $\sum_{i \in [15n']} |C_i| + \mathrm{poly}(n) \leq c2^{4n'/5} \cdot n^{c+2}$ using f'-gates. Since the function f' can be implemented by a circuit of size $5n$ with f-gates, and since $n' = n+1$, C can be implemented as a circuit with f-gates, in size at most $10c \cdot 2^{4n/5} \cdot n^{c+3}$.

For the correctness, fix $y \in f([2^n])$, and observe that every pre-image of y with respect to f' is a also a pre-image of y with respect to f. By the promise that a random sample from $\mathcal{P}$ inverts y successfully with probability $1/10$, we get that a random sample C from $\mathcal{P}'$ fails to invert y with probability at most $(9/10)^{15n'} \leq 1/2^{2n'}$. By the union bound, C inverts simultaneously all $y \in f([2^n])$ with probability at least $1 - 2^{-n}$.

Since with positive probability C is a circuit of size $c2^{4n/5} \cdot n^{c+2}$ that inverts f on any image, such a circuit exists. □

3.1 Proving Lemma 3.3

We now move to prove Lemma 3.3. In the following, fix $n \in \mathbb{N}$ and a function $f\colon [2^n] \to [2^n]$ which is not trivial to invert. Let $N = 2^n$.

To prove Lemma 3.3, let m, t, ℓ, k and $U \geq 2^{n/5}$ be parameters to be chosen later. Let $\mathcal{A} = \{y_1, \ldots, y_r\}$ be the set of all images y_i such that $\left|f^{-1}(y_i)\right| \geq U$, and observe that $r := |\mathcal{A}| \leq N/U$. In the Fiat-Naor algorithm, we give as an advice to the inverter a pre-image $x_i \in f^{-1}(y_i)$ of every $y_i \in \mathcal{A}$. As a result, we only left to invert f on $y \notin \mathcal{A}$.[3] Let $\mathcal{D} = \{x\colon f(x) \notin \mathcal{A}\}$. Since f is not U-trivial to invert (recall that $U \geq 2^{n/5}$), it holds that $|\mathcal{D}| \geq N/2$.

Next, we need to invert the function f on the domain $\mathcal{D}$. For the yet to be chosen parameter ℓ, the algorithm of Fiat and Naor [FN00] uses ℓ hash functions $g_1, \ldots, g_\ell\colon [N] \to \mathcal{D}$. We will later describe how to choose this functions, but for now this can be thought as ℓ random functions. Finally, for every $i \in [\ell]$, let $h_i\colon [N] \to \mathcal{D}$ be defined by $h_i(x) = g_i(f(x))$.

In the following algorithm, for every $i \in [\ell]$ and $j \in [m]$, let $x_{i,j} \in [N]$ be inputs to f, to be chosen (randomly) later. For every $i \in [r]$, let x_i be such that $f(x_i) = y_i$, for $\mathcal{A} = \{y_1, \ldots, y_r\}$.

Algorithm 3.4 (Fiat-Naor [FN00]).

```
Parameters: $m, t, \ell, U \in \mathbb{N}$
Advice: $\{(x_i, y_i)\}_{i \in [r]}$, $\{g_i\}_{i \in [\ell]}$, $\{(x_{i,j}, h_i^t(x_{i,j}))\}_{i \in [\ell], j \in [m]}$.
Input: $y \in [N]$.
1. Check if there exists $i \in [r]$ with $y_i = y$. If so, output $x_i$.
2. For every $i \in [\ell]$, set $u_i^0 = g_i(y)$.
3. For every $v \in [t]$:
    (a) For every $i \in [\ell]$, compute $u_i^v = h_i(u_i^{v-1})$.
```

[3]Actually, in [FN00], for $r = \tilde{O}(N/U)$, the points $x_1, \ldots, x_r$ are chosen uniformly at random from the domain $[N]$, and the set $\mathcal{A}$ is defined to be $\{f(x_1), \ldots, f(x_r)\}$. In [FN00] it is shown that with high probability the set $\mathcal{A}$ contains all the images with more than U pre-images, which is the only property of $\mathcal{A}$ that is used in the proof. We could also choose $\mathcal{A}$ in the same way here.

4. For every $i \in [\ell]$, let $\mathcal{J}_i = \{j \colon h_i^t(x_{i,j}) \in \{u_i^0, \ldots, u_i^t\}\}$. Set $\hat{u}_{i,j}^0 = x_{i,j}$ for every $i \in [\ell]$ and $j \in \mathcal{J}_i$,.

5. For every $v \in [t]$:

(a) For every $i \in [\ell]$ and $j \in \mathcal{J}_i$, let $\hat{u}_{i,k}^v = h_i(\hat{u}_{i,k}^{v-1})$.

6. For every $i \in [\ell], v \in [t]$ and $j \in \mathcal{J}_i$, compute $f(\hat{u}_{i,k}^v)$, and output $\hat{u}_{i,k}^v$ if $f(\hat{u}_{i,k}^v) = y$.

That is, for every hash function g_i, the advice contains m points, $x_{i,1}, \ldots, x_{i,m}$. On each such point $x_{i,j}$ we apply the function $h_i = g_i \circ f$ t times in a chain, to get $h_i^t(x_{i,j})$, and keep the result in the advice.

Giving an input y, the algorithm first check if $y \in \mathcal{A}$. If it does, a pre-image of y appears in the advice, and thus y can be trivially inverted. Otherwise, we want to check if $y = f(h_i^v(x_{i,j}))$ for some i, j and $v < t$. That is, if y appears in the chain starting with $x_{i,j}$. If this is the case, then $g_i(y) = h_i^{v+1}(x_{i,j})$, and thus the chain $(g_i(y), h_i(g_i(y)), \ldots, h_i^t(g_i(y)))$ contains the point $h_i^t(x_{i,j})$. The algorithm therefore search for every i, j such that $h_i^t(x_{i,j})$ is in the chain $(g_i(y), h_i(g_i(y)), \ldots, h_i^t(g_i(y)))$. For every such i, j, the algorithm then computes the chain starting with $x_{i,j}$, $(x_{i,j}, \ldots h_i^t(x_{i,j}))$, and checks if one of the element in the latter is indeed a pre-image of y.

Choosing $g_1, \ldots, g_\ell$. We next describe how to chose the functions $g_1, \ldots, g_\ell$. The idea in Fiat and Naor [FN00] is to choose these functions, such that, instead of being ℓ random functions, each function will be $O(t)$-wise independent, and the joint distribution of the functions will be pair-wise independent. This is enough for the analysis of the algorithm, and allows computing the functions efficiently.

One problem we have is that the set $\mathcal{D} \subseteq [N]$ is dependent on the function f, and thus it is not simple to construct a function from $[N]$ to $\mathcal{D}$. To overcame this, Fiat and Naor [FN00] constructed, for each $i \in [\ell]$, a function $g_i' \colon [N] \times [N] \to [N]$. Then, for every $x \in [N]$, they defined $g_i(x)$ to be equal to $g_i'(x, z)$ for the minimal value of $z \in \mathbb{N}$ for which $g_i'(x, z) \in \mathcal{D}$. Notice that given the set $\mathcal{A}$ we can check efficiently if $g_i'(x, z) \in \mathcal{D}$, by checking whether $f(g_i'(x, z)) \in \mathcal{A}$. Moreover, since $|\mathcal{D}| \geq N/2$, it is not hard to see that the expected value of z is $O(1)$.

To construct the functions g_i', we think on the domain $[N] \times [N]$ as $[N^2]$. Let $k \in \mathbb{N}$ be a parameter (we will choose $k = \tilde{O}(t)$), and for every $j \in [k]$, let $a_j, b_j \leftarrow [N^2]$ be random numbers. For every $i \in [\ell]$, let $g_i' \colon [N^2] \to [N]$ be defined by $g_i'(\alpha) = (\sum_{j \in [k]} (a_j \cdot i + b_j) \alpha^{j-1})_{\leq n}$ (where all the operations are preformed over a field of size $N^2 = 2^{2n}$, and we take the first n bits of the result).

By the construction the functions $g_1', \ldots, g_\ell'$ are pair-wise independent, where each of them is k-wise independent. As described above, for every $x \in [N]$, let $g_i(x)$ to be equal to $g_i'(x, z^*)$ for $z^* = \min\{z \colon g_i'(x, z) \in \mathcal{D}\}$ (letting $z^* = N$ if no such exists).

Correctness. By Fiat and Naor [FN00], for a random choice of points $x_{i,j}$ and the polynomials g_i, Algorithm 3.4 inverts f on any fixed y with a good probability. We need a slightly different version of Fiat and Naor [FN00] theorem, stated below.

Theorem 3.5. *Let $m, t, \ell, k, U \in \mathbb{N}$ be parameters such that $m \leq N$, $8 \log(4N) \leq t \leq N$, $k = 8t$, $mk^2U = N$ and $\ell = N/mt$. Let $f \colon [N] \to [N]$ be a function which is not U-trivial to invert. Then, the following holds for every $y \in f([N])$.*

Over a random choice of $\{(a_j, b_j)\}_{j\in[k]}$ *and* $\{(x_{i,j}\}_{i\in[\ell],j\in[m]}$*, Algorithm 3.4 inverts y with probability at least* $1/8$*. Moreover, under the same distribution,* $\mathrm{E}\Big[\sum_{i\in[\ell]}|\mathcal{J}_i|\Big] \leq \ell$*.*

We remark the [FN00] proved the above theorem for $U = N/S$. However, their proof directly generalized to the above set of parameters.

Proof. Theorem 3.5 follows almost directly from the proof in [FN00]. There, to get the best parameters, U is chosen such that $N/U = m \cdot \ell$, and we do not add this constrain. Theorem 3.5 follows by setting $S = N/U$ in the proof of [FN00].

□

3.1.1 Circuit implementation

We now show that, for the right choice of parameters, it is possible to implement Algorithm 3.4 with a relatively small circuit. Recall that we allow f-gates in our circuit.

One main component in the implementation of *Algorithm* 3.4 is the computation of the functions g_i on different inputs. This is done $\Omega(\ell \cdot t)$ times during the algorithm, and thus must be done efficiently. Fiat and Naor [FN00] showed that, due to the dependency between them, the evaluation of the functions $g_1, \ldots, g_\ell$ *simultaneously* on one point each, can be done in $\tilde{O}(\ell)$ time (or, in $\tilde{O}(1)$ amortized time). The following claim shows that (for different parameters) this can be done by a small circuit.

Claim 3.6. *There exists a constant c such that the following holds for every* $n \in \mathbb{N}$ *and* $f\colon [2^n] \to [2^n]$*. Let* $\mathcal{A}, r, k, g'_1, \ldots, g'_\ell$ *and* $g_1, \ldots, g_\ell$ *be as defined above, and let* $\tau, p \in \mathbb{N}$ *be numbers. There exists a C circuit of size* $c \cdot (r + k + \tau \cdot p) \cdot (n + \log(\tau p r))^c$ *such that the following holds. For every* $i \in [\tau]$*, let* $\alpha(i) \in [\ell]$ *and* $u_i \in [N]$ *be such that* $g'_{\alpha(i)}(u_i, z) \in \mathcal{D}$ *for some* $z \leq p$*. Then given* $\alpha(1), \ldots, \alpha(\tau)$ *and* $u_1, \ldots, u_\tau$*, C computes* $g_{\alpha(1)}(u_1), \ldots, g_{\alpha(\tau)}(u_\tau)$

It is worth to mention here that in the setting of [FN00], the main cost in evaluating g_i is the evaluation of the polynomial g'_i. Thus, for [FN00] it was enough to show how to compute $g'_1, \ldots, g'_\ell$ *simultaneously.* While their technique helps us getting better parameters, the crucial cost in the setting here is to check if the output of $g'_i(x, z)$ is in the domain $\mathcal{D}$, or equivalently, if $f(g'_i(x, z)) \in \mathcal{A}$. With RAM machine this can be done efficiently, for example using a binary search, but with a circuit we must a circuit with size linear in the length of $\mathcal{A}$. In the proof we show that this also can be done simultaneously with a smaller amortized cost. An important part of our proof is showing that there is a near linear sized circuit that preform a task for which we call "batch lookup". This task is now defined.

Definition 3.7 (Batch look-up)**.** *Given a list* $\mathcal{A} = (a_1, \ldots, a_r) \in (\{0,1\}^n)^r$ *of length r, and a list* $\mathcal{L} = (y_1, \ldots, y_\tau) \in (\{0,1\}^n)^\tau$ *of* τ *n-bits numbers, the* batch look-up *functionality,* $\mathsf{BLOOKUP}(\mathcal{A}, \mathcal{L})$ *returns* τ *bits* $b_1, \ldots, b_\tau$*, such that* $b_i = 1$ *if* y_i *appears in* $\mathcal{A}$*.*

In Section 3.1.2 we show how to implement the batch look-up functionality with a small circuit. We prove the following lemma.

Lemma 3.8. *There exists a constant c such that the following holds for every* $n \in \mathbb{N}, r \in \mathbb{N}$ *and* $\tau \in \mathbb{N}$*. There exists a circuit C of size* $c \cdot (r + \tau) \cdot (n + \log(r\tau))^c$ *that computes the* $\mathsf{BLOOKUP}$ *functionality.*

Using Lemma 3.8 we prove Claim 3.6.

Proof. In the following we use the notation $\tilde{O}(\cdot)$ to hide fixed multiplicative polynomial factors in n and $\log(p\tau r)$. Without loss of generality, assume that $\tau \cdot p \geq k$. Recall that to compute $g_{\alpha(i)}(u_i)$, we need to find the minimal $z \in \mathbb{N}$ such that $g'_{\alpha(i)}(u_i, z) \in \mathcal{D}$. Moreover, by assumption, $z \in [p]$. Thus we start with computing $g_{\alpha(i)}(u_i, z)$ for every $z \in [p]$ and for every $i \in [\tau]$.

To do so, observe that for the polynomials $A(x) = \sum a_j x^j$, and $B(x) = \sum b_j x^j$ (for $x \in [N^2]$), we can write $g'_i(x) = i \cdot A(x) + B(x)$ for every $i \in [\ell]$. Thus, to compute $g'_{\alpha(i)}(u_i, z)$ for every $i \in [\tau]$, $z \in [p]$, it is enough to compute $A(u_i, z)$ and $B(u_i, z)$ for every such i, z. This can be done with a circuit of size $\tilde{O}(k + \tau \cdot p)$, using Lemma 2.5.

Next, given the values $\left\{g'_{\alpha(i)}(u_i, z)\right\}_{i \in [\tau], z \in [p]}$, we need to find, for every $i \in [\tau]$, the minimal z for which $g'_{\alpha(i)}(u_i, z) \in \mathcal{D}$. This can be done by applying the $\mathsf{BLOOKUP}$ circuit on $\mathcal{A}$ and $\mathcal{L} = (f(g'_{\alpha(1)}(u_1, 1)), \ldots, f(g'_{\alpha(1)}(u_1, p)), \ldots, f(g'_{\alpha(\tau)}(u_\tau, 1)), \ldots, f(g'_{\alpha(\tau)}(u_\tau, p)))$ to compute $(b_{1,1}, \ldots, b_{1,p}, \ldots, b_{\tau,1}, \ldots, b_{\tau,p})$, where $b_{i,z} = 1$ if and only if $f(g'_{\alpha(i)}(u_i, z)) \in \mathcal{A}$ (or, equivalently, if $g'_{\alpha(i)}(u_i, z) \notin \mathcal{D}$). Then, for each block $i \in [\tau]$ of size p, we can output $g'_{\alpha(i)}(u_i, z)$ for the first z in the block for which $b_{i,z} = 0$. Since the last step can be implemented with an efficient one-pass algorithm, and by Lemma 3.8, the entire process can be done with a circuit of size $\tilde{O}(k + r + p \cdot \tau)$. Moreover, the output of the above process is exactly $g_{\alpha(1)}(u_1), \ldots, g_{\alpha(\tau)}(u_\tau)$. □

We next use Claim 3.6 to prove Lemma 3.3.

Proof of Lemma 3.3. In the following we assume that $n > 10$, as we can choose the constant c in Lemma 3.3 such that the theorem will hold trivially for every smaller n. Fix such $n > 10$ and a function f as in Lemma 3.3, and let $N = 2^n$.

We start with our choice of parameters. Let $U = N^{\frac{2}{5}}/64$, $m = N^{\frac{1}{5}}$, $t = N^{\frac{1}{5}}$ and $\ell = N^{\frac{3}{5}}$. It is not hard to verify that the conditions of Theorem 3.5 hold with respect to these parameters, and thus the correctness holds. Namely, for every $y \in f([N])$, over a random value of $a_1, \ldots, a_\ell, b_1, \ldots, b_\ell \leftarrow [N^2]$, and $x_{1,1}, \ldots, x_{\ell,m} \leftarrow [N]$, Algorithm 3.4 finds a pre-image of y with probability at least $1/8$.

To implement Algorithm 3.4 as a circuit, we need a bound some running-time parameters. That is, we need to bound the worst-case number of times we will need to evaluate g'_i for each time Algorithm 3.4 evaluates g_i (so we will be able to use Claim 3.6). Additionally, we will need to bound the size of the sets $\mathcal{J}_i$ defined in Step 4 of the algorithm.

In the following we show that with probability 0.99 over the choice of $a_1, \ldots, a_\ell, b_1, \ldots, b_\ell \leftarrow [N^2]$ it holds that:

1. For every $x \in [N]$ and $i \in [\ell]$, there exists $z \in [3n]$ such that $g'_i(x, z) \in \mathcal{D}$, and,
2. $\sum_{i \in [\ell]} |\mathcal{J}_i| \leq 200\ell$.

Then, we show that when Items 1 and 2 above hold, we can implement Algorithm 3.4 with a small circuit. Overall, by the union bound, this shows that for every $y \in f([N])$, the following holds with probability at least $1/8 - 0.01 \geq 1/10$ over a random value of $a_1, \ldots, a_\ell, b_1, \ldots, b_\ell \leftarrow [N^2]$, and $x_{1,1}, \ldots, x_{\ell,m} \leftarrow [N]$: Algorithm 3.4 can be implemented with a small circuit *and* it finds a pre-image of y. Given this promise, we can easily construct the distribution $\mathcal{P}$: To sample a circuit from $\mathcal{P}$, we randomly choose the parameters $a_1, \ldots, a_\ell, b_1, \ldots, b_\ell \leftarrow [N^2]$, and $x_{1,1}, \ldots, x_{\ell,m} \leftarrow [N]$. Then, if Algorithm 3.4 can be implemented with a circuit of size $\tilde{O}(N^{4/5})$ with the chosen parameters, we output this circuit. Otherwise, we output an arbitrary small circuit.

To see that Items 1 and 2 above holds, first notice that the second item hold with probability $1 - 1/200$ by Markov and the promise that $\mathrm{E}[\sum_i |\mathcal{J}_i|] \leq \ell$. For the first item, randomly choose $a_1, \ldots, a_\ell, b_1, \ldots, b_\ell \leftarrow [N^2]$. We start with showing that $g'_i(x,z) \in \mathcal{D}$ for every i, x and some $z \in [3n]$. Fix i and x and recall that g'_i is a degree $k > 3n$ polynomial. Thus, for every $z \in [3n]$, $\Pr[g_i(x,z) \in \mathcal{D}] = \mathcal{D}/N \geq 1/2$ (where the probability is taken over the choice of $\{a_i, b_i\}$). By the k-wise independence of g'_i, we get that

$$\Pr\big[\forall z \in [3n] \ s.t. \ g'_i(x,z) \notin \mathcal{D}\big] \leq 1/2^{3n} = 1/N^3.$$

By applying the union bound on all possible values of $i \in [\ell]$ and $x \in [N]$, we get that

$$\Pr\big[\exists i \in [\ell], x \in [N], \forall z \in [3n] \ s.t. \ g'_i(x,z) \notin \mathcal{D}\big] \leq \ell \cdot N \cdot 1/N^3 \leq 1/N.$$

In other words, with all but $1/N$ probability, for every i and x, there exists some $z \leq [2n]$ such that $g'_i(x,z) \in \mathcal{D}$, as we wanted to show.

Next, we show how to implement Algorithm 3.4 with a small circuit, under the above assumption. We implement each step of Algorithm 3.4 separately using a small circuit. Recall that $r = N/U = N^{\frac{3}{5}}$. In the following we use the notation $\tilde{O}(\cdot)$ to hide fixed multiplicative polynomial factors in n.

Step 1 can be done trivially by a circuit of size $\tilde{O}(r)$. Similarly, the last step can be done with a circuit of size $\tilde{O}(\ell \cdot t)$ under the assumption in Item 2, and using the f-gates.

By Claim 3.6, Step 2 can be implemented by a circuit of size $\tilde{O}(r + k + \ell)$. Similarly, as a corollary of Claim 3.6, Steps 3 and 5 (under the assumption that Item 2 holds) can be done with a circuit of size $\tilde{O}(t \cdot (r + k + \ell))$.

Lastly, Step 4 can be done with circuit of size $\tilde{O}(\ell \cdot t + \ell \cdot m)$, by using batch look-up again. In more detail, let $\mathcal{A} = (u_1^0, \ldots, u_1^t, \ldots, u_\ell^0, \ldots, u_\ell^t)$ and $\mathcal{L} = (h_1^t(x_{1,1}), \ldots, h_\ell^t(x_{\ell,m}))$. We first compute $\mathsf{BLOOKUP}(\mathcal{A}, \mathcal{L})$ to get $\mathcal{R} = (b_{1,1}, \ldots, b_{\ell,m})$. Then, we can easily change the list $\mathcal{R}$ such that it will contain the block index and the input $x_{i,j}$ in every coordinate, that is $\mathcal{R}' = ((b_{1,1}, x_{1,1}, 1), \ldots, (b_{\ell,m}, x_{\ell,m}, \ell))$. We can now sort the list according to the first entry $b_{i,j}$, and keep only the last 100ℓ entries. By the assumption in Item 1, those entries contains all the entries i, j for which $h_i^t(x_{i,j}) \in \{u_i^0, \ldots, u_i^t\}$. Each element in the output list contains both $x_{i,j}$ and i, which is enough to apply Claim 3.6 in Step 5.

Overall, the entire process can be implemented by a circuit of size $\tilde{O}(t(r+k+\ell)+\ell\cdot m) = \tilde{O}(N^{4/5})$, as stated.

□

3.1.2 Implementing Batch Look-Up

We now prove Lemma 3.8. The main idea is to sort the lists $\mathcal{A}, \mathcal{L}$ together, and then find all the duplications in the sorted list.

Proof. Let $\tau, r, n, \mathcal{A} = (a_1, \ldots, a_r)$ and $\mathcal{L} = (y_1, \ldots, y_\tau)$ be as in Definition 3.7. Let $k = \lceil \log \tau \rceil$ To compute $\mathsf{BLOOKUP}(\mathcal{A}, \mathcal{L})$ we do the following:

1. First, we construct the lists $\mathcal{A}' = ((a_1, 0^k), \ldots, (a_r, 0^k)) \in (\{0,1\}^n \times \{0,1\}^k)^r$ and $\mathcal{L}' = ((y_1, 1), \ldots, (y_\tau, \tau)) \in (\{0,1\}^n \times \{0,1\}^k)^\tau$. Namely, we add the index i (represented as a k-bits string) to every element in $\mathcal{L}$, and 0 for every element in $\mathcal{A}$. This can be done with a circuit of size $(r + \tau)(n + k)$.

2. Next, we sort $(\mathcal{A}', \mathcal{L}')$ jointly, according to the first and then the second entry. Let $\mathcal{R}_1$ be the output. This can be done with a circuit of size $c_1 \cdot (r+\tau)(n+k+\log(r+\tau))^{c_1}$, by Lemma 2.6, for some universal constant c_1. Let $\mathcal{R}_1$ be the output of this step.

3. In the next step, we replace with 1 the first entry of every element (y_i, i) in $\mathcal{R}_1$, such that an element of the form $(y_i, 0)$ appears earlier in the list $\mathcal{R}_1$. We replace by 0 the first entry of every element (y_i, i) in $\mathcal{R}_1$, such that $(y_i, 0)$ does not appear earlier in the list. Let $\mathcal{R}_2$ be the output of this step. The result of this step is that $\mathcal{R}_2$ contains entries of the form (b_i, i), where b_i is 1 if and only if y_i appears in $\mathcal{A}$. As this step can be implemented by an efficient, one-pass algorithm (the algorithm only needs to remember the first entry of the last element with second entry equal to 0 as its state), it can be also be implemented by circuit of size $c_2 \cdot (r+\tau)(n+k+\log(r+\tau))^{c_2}$ by Lemma 2.7, for some universal constant c_2.

4. Lastly, we sort the list again according to the second entry, to get the elements in order according to the indexes. We output the first entry of each of the last τ items in the sorted list. It is not hard to see that the values we output are exactly the one we wanted to compute. Moreover, this last step can also be implemented by a circuit of size $c_1 \cdot (r+\tau)(n+k+\log(r+\tau))^{c_1}$.

By concatenating the circuits for each of the above steps, we get a circuit of size $c \cdot (r+\tau)(n + \log r\tau)^c$ that computes the functionality $\mathsf{BLOOKUP}$, for a sufficiently large constant c. $\square$

4 Computing t-Bounded Kolmogorov Complexity

In this part we use Theorem 3.1 to prove our main theorem. The latter is a corollary of the following theorem.

Theorem 4.1. *For any universal TM U and every function $t = t(n)$ there exists a circuit family $\mathcal{C} = \{C_n\}_{n\in\mathbb{N}}$ of size $O(2^{4n/5} \cdot \mathrm{poly}(n))$ such that the following holds for every $n \in \mathbb{N}$. C_n is a circuit with f_n-gates, for $f_n \colon \{0,1\}^{\leq 2n} \to \{0,1\}^*$ defined by $f_n(\Pi) = \mathsf{U}(\Pi, 1^{t(n)})$, and for every $x \in \{0,1\}^n$, $C_n(x)$ outputs $\mathrm{K}^t_{\mathsf{U}}(x)$.*

Recall that we defined $\mathrm{K}^t = \mathrm{K}^t_{\mathsf{U}}$ for some fixed U with polynomial running time overhead. We get the following corollary.

Corollary 4.2 (Main theorem)**.** *For every $t = t(n)$, there exists a circuit family $\mathcal{C} = \{C_n\}_{n\in\mathbb{N}}$ of size $O(2^{4n/5} \cdot \mathrm{poly}(t(n), n))$ (over the DeMorgan basis) such that, for every $n \in \mathbb{N}$ and for every $x \in \{0,1\}^n$, $C_n(x)$ outputs $\mathrm{K}^t(x)$.*

Proof of Corollary 4.2. Corollary 4.2 follows by Theorem 4.1, observing that we can replace every f_n-gate with a circuit of size $\mathrm{poly}(t(n), n)$ that computes f_n. We get a circuit family of size $O(2^{4n/5} \cdot \mathrm{poly}(n, t(n)))$ that computes K^t. $\square$

We now prove Theorem 4.1

Proof of Theorem 4.1. Let c be a constant such that $\mathrm{K}^t(x) \leq |x| + c$ for every x. Let ${f'}_n \colon \{0,1\}^{n+c} \times [n+c] \to \{0,1\}^n \times [n+c]$ be defined as

$$
{f'}_n(\Pi, i) = \begin{cases} (f_n(\Pi_{\leq i}), i) & |f_n(\Pi_{\leq i})| = n \\ 0^n & \text{Otherwise} \end{cases}
$$

Let $n' = n + c + \lceil \log(n+c) \rceil$. In the following, we assume that both the domain and the range of f_n is $[2^{n'}]$, by the use of appropriate encoding and padding.

By Theorem 3.1, the above imply that there is a circuit family $\widehat{\mathcal{C}} = \left\{ \widehat{C}_n \right\}_{n \in \mathbb{N}}$ with ${f'}_n$ gates, of size $O(2^{4n'/5} \cdot \mathrm{poly}(n)) = O(2^{4n/5} \cdot \mathrm{poly}(n))$ that inverts ${f'}_n$ with probability 1.

Given a circuit $\widehat{C}_n$ that inverts ${f'}_n$, we can construct a circuit C_n (with ${f'}_n$ gates) that computes the K^t complexity of any string x of length n. This can be done by computing ${f'}_n^{-1}(x, 1), \ldots, {f'}_n^{-1}(x, n + c)$ and taking the output (Π, i) for the minimal value of i such that $U(\Pi_{<i}, 1^{t(n)}) = x$ (the t-bounded Kolmogorov complexity of the string 0^n can be hardcoded in the circuit).

Observe that the size of C_n is $n \cdot \left| \widehat{C}_n \right| + \mathrm{poly}(n)$. Thus, there exists a circuit family of size $O(2^{4n/5} \cdot \mathrm{poly}(n))$, with ${f'}_n$ gates, that computes K^t.

Lastly, observe that f'_n can be efficiently computed from f_n, thus we can replace the f'_n gates with a small circuit. □

5 Lower Bound on the Circuit Size

The result presented in Section 4 holds when K^t is defined with respect to any universal TM with polynomial running-time overhead (more specifically, for every universal TM that can be implemented as a circuit with polynomial size in the running time $t(n)$ and the program Π). Moreover, by Theorem 4.1, the result generalized for any universal TM U, if we allow the circuit to have oracle gates to the UTM U.

In this section we prove a lower bound on this type of circuits. Namely, we show that there is a "black-box" universal TM, with respect to there is no circuit of sub-exponential size that computes K^t using oracle gates to U. In more details, we consider the black-box universal TM definition from [LP23].

Definition 5.1 (Black-box universal TM). *A function* $\mathsf{U} : \{0,1\}^* \times 1^* \to \{0,1\}^* \cup \{\perp\}$, *we say that* U *is a* black-box universal Turing machine (black-box UTM) *if*

- *(Universality) Informally, this requires that* U *simulates "valid programs" correctly: There exists a standard universal Turing machine* U_0 *such that for any* $(M, 1^t)$*, if* M *is a valid description of a Turing machine (w.r.t* U_0*),* $\mathsf{U}(M, 1^t)$ *outputs what* M *outputs after* t *steps.*
- *(Any "program" has a unique output) For any* $M \in \{0,1\}^*, t_1, t_2 \in \mathbb{N}, t_1 \leq t_2$*, if* $\mathsf{U}(M, 1^{t_1}) \neq \perp$*,* $\mathsf{U}(M, 1^{t_2}) = \mathsf{U}(M, 1^{t_1})$.

We remark that the above definition is black-box in the following two ways: (1) U is defined to be a function; (2) U is allowed to assign the output of invalid "programs" with an arbitrary string.

For any black-box UTM U, we can define the time-bounded Kolmogorov complexity with respect to U. Formally, for any black-box universal Turing machine U, any string $x \in \{0,1\}^*$, let

$$\mathrm{K}^t_{\mathsf{U}}(x) = \min_{\Pi \in \{0,1\}^*} \{ |\Pi| \mid \mathsf{U}(\Pi, 1^{t(n)}) = x \}.$$

In the following, for a universal TM U, a function $t \colon \mathbb{N} \to \mathbb{N}$, and a number $n \in \mathbb{N}$, we let f_{U} be the function defined by $f^{\mathsf{U}}_n(\Pi) = \mathsf{U}(\Pi, 1^{t(n)})$ for any $\Pi \in \{0,1\}^{\leq 2n}$. Using appropriate encoding, we can see f^{U}_n as a function from $n + \log n$ bits to $2t(n)$ bits.

A circuit C with oracle gates is a circuit C with special $\mathcal{O}$-gates. For a function f (with the same input and output length), we let C^f be the circuit received by replacing the $\mathcal{O}$-gates with f-gates.

We next define black-box K^t-solvers. We give two definitions, of fully black-box and (plain) black-box. The first is now defined.

Definition 5.2 (Fully black-box K^t-solver)**.** *A circuit family $\mathcal{C} = \{C_n\}_{n\in\mathbb{N}}$ is* fully black-box K^t-solver *if for every black-box universal TM U, for every $n \in \mathbb{N}$ and for every input $x \in \{0,1\}^n$, $C_n^{f_n^{\mathsf{U}}}(x) = \mathrm{K}^t_{\mathsf{U}}$.*

That is, the circuit family $\mathcal{C}$ can be used to compute the $\mathrm{K}^t_{\mathsf{U}}$ complexity with respect to any universal TM U. Our construction in Section 4 is not fully black-box. Indeed, the circuit family we construct is dependent in the universal TM with respect to we defined K^t. We next define a weaker notion of black-box solutions, that captures these kinds of constructions.

Definition 5.3 (Black-box K^t-solver)**.** *For any black-box universal TM U, a circuit family $\mathcal{C} = \{C_n\}_{n\in\mathbb{N}}$ is* black-box $\mathrm{K}^t_{\mathsf{U}}$-solver *if for every $n \in \mathbb{N}$, C_n is a circuit with f_n^{U}-gates, and for every input $x \in \{0,1\}^n$, $C_n^{f_n^{\mathsf{U}}}(x) = \mathrm{K}^t_{\mathsf{U}}$.*

Here we only require the circuit to compute $\mathrm{K}^t_{\mathsf{U}}$ with respect to U, and allow the circuit to use black-box access to the universal TM. Theorem 4.1 directly implies the following theorem.

Theorem 5.4. *For any universal TM U and every function $t = t(n)$ there exists a black-box $\mathrm{K}^t_{\mathsf{U}}$ solver $\mathcal{C} = \{C_n\}_{n\in\mathbb{N}}$ of size $O(2^{4n/5} \cdot \mathrm{poly}(n))$.*

In this part we prove the following lower bound on the size of black-box K^t solvers. This lower bound immediately generalized to fully black-box solvers.

Theorem 5.5. *There exists a black-box UTM $\widehat{\mathsf{U}}$ and a constant c such that the following holds for every function $t(n) \in \mathrm{poly}$ with $t(n) \geq n^c$. For any black-box $\mathrm{K}^t_{\widehat{\mathsf{U}}}$-solver $\mathcal{C} = \{C_n\}_{n\in\mathbb{N}}$ it holds that $|C_n| \geq 2^{n/2-o(n)}$ for infinitely many $n \in \mathbb{N}$.*

We leave open the question of closing the gap between the lower and upper bounds on the size of black-box K^t-solvers.

In the following we give two proofs for the above theorem. In both proofs, we use an universal TM U_{oracle} with an oracle as our black-box UTM. In the first proof, we we give U_{oracle} an oracle to a random function, used as a pseudorandom generator (PRG). We then use the K^t-solver to break the PRG. We use the result of Coretti, Dodis, Guo, and Steinberger [CDGS18] which states that a random function is a good PRG against circuits of size roughly $2^{n/2}$.

The second proof get slightly worse parameters, but is based on more standard results. Here we take the oracle to be a random permutation σ, and use a result from [Imp11] that states of that a random permutation cannot be inverted by a circuit of size $2^{n/2}$. Then, we use Goldreich-Levin hardcore functions to construct a PRG from σ.

The first proof. For our first proof, we will use the following theorem from [CDGS18].

Theorem 5.6 ([CDGS18])**.** *There exists a constant c such that the following holds for every numbers $m > n$. Let $(\mathsf{A}_0, \mathsf{A}_1)$ be a pair of oracle-aided algorithms, such that given a oracle*

$\mathcal{O}\colon \{0,1\}^n \to \{0,1\}^m$, $\mathsf{A}_0^{\mathcal{O}}$ outputs advice a of length $S > n$, and A_1 makes at most T queries to $\mathcal{O}$. Then

$$\left|\Pr_{\mathcal{O}\leftarrow\mathcal{F}_{n,m},x\leftarrow\{0,1\}^n}\left[\mathsf{A}_1^{\mathcal{O}}(\mathsf{A}_0^{\mathcal{O}},\mathcal{O}(x))=1\right]-\Pr_{\mathcal{O}\leftarrow\mathcal{F}_{n,m},y\leftarrow\{0,1\}^m}\left[\mathsf{A}_1^{\mathcal{O}}(\mathsf{A}_0^{\mathcal{O}},y)=1\right]\right|\leq cm(\sqrt{\frac{ST}{2^n}}+\frac{Tn}{2^n})$$

where $\mathcal{F}_{n,m}$ is the family of all functions from $\{0,1\}^n$ to $\{0,1\}^m$.

That is, a random oracle is a good PRG, even against adversaries with non-uniform advice. The following corollary, stating that there exists a function which is a good PRG against small circuits, follows almost directly from Theorem 5.6.

Corollary 5.7. *There exists a constant c such that for every number $n \in \mathbb{N}$ and for $m = n + \log n$, there exists a function $G\colon \{0,1\}^n \to \{0,1\}^m$, such that the following holds for every circuit of size $2^{n/2-2\log^2 n}$ with G-gates.*

$$\left|\Pr_{x\leftarrow\{0,1\}^n}[C(G(x))=1]-\Pr_{y\leftarrow\{0,1\}^m}[C(y)=1]\right|\leq 1/10.$$

Proof of Corollary 5.7. Fix large enough $n \in \mathbb{N}$, and assume toward a contradiction that for every function $G\colon \{0,1\}^n \to \{0,1\}^m$, there exists a circuit C with $\mathcal{O}\colon \{0,1\}^n \to \{0,1\}^m$ oracle gates of size $2^{n/2-2\log^2 n}$, such that

$$\left|\Pr_{x\leftarrow\{0,1\}^n}[C(G(x))=1]-\Pr_{y\leftarrow\{0,1\}^m}[C(y)=1]\right|> 1/10.$$

By flipping the answer of C, we can assume that

$$\Pr_{x\leftarrow\{0,1\}^n}[C(G(x))=1]-\Pr_{y\leftarrow\{0,1\}^m}[C(y)=1]> 1/10.$$

First, observe that the circuit C can be described with less than $S := 3\cdot|C|\cdot n^2 \leq 2^{n/2+3\log n-2\log^2 n}$ bits (see for example [GGKT05]).

Next, consider the oracle-aided pair of algorithms $(\mathsf{A}_0, \mathsf{A}_1)$, that given access to an oracle G, A_0^G outputs a description of length $2^{n/2+3\log n-2\log^2 n}$ of the circuit C that distinguish $G(x)$ from uniform. Given the advice, and an input z, A_1^G simulates $C^G(z)$. Clearly A_1 makes at most $T = |C|$ queries to G. By our assumption, for every G,

$$\Pr_{x\leftarrow\{0,1\}^n}\left[\mathsf{A}_1^{G}(\mathsf{A}_0^{G},G(x))=1\right]-\Pr_{y\leftarrow\{0,1\}^m}\left[\mathsf{A}_1^{G}(\mathsf{A}_0^{G},y)=1\right]>1/10.$$

By taking expectation on $G \leftarrow \mathcal{F}_{n,m}$, and by Theorem 5.6, it must holds that

$$1/10\leq cm(\sqrt{\frac{ST}{2^n}}+\frac{Tn}{2^n})\leq 2cm\sqrt{\frac{ST}{2^n}}\leq 4cn\sqrt{2^{3\log n-4\log^2 n}},$$

which is a contradiction for large enough n. □

We can now prove Theorem 5.5.

Proof of Theorem 5.5. Let $\mathcal{N} = \{n_1, n_2, \dots\} \subseteq \mathbb{N}$ be the infinite set defined by $n_1 = 1$ and $n_{i+1} = 2^{n_i}$. We will show a black-box UTM, with respect to, $|C_{2n+\log n}| \geq 2^{n/4-o(n)}$ for every $n \in \mathcal{N}$, which concludes the theorem.

The black-box UTM we consider is an oracle universal Turing machine U_{oracle} such that for any oracle $\mathcal{O}$, $\mathsf{U}_{oracle}^{\mathcal{O}}$ simulates any oracle machine Π with the oracle $\mathcal{O}$. In addition, if Π does not make oracle query, U_{oracle} will also simulate the execution of Π. We next describe our choice of the oracle $\mathcal{O}$: For every large enough $n \in \mathcal{N}$, let $G_n \colon \{0,1\}^n \to \{0,1\}^{n+\log n}$ be the function promised by Corollary 5.7 (for $m = n + \log n$). For every $x \in \{0,1\}^n$ with $n \in \mathcal{N}$, let $\mathcal{O}(x) = G_n(x)$. For every $x \in \{0,1\}^n$ with $n \notin \mathcal{N}$, let $\mathcal{O}(x) = x$. Let $\widehat{\mathsf{U}} = \mathsf{U}_{oracle}^{\mathcal{O}}$. The choice of the sparse set $\mathcal{N}$ is to make sure that on input of length n, it is not useful to $\widehat{\mathsf{U}}$ to call the oracle $\mathcal{O}$ on inputs with length different than n (see more detail below). This is a similar technique to the one used in the sparsification lemma in [CLMP12].

Intuitively, we use the oracle $\mathcal{O}$ as a PRG. Observe that for every $n \in \mathcal{N}$, G can be computed in polynomial time using the oracle to $\mathcal{O}$. Let M_G be such an efficient oracle-aided TM that computes G, and let c be a constant such that n^c is a upper bound on the running time of M_G. Fix a function $t(n)$ such that $t(n) \geq n^c$ for every $n \in \mathbb{N}$. Clearly, $\mathrm{K}^t_{\widehat{\mathsf{U}}}(G(x)) \leq n + 2|M_G| \leq n + 0.1 \log n$, for every $x \in \{0,1\}^n$ and every large enough n.

On the other hand, for random $Y \leftarrow \{0,1\}^m$, $\mathrm{K}^t_{\widehat{\mathsf{U}}}(Y) \geq n + \log n - 1$ with probability at least $1/2$. Thus, every circuit C that computes $\mathrm{K}^t_{\widehat{\mathsf{U}}}$ on strings of length $n + \log n$ can be used to distinguish between $G(X)$ and Y, for $X \leftarrow \{0,1\}^n$, with advantage $1/2$ by outputting 1 if the output of C is larger or equal to $n + \log n - 1$. By the choice of G_n, it follows that if C is a circuit with G_n-gates, it must holds that $|C| \geq 2^{n/2 - 2\log^2 n}/n^c$.

Recall that the circuits in $\mathcal{C} = \{C_n\}_{n \in \mathbb{N}}$ have oracle gates to $f_n := f_n^{\widehat{\mathsf{U}}}$, and our lower bound above only holds for circuits with G_n gates. To conclude the theorem, we thus need to show that f_n can be implemented with a circuit of size $\mathrm{poly}(n)$ with G_n-gates. Indeed, in this case, taking $C = C_{n+\log n}$, we get that $|C_{n+\log n}| \geq 2^{n/2 - 2\log^2 n}/\mathrm{poly}(n)$.

To see that we can implement f_n with circuit of polynomial size (with G_n-gates) first observe that if we allow $\mathcal{O}$-gates, we can simulate f_n with a circuit of size $\mathrm{poly}(t(n))$. Thus, we only left to implement the oracle $\mathcal{O}$.

To do so, recall that on inputs x of length n, $\mathcal{O}(x) = G_n(x)$. Therefore we only need to simulate $\mathcal{O}$ on other input lengths. For every $n' > n$ with $n' \in \mathcal{N}$, it holds that $n' \geq 2^n$. Since $t(n) \leq 2^n$, we get that for every call to f_n, $\widehat{\mathsf{U}}$ cannot call to $\mathcal{O}$ on inputs length $n' \in \mathcal{N}$ with $n' > n$. Thus, we do not need to simulate $\mathcal{O}$ on those inputs. Moreover, for every $n' < n$ with $n' \in \mathcal{N}$, it holds that $n' \leq \log n$. Thus, $\mathcal{O}(x)$ for any $x \in \{0,1\}^{n'}$ can be computed by a circuit of size $O(2^{n'}) = O(n)$. Lastly, for every x with $|x| \notin \mathcal{N}$, it holds that $\mathcal{O}(x) = x$. Thus the oracle can be implemented trivially also in this case. □

The second proof. For our second proof of Theorem 5.5, we will use the following two theorems. The first state that for every large enough n, there exists a permutation σ which is hard to invert by any circuit of size $2^{n/2}$, even using σ gates.

Theorem 5.8 (Hard to invert permutation)**.** *For every large enough $n \in \mathbb{N}$, there exists a permutation $\sigma \colon \{0,1\}^n \to \{0,1\}^n$ such that the following holds for every circuit C of size at most $2^{n/2 - 2\log^2 n}$ with σ-gates.*

$$\Pr_{x \leftarrow \{0,1\}^n}[C(\sigma(x)) = x] \leq n^{-\log n}.$$

Theorem 5.8 follows directly from [Imp11], with the same lines of the proof of Corollary 5.7. We give the proof of Theorem 5.8 in Section 5.1.

We will also use the following theorem, which is similar to the Goldriech-Levin hardcore predicate. This version allow to output $\log n$ hardcore bits using a seed of length n. We will need a weak version of their result with a stronger assumption.

Theorem 5.9 (Goldreich-Levin hardcore functions [GL89; Gol01])**.** *There exists an efficiently computable function $g\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^{\log n}$, an efficient oracle aided algorithm R, and a constant $c \in \mathbb{N}$ such that the following holds. Let $f\colon \{0,1\}^n \to \{0,1\}^n$ be a function, and A an algorithm, such that*

$$\left|\Pr_{x\leftarrow\{0,1\}^n, s\leftarrow\{0,1\}^n}[\mathsf{A}(s, f(x), g(x,s)) = 1] - \Pr_{x\leftarrow\{0,1\}^n, s\leftarrow\{0,1\}^n, z\leftarrow\{0,1\}^{\log n}}[\mathsf{A}(s, f(x), z) = 1]\right| \geq 0.1.$$

Then,

$$\Pr_{x\leftarrow\{0,1\}^n}\left[\mathsf{R}^{\mathsf{A}}(f(x)) = x\right] \geq n^{-c}.$$

Moreover, R makes at most n^c oracle calls to A.

In the following we prove a slightly weaker lower bound of $2^{n/4-o(n)}$ on the size of black-box K^t-solvers.

Proof of Theorem 5.5, second proof. As in the first proof above, let $\mathcal{N} = \{n_1, n_2, \dots\} \subseteq \mathbb{N}$ be the infinite set defined by $n_1 = 1$ and $n_{i+1} = 2^{n_i}$. We will show a black-box UTM, with respect to, $|C_{2n+\log n}| \geq 2^{n/4-o(n)}$ for every $n \in \mathcal{N}$, which concludes the theorem.

We again consider oracle universal Turing machine U_{oracle} as our black-box UTM. We next describe our choice of the oracle $\mathcal{O}$: For every large enough $n \in \mathcal{N}$, let σ_n be the permutation promised by Theorem 5.8. For every $x \in \{0,1\}^n$ with $n \in \mathcal{N}$, let $\mathcal{O}(x) = \sigma_n(x)$. For every $x \in \{0,1\}^n$ with $n \notin \mathcal{N}$, let $\mathcal{O}(x) = x$. Let $\widehat{\mathsf{U}} = \mathsf{U}_{oracle}^{\mathcal{O}}$.

Intuitively, we use the oracle $\mathcal{O}$ to construct a pseudorandom generator G. In more detail, fix $n \in \mathcal{N}$ and let $G\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^{2n+\log n}$ be the function defined by

$$G(x,s) = (s, \sigma_n(x), g(x,s)),$$

where g is the function promised by Theorem 5.9. The proof now proceeds similarly to the first proof. Let M_G be an efficient oracle-aided TM that computes G, and let c be a constant such that n^c is a upper bound on the running time of M_G. Fix a function $t(n)$ such that $t(n) \geq n^c$ for every $n \in \mathbb{N}$. Clearly, $\mathrm{K}^t_{\widehat{\mathsf{U}}}(G(x,s)) \leq 2n + 2|M_G| \leq 2n + 0.1\log n$, for every $x, s \in \{0,1\}^n$ and every large enough n.

On the other hand, for random $X \leftarrow \{0,1\}^n, S \leftarrow \{0,1\}^n$ and $Z \leftarrow \{0,1\}^{\log n}$, $\mathrm{K}^t_{\widehat{\mathsf{U}}}(S, \pi(X), Z) \geq 2n + \log n - 1$ with probability at least $1/2$. Thus, every circuit C that computes $k^t_{\widehat{\mathsf{U}}}$ on strings of length $2n + \log n$ can be used to distinguish between $(S, \sigma_n(X), g(X,S))$ and $(S, \sigma_n(X), g(X,Z))$ with advantage $1/2$, by outputting 1 if the output of C is larger or equal to $2n+\log n-1$. It follows that, by Theorem 5.9, there exists a n^c-size circuit C' that, given oracle gates to C, inverts σ_n with probability at least n^{-c}. By the choice of σ_n, it follows that if C is a circuit with σ_n-gates, it must holds that $|C| \geq 2^{n/2-2\log^2 n}/n^c$.

To conclude the theorem, we need to show that f_n can be implemented with a circuit of size $\mathrm{poly}(n)$ with σ_n-gates. As in the previous proof, if we allow $\mathcal{O}$-gates, we can simulate f_n with a circuit of size $\mathrm{poly}(t(n))$. Moreover, by over choice of $\mathcal{N}$, we can simulate the oracle $\mathcal{O}$ on every input length given an oracle to σ_n. This is done using the same method used in the first proof above. □

We believe that we can strengthen Theorem 5.5 to hold for every input length. In the above proof we use a similar technique to the one used in the sparsification lemma in [CLMP12], to make sure the circuit only use the oracle on inputs with length n. However, we believe that by following the proof of Theorem 5.8, it is possible to show similar lower bound even when there is oracle access to all the permutations $\{\sigma_n\}_{n\in\mathbb{N}}$. We left the weaker statement to keep the proof modular.

5.1 Proving Theorem 5.8

In this part we prove Theorem 5.8. We will use the following theorem from [Imp11].

Theorem 5.10. *There exists a constant c such that the following holds. Let $n \in \mathbb{N}$ and $1 > \delta > 0$ be numbers, and let $\mathcal{A}$ be a T-query oracle-aided algorithm, with advice of length S. Then with probability $1-\delta$ over the choice of a random permutation $\sigma\colon \{0,1\}^n \to \{0,1\}^n$, the following holds for any advice $w \in \{0,1\}^S$.*

$$\Pr_{x\leftarrow\{0,1\}^n}[\mathcal{A}^{\sigma}(\sigma(x), w) = x] \leq c\frac{T(S+\log 1/\delta)}{2^n}$$

Proof of Theorem 5.8. Fix large enough $n \in \mathbb{N}$, and assume toward a contradiction that for every permutation $\sigma\colon \{0,1\}^n \to \{0,1\}^n$, there exists a circuit C with $\mathcal{O}\colon \{0,1\}^n \to \{0,1\}^n$ oracle gates of size $2^{n/2-2\log^2 n}$, such that $\Pr_{x\leftarrow\{0,1\}^n}[C^{\sigma}(\sigma(x)) = x] \geq n^{-\log n}$. First, observe that the circuit C can be described with less than $S := 3\cdot|C|\cdot n^2 \leq 2^{n/2+3\log n-2\log^2 n}$ bits (see for example [GGKT05]).

Next, consider the oracle-aided algorithm A, that given access to an oracle σ, advice $w \in \{0,1\}^S$ and input y, A interpret w as a description of a circuit C of size $2^{n/2-2\log^2 n}$, and simulate $C^{\sigma}(y)$.

Clearly A makes at most $|C|$ queries to σ. By our assumption, for every σ there exists an advice w such that $\Pr_{x\leftarrow\{0,1\}^n}[\mathsf{A}^{\sigma}(w, \sigma(x)) = x] \geq n^{-\log n}$. By Theorem 5.10, setting $\delta = 1/2$, we get that

$$n^{-\log n} \leq c\frac{|C|(S+1)}{2^n} \leq c\frac{2^{n/2-2\log^2 n}(2^{n/2+3\log n-2\log^2 n}+1)}{2^n}$$

which is a contradiction for large enough n. □

References

[AKS83] Miklós Ajtai, János Komlós, and Endre Szemerédi. “An 0 (n log n) sorting network”. In: *Proceedings of the fifteenth annual ACM symposium on Theory of computing.* 1983, pp. 1–9 (cit. on p. 8).

[Bav12] Mohammad Bavarian. “Lecture 6”. In: *Lecture notes in ”6.S897: Algebra and Computation” by Madhu Sudan* (2012) (cit. on p. 8).

[CDGS18] Sandro Coretti, Yevgeniy Dodis, Siyao Guo, and John Steinberger. “Random oracles and non-uniformity”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques.* Springer. 2018, pp. 227–258 (cit. on p. 17).

[Cha69] Gregory J. Chaitin. “On the Simplicity and Speed of Programs for Computing Infinite Sets of Natural Numbers”. In: *J. ACM* 16.3 (1969), pp. 407–422 (cit. on p. 2).

[CLMP12] Kai-Min Chung, Edward Lui, Mohammad Mahmoody, and Rafael Pass. "Unprovable Security of 2-Message Zero Knowledge". In: *Cryptology ePrint Archive* (2012) (cit. on pp. 19, 21).

[DH76] Whitfield Diffie and Martin E. Hellman. "New Directions in Cryptography". In: *IEEE Transactions on Information Theory* (1976), pp. 644–654 (cit. on p. 2).

[FN00] Amos Fiat and Moni Naor. "Rigorous time/space trade-offs for inverting functions". In: *SIAM Journal on Computing* 29.3 (2000), pp. 790–803 (cit. on pp. 2–6, 8–12, 23, 24).

[GGKT05] Rosario Gennaro, Yael Gertner, Jonathan Katz, and Luca Trevisan. "Bounds on the Efficiency of Generic Cryptographic Constructions". In: *SIAM Journal on Computing* 35.1 (2005), pp. 217–246 (cit. on pp. 18, 21).

[GL89] Oded Goldreich and Leonid A. Levin. "A Hard-Core Predicate for all One-Way Functions". In: *Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC).* 1989, pp. 25–32 (cit. on p. 20).

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Tools.* Cambridge University Press, 2001 (cit. on p. 20).

[Har83] J. Hartmanis. "Generalized Kolmogorov complexity and the structure of feasible computations". In: *24th Annual Symposium on Foundations of Computer Science (sfcs 1983).* 1983, pp. 439–445. DOI: `10.1109/SFCS.1983.21` (cit. on p. 2).

[Hel80] Martin Hellman. "A cryptanalytic time-memory trade-off". In: *IEEE transactions on Information Theory* 26.4 (1980), pp. 401–406 (cit. on pp. 2, 3).

[Imp11] Russell Impagliazzo. "Relativized separations of worst-case and average-case complexities for NP". In: *2011 IEEE 26th Annual Conference on Computational Complexity.* IEEE. 2011, pp. 104–114 (cit. on pp. 17, 19, 21).

[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.* 2000, pp. 73–79 (cit. on p. 2).

[Ko86] Ker-I Ko. "On the Notion of Infinite Pseudorandom Sequences". In: *Theor. Comput. Sci.* 48.3 (1986), pp. 9–33. DOI: `10.1016/0304-3975(86)90081-2`. URL: `https://doi.org/10.1016/0304-3975(86)90081-2` (cit. on p. 2).

[Kol68] A. N. Kolmogorov. "Three approaches to the quantitative definition of information". In: *International Journal of Computer Mathematics* 2.1-4 (1968), pp. 157–168 (cit. on p. 2).

[LP20] Yanyi Liu and Rafael Pass. "On one-way functions and Kolmogorov complexity". In: *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS).* IEEE. 2020, pp. 1243–1254 (cit. on pp. 2, 3).

[LP23] Yanyi Liu and Rafael Pass. "On One-way Functions and the Worst-case Hardness of Time-Bounded Kolmogorov Complexity". In: *Cryptology ePrint Archive* (2023) (cit. on p. 16).

[RS21] Hanlin Ren and Rahul Santhanam. "Hardness of KT Characterizes Parallel Cryptography". In: *36th Computational Complexity Conference (CCC 2021).* Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2021 (cit. on p. 2).

[Sip83] Michael Sipser. “A Complexity Theoretic Approach to Randomness”. In: 1983, pp. 330–335 (cit. on p. 2).

[Sol64] R.J. Solomonoff. “A formal theory of inductive inference. Part I”. In: *Information and Control* 7.1 (1964), pp. 1 –22. ISSN: 0019-9958. DOI: `https://doi.org/10.1016/S0019-9958(64)90223-2` (cit. on p. 2).

[Tra84] Boris A Trakhtenbrot. “A survey of Russian approaches to perebor (brute-force searches) algorithms”. In: *Annals of the History of Computing* 6.4 (1984), pp. 384–400 (cit. on p. 2).

[Yab59a] Sergey Yablonski. “The algorithmic difficulties of synthesizing minimal switching circuits”. In: *Problemy Kibernetiki* 2.1 (1959), pp. 75–121 (cit. on p. 2).

[Yab59b] Sergey V Yablonski. “On the impossibility of eliminating perebor in solving some problems of circuit theory”. In: *Doklady Akademii Nauk SSSR* 124.1 (1959), pp. 44–47 (cit. on p. 2).

A Correctness Sketch for Fiat-Naor

In this part, we sketch the correctness proof of the Fiat-Naor algorithm. We want to show that for every $y \in f(\mathcal{D})$, and over a random choice of the function g and the points $x_1, \ldots, x_m \leftarrow [2^n]$, the algorithm succeeds in finding a pre-image of y with probability mt/N. Moreover, to bound the running time of the algorithm we need to show that the number of false-positive indexes is small with high probability.

For the first part, observe that if y is covered by one of the chains, namely $y = f(h^k(x_i))$ for some $i \in [m]$ and $k \le t$, then A finds a pre-image of y. As there are $m \cdot t$ choices for k and i, if the values $h^k(x_i)$ were uniformly and independently chosen, the success probability of A would be roughly mt/N. Of course, the values $h^k(x_i)$ are dependent on the function f and g and are not independent. Yet, [FN00] showed that for the right choice of m and t, this is not far from the real distribution. Specifically, [FN00] showed that this is the case if $mt^2U \approx 2^n$. We now sketch the proof.

The probability that y is covered by a specific chain. Fix $i \in [m]$. We start by showing that when $t^2U << 2^n$, the distribution of the chain $(x_i, h(x_i), \ldots, h^t(x_i))$, over the choice of x_i and g, is close (in statistical distance) to the distribution of t randomly distributed points in $\mathcal{D}$. Indeed, recall that $g \colon [2^n] \to \mathcal{D}$ is a random function, and $h = g \circ f$. Thus, for every prefix of length j of the chain $(x_i, h(x_i), \ldots, h^j(x_i))$, if $f(h^j(x_i))$ did not appear as an image in the chain earlier (that is, $f(h^j(x_i)) \neq f(h^k(x_i))$ for every $k \le j$), then the next point in the chain, $g(f(h^j(x_i)))$ is uniformly distributed. Thus the above statistical distance is bounded by the probability that for random $w_1, \ldots, w_t \leftarrow \mathcal{D}$, there are $i \neq j$ such that $f(w_i) = f(w_j)$. As every image of f has at most U pre-images, for every fixed i and j it holds that $\Pr[f(w_i) = f(w_j)] \le U/|\mathcal{D}| \approx U/2^n$ (recall that we assumed that $|\mathcal{D}| \ge 2^n/2$). By taking union bound on all possible t^2 pairs of indexes, we get that the probability that there exists some pair is at most $t^2U/2^n \ll 1$. This shows that every chain contains y with probability $\approx t/2^n$

The probability that y is covered by one of the chains. Next, [FN00] show that when $mt^2U < 2^n$, the m chains rooted with $x_1, \ldots, x_m$ contains roughly $m \cdot t$ different images, and more importantly, contain y with probability $mt/2^n$. This can be done by showing that with a good probability, for every $i \leq m$, the i-th chain does not collide with any of the first $i-1$ chains. Thus, every chain covers $\Omega(t)$ new images in expectation, and thus increases the inversion probability by $\Omega(t/2^n)$. The proof that the i-th chain indeed does not collide with the first $i-1$ chains is similar to the above proof that every chain covers t images: fix the first $i-1$ chains. These chains contain at most $(i-1)t \leq m \cdot t$ different images. Thus, the probability that the image of a random input $w \leftarrow [2^n]$ is equal to the image of some point in one of the chains is at most $mtU/2^n$. Since the i-th chain contains t points, each one of them is random as long as there was no collision yet, we get that the collision probability is at most $mt^2U/2^n < 1$.[4]

Bounding the false-positive probability. To bound the number of false-positive indexes, we now bound the probability that the chain $(g(y), h(g(y)), h^2(g(y)), \ldots, h^t(g(y))$ collide with the chain $(x_i, h(x_i), \ldots, h^t(x_i))$ for some fixed i. Similarly to the previous argument, since as long as there is no collision the next point in each chain is random, the collision probability of any specific pair is at most $U/2^n$. Since there are at t^2 such pairs, the collision probability is $Ut^2/2^n < 1/m$. Thus, on expectation, we will see one false positive index per repetition of the algorithm A. In total for all the ℓ repetitions, so we will see less than 10ℓ such false-positive indexes with a good probability.

[4]In [FN00] the function g is not a random function, and the chains are not independent. Instead, the function g is $2t$-wise independent, which implies that the chains' distribution is pair-wise independent. This is enough for the proof to go through, using the the inclusion/exclusion principle.