Electronic Colloquium on Computational Complexity, Report No. 56 (2022)

Optimal Coding Theorems in Time-Bounded Kolmogorov Complexity

Zhenjian Lu* Igor C. Oliveira† Marius Zimand‡

April 18, 2022

Abstract

The classical coding theorem in Kolmogorov complexity states that if an n-bit string x is sampled with probability δ by an algorithm with prefix-free domain then $\mathsf{K}(x) \leq \log(1/\delta) + O(1)$. In a recent work, Lu and Oliveira [LO21] established an unconditional time-bounded version of this result, by showing that if x can be efficiently sampled with probability δ then $\mathsf{rKt}(x) = O(\log(1/\delta)) + O(\log n)$, where rKt denotes the randomized analogue of Levin's Kt complexity. Unfortunately, this result is often insufficient when transferring applications of the classical coding theorem to the time-bounded setting, as it achieves a $O(\log(1/\delta))$ bound instead of the information-theoretic optimal $\log(1/\delta)$.

Motivated by this discrepancy, we investigate optimal coding theorems in the time-bounded setting. Our main contributions can be summarised as follows.

• **Efficient coding theorem for rKt with a factor of 2.** Addressing a question from [LO21], we show that if x can be efficiently sampled with probability at least δ then $\mathsf{rKt}(x) \leq (2 + o(1)) \cdot \log(1/\delta) + O(\log n)$. As in previous work, our coding theorem is *efficient* in the sense that it provides a polynomial-time probabilistic algorithm that, when given x, the code of the sampler, and δ, it outputs, with probability ≥ 0.99, a probabilistic representation of x that certifies this rKt complexity bound.

• **Optimality under a cryptographic assumption.** Under a hypothesis about the security of cryptographic pseudorandom generators, we show that no efficient coding theorem can achieve a bound of the form $\mathsf{rKt}(x) \leq (2 - o(1)) \cdot \log(1/\delta) + \mathsf{poly}(\log n)$. Under a weaker assumption, we exhibit a gap between *efficient* coding theorems and *existential* coding theorems with near-optimal parameters.

• **Optimal coding theorem for pK^t and unconditional Antunes-Fortnow.** We consider pK^t complexity [GKLO22], a variant of rKt where the randomness is public and the time bound is fixed. We observe the existence of an optimal coding theorem for pK^t, and employ this result to establish an *unconditional* version of a theorem of Antunes and Fortnow [AF09] which characterizes the worst-case running times of languages that are in average polynomial-time over all P-samplable distributions.

*University of Warwick, UK. E-mail: zhen.j.lu@warwick.ac.uk
†University of Warwick, UK. E-mail: igor.oliveira@warwick.ac.uk
‡Towson University, US. E-mail: mzimand@towson.edu.

ISSN 1433-8092

Contents

1 Introduction

1.1 Context and Background

A sampler is a probabilistic function that outputs Boolean strings. For any string $x \in \{0,1\}^*$ in its range, let $\mu(x)$ denote the probability with which x is generated. The Coding Theorem in Kolmogorov complexity states that if the sampler is computable and its domain is a prefix-free set, then for every x in its range

$$\mathsf{K}(x) \leq \log(1/\mu(x)) + O(1),$$

where $\mathsf{K}(\cdot)$ is the prefix-free Kolmogorov complexity. In other words, strings that are sampled with non-trivial probability have short representations. Note that the coding theorem achieves *optimal* expected length, since no uniquely decodable code can have expected length smaller than $\sum \mu(x) \log_2(1/\mu(x))$, the entropy of the sampler (the sum is over all x in the range of the sampler, assumed here to be finite).

The coding theorem is a central result in Kolmogorov complexity.[1] While it has found a number of applications in theoretical computer science (see, e.g., [LV92, Lee06, Aar14]), it comes with an important caveat: many aspects of the theory of Kolmogorov complexity are *non-constructive*. For instance, there is provably no algorithm that estimates $\mathsf{K}(x)$. Similarly, for arbitrary samplers, there is no effective compressor achieving the short representation provided by the coding theorem[2] and also no upper bound on the running time required to decompress x from it.

In order to translate results and techniques from Kolmogorov complexity to the setting of *efficient* algorithms and computations, several *time-bounded* variants of Kolmogorov complexity have been proposed. We refer to the book [LV19], thesis [Lee06], and the surveys [All92, All01, For04, All17] for a comprehensive treatment of this area and its numerous applications to algorithms, complexity, cryptography, learning, and pseudorandomness, among other fields. We highlight that many exciting new results, which include worst-case to average-case reductions for NP problems [Hir18, Hir21] and complexity-theoretic characterizations of one-way functions [LP20, RS21], rely in a crucial way on time-bounded Kolmogorov complexity. These recent developments further motivate the investigation of key results from Kolmogorov complexity in the time-bounded setting.

In time-bounded Kolmogorov complexity we consider the minimum description length of a string x *with respect to machines that operate under a time constraint*. We informally review next two central notions in this area (see Section 2 for precise definitions). For a Turing machine $\mathcal{M}$, we let $|\mathcal{M}|$ denote its description length according to a fixed universal machine U. $\mathcal{M}(\varepsilon)$ denotes the computation of $\mathcal{M}$ over the empty string.

Kt Complexity [Lev84]. This notion simultaneously considers description length and running time when measuring the complexity of a string x.

$$\mathsf{Kt}(x) \;=\; \min_{\text{TM } \mathcal{M},\, t \geq 1} \{|\mathcal{M}| + \log t \mid \mathcal{M}(\varepsilon) \text{ outputs } x \text{ in } t \text{ steps}\}\,.$$

[1] For instance, [Lee06] describes it as one of the four pillars of Kolmogorov complexity.

[2] However, there exists a probabilistic polynomial-time compressor that given x and an integer $m \geq \log(1/\mu(x))$ outputs a description of x of length m + small polylogarithmic overhead [BZ19].

K^t Complexity [Sip83]. In contrast with Kt, here we fix the time bound $t\colon \mathbb{N} \to \mathbb{N}$, and consider the minimum description with respect to machines that run in time at most $t(|x|)$.

$$\mathsf{K}^t(x) \;=\; \min_{\mathrm{TM}\,\mathcal{M}} \{|\mathcal{M}| \mid \mathcal{M}(\varepsilon) \text{ outputs } x \text{ in } t(|x|) \text{ steps}\}\,.$$

While Kt complexity is tightly related to optimal search algorithms (see [Kra21] for a recent application), K^t is particularly useful in settings where maintaining a polynomial bound on the running time t is desired (see, e.g., [Hir18]).

Antunes and Fortnow [AF09] introduced techniques that can be used to establish (conditional) coding theorems for K^t and Kt. In particular, if a sampler runs in *polynomial time* and outputs a string x with probability at least δ, then $\mathsf{Kt}(x) \leq \log(1/\delta) + O(\log n)$. Note that this coding theorem also achieves an optimal dependence on the probability parameter δ. However, the results of [AF09] rely on a strong derandomization assumption. For this reason, their application often lead to *conditional* results.

More recently, [LO21] established an *unconditional* coding theorem for a *randomized* analogue of Kt complexity. Before explaining their result, we review the definitions of rKt and rK^t.

rKt Complexity [Oli19]. In this definition, we consider randomized machines that output x with high probability.

$$\mathsf{rKt}(x) \;=\; \min_{\mathrm{RTM}\,\mathcal{M},\, t\geq 1} \{|\mathcal{M}| + \log t \mid \mathcal{M}(\varepsilon) \text{ outputs } x \text{ in } t \text{ steps with probability} \geq 2/3\}\,.$$

rK^t Complexity [BLvM05, LOS21].[3] This is the randomized analogue of K^t, where the time bound t is fixed in advance.

$$\mathsf{rK}^t(x) \;=\; \min_{\mathrm{RTM}\,\mathcal{M}} \{|\mathcal{M}| \mid \mathcal{M}(\varepsilon) \text{ outputs } x \text{ in } t(|x|) \text{ steps with probability} \geq 2/3\}\,.$$

In both cases, we can think of the randomized Turing machine $\mathcal{M}$ as a *probabilistic representation* of the input string x, in the sense that x can be recovered with high probability from its description. These measures allow us to employ methods from time-bounded Kolmogorov complexity in the setting of randomized computation, which is ubiquitous in modern computer science. For instance, [Oli19, LOS21] employed rKt and rK^t to obtain bounds on the compressibility of prime numbers and other objects and to show that certain problems about time-bounded Kolmogorov complexity can be intractable. We note that, under derandomization assumptions (see [Oli19]), for every string x, $\mathsf{rKt}(x) = \Theta(\mathsf{Kt}(x))$. Similarly, one can conditionally show that $\mathsf{K}^t(x)$ is essentially $\mathsf{rK}^t(x)$, up to a $O(\log |x|)$ additive term (see [GKLO22]). Consequently, insights obtained in the context of probabilistic notions of Kolmogorov complexity can often inform the study of more classical notions such as Kt and K^t.

Among other results, [LO21] established the following unconditional coding theorem in time-bounded Kolmogorov complexity: if a sampler runs in polynomial time and outputs a string x with probability at least δ, then $\mathsf{rKt}(x) = O(\log(1/\delta) + O(\log n)$. While this result can be used to port some applications of the coding theorem from Kolmogorov complexity to the time-bounded setting, in many cases it is still insufficient. This is because its dependence on the probability parameter δ is not optimal, which is often crucial in applications (see, e.g., [AF09, Aar14]).

[3] [BLvM05] refers to this notion as CBP^t complexity.

1.2 Results

In this work, we investigate optimal coding theorems in time-bounded Kolmogorov complexity. We describe our results next.

1.2.1 A Tighter Efficient Coding Theorem

Our first result addresses the question posed in [LO21, Problem 37].

Theorem 1. *Suppose there is an efficient algorithm A for sampling strings such that $A(1^n)$ outputs a string $x \in \{0,1\}^n$ with probability at least δ. Then*

$$\mathsf{rKt}(x) \;\leq\; 2\log(1/\delta) + O\big(\log n + \log^2 \log(1/\delta)\big)\,,$$

where the constant behind the $O(\cdot)$ depends on A and is independent of the remaining parameters. Moreover, given x, the code of A, and δ, it is possible to compute in time $\mathsf{poly}(n, |A|)$, with probability ≥ 0.99, a probabilistic representation of x certifying this rKt-complexity bound.

In [BFL01, Lemma 4], it was observed that by hashing modulo prime numbers one can obtain short descriptions of strings. As discussed in [LO21, Section A.2.1], for each efficient sampling algorithm, this technique implies that if some string x is produced with probability $\geq \delta$, then $\mathsf{rKt}(x) \leq 3\log(1/\delta) + O(\log n)$.[4] In contrast, Theorem 1 achieves a bound of the form $(2 + o(1)) \cdot \log(1/\delta) + O(\log n)$.

Theorem 1 readily improves some parameters in the applications of the coding theorem for rKt discussed in [LO21], such as the efficient instance-based search-to-decision reduction for rKt. We omit the details.

In Section 3.1, we discuss extensions of this result. In particular, we describe precise bounds on the running time used in producing the corresponding probabilistic representation, and discuss computational aspects of the compression and decompression of x in detail. In Appendix A, we discuss the computation of a probabilistic representation of the string x when one does not know a probability bound δ.

1.2.2 Matching Lower Bound Under a Cryptographic Assumption

It is possible to extend techniques from [AF09] to show the following conditional result (see Section 3.2).

Proposition 2. *Assume there is a language $L \in \mathsf{BPTIME}\left[2^{O(n)}\right]$ that requires nondeterministic circuits of size $2^{\Omega(n)}$ for all but finitely many n. Suppose there is an efficient algorithm A for sampling strings such that $A(1^n)$ outputs a string $x \in \{0,1\}^n$ with probability at least $\delta > 0$. Then*

$$\mathsf{rKt}(x) \;\leq\; \log(1/\delta) + O(\log n).$$

While Proposition 2 provides a better bound than Theorem 1, the result is only *existential*, i.e., it does not provide an efficient algorithm that produces a probabilistic representation of x. In other words, Proposition 2 does not establish an *efficient* coding theorem. Our next result shows that the bound achieved by Theorem 1 is optimal for efficient coding theorems, under a

[4] The bound from [LO21, Section A.2.1] is different because it does not take into account the running time, which incurs an additional overhead of $\log(1/\delta)$.

cryptographic assumption.

The Cryptographic Assumption. For a constant $\gamma \in (0,1)$, we introduce the γ-Crypto-ETH assumption, which can be seen as a cryptographic analogue of the well-known exponential time hypothesis about the complexity of k-CNF SAT [IP01]. Informally, we say that γ-Crypto-ETH holds if there is a pseudorandom generator $G\colon \{0,1\}^{\ell(n)} \to \{0,1\}^n$ computable in time $\mathrm{poly}(n)$ that fools *uniform algorithms* running in time $2^{\gamma \cdot \ell(n)}$. Any seed length $(\log n)^{\omega(1)} \leq \ell(n) \leq n/2$ is sufficient in our negative results.

In analogy with the well-known ETH and SETH hypotheses about the complexity of k-CNF SAT, we say that Crypto-ETH holds if γ-Crypto-ETH is true for some $\gamma > 0$, and that Crypto-SETH holds if γ-Crypto-ETH is true for every $\gamma \in (0,1)$. Since a candidate PRG of seed length $\ell(n)$ can be broken in time $2^{\ell(n)}\mathsf{poly}(n)$ by trying all possible seeds, these hypotheses postulate that for some PRGs one cannot have an attack that does sufficiently better than this naive brute-force approach.

We stress that these assumptions refer to uniform algorithms. In the case of non-uniform distinguishers, it is known that Crypto-SETH does not hold (see [FN99, DTT10, CGLQ20] and references therein). We provide a formal treatment of the cryptographic assumption in Section 4.

Theorem 3 (Informal)**.** *Let $\gamma \in (0,1)$ be any constant. If γ-Crypto-ETH holds, there is no efficient coding theorem for* rKt *that achieves bounds of the form* $(1+\gamma-o(1)) \cdot \log(1/\delta) + \mathsf{poly}(\log n)$.

Theorem 3 shows that if Crypto-ETH holds then the best parameter achieved by an *efficient* coding theorem for rKt is $(1+\Omega(1)) \cdot \log(1/\delta) + \mathsf{poly}(\log n)$. This exhibits an inherent gap in parameters between the efficient coding theorem (Theorem 1) and its existential analogue (Proposition 2). On the other hand, if the stronger Crypto-SETH hypothesis holds, then no efficient coding theorem for rKt achieves parameter $(2-o(1)) \cdot \log(1/\delta) + \mathsf{poly}(\log n)$. In this case, Theorem 1 is essentially optimal with respect to its dependence on δ.

Fine-grained complexity of coding algorithms for polynomial-time samplers. An rKt bound refers to the time necessary to *decompress* a string x from its probabilistic representation. On the other hand, an *efficient* coding theorem provides a routine that can *compress* x in polynomial time. More generally, a coding procedure for a sampler A consists of a pair of probabilistic algorithms (Compress, Decompress) that aim to produce a "good" codeword p for every string y sampled by A. The quality of p depends on three values: the length of p, the number of steps t_C used to produce p from y (the compression time), and the number of steps t_D used to produce y from p (the decompression time). It is interesting to understand the trade-off between these three values. Toward this goal, we aggregate them in a manner similar to rKt, by defining the 2-sided-rKt complexity of y to be, roughly, $|p| + \log(t_C + t_D)$ (the formal Definition 25 is more complicated because it takes into account that Compress and Decompress are probabilistic). Thus according to 2-sided-rKt, each bit gained by a shorter codeword is worth doubling the compression/decompression time. For instance, for simple samplers (say, having a finite range, or generating strings with the uniform distribution), there exist trivial polynomial time Compress and Decompress, which in case y is sampled with probability at least δ, produce a codeword p with $|p| = \log(1/\delta)$ (provided Compress and Decompress know δ). Such a coding procedure certifies for each sampled string a 2-sided-rKt complexity of $\log(1/\delta) + O(\log n)$. We say that the sampler admits coding with 2-sided-rKt

complexity bounded by $\log(1/\delta) + O(\log n)$. In general, we have to include also the error probability of Compress and Decompress, which we omit in this informal discussion.

Similarly to Theorem 1 and Theorem 3 (and also with similar proofs), we establish the following theorem.

Theorem 4 (Informal)**.** *The following results hold.*

(*a*) (Upper Bound) *Every polynomial-time sampler admits coding with* 2*-sided-*rKt *complexity* $2\log(1/\delta) + O(\log^2 \log(1/\delta)) + O(\log n)$.

(*b*) (Conditional Lower Bound) *Let* $\gamma \in (0,1)$ *be any constant. If* γ*-*Crypto-ETH *holds, there exists a polynomial-time sampler that does not admit coding with* 2*-sided-*rKt *complexity bounded by* $(1+\gamma-o(1)) \cdot \log(1/\delta) + \mathsf{poly}(\log n)$*, unless the error probability is greater than* $1/7$.

1.2.3 An Optimal Coding Theorem and Unconditional Antunes-Fortnow

While Theorem 1 improves the result from [LO21] to achieve a bound that is tight up to a factor of 2 and that is possibly optimal among efficient coding theorems, it is still insufficient in many applications. We consider next a variant of rKt that allows us to establish an *optimal* and *unconditional* coding theorem in time-bounded Kolmogorov complexity.

Fix a function $t\colon \mathbb{N} \to \mathbb{N}$. For a string $x \in \{0,1\}^*$, the probabilistic t-bounded Kolmogorov complexity of x (see [GKLO22]) is defined as

$$\mathsf{pK}^t(x) = \min\left\{ k \;\middle|\; \Pr_{w\sim\{0,1\}^{t(|x|)}}\left[\exists\, \mathcal{M} \in \{0,1\}^k,\ \mathcal{M}(w) \text{ outputs } x \text{ within } t(|x|) \text{ steps}\right] \geq \frac{2}{3} \right\}.$$

In other words, if $k = \mathsf{pK}^t(x)$, then with probability at least $2/3$ over the choice of the random string w, x admits a time-bounded encoding of length k. In particular, if two parties share a typical random string w, then x can be transmitted with k bits and decompressed in time $t = t(|x|)$. (Recall that here the time bound t is fixed, as opposed to rKt, where a $\log t$ term is added to the description length.)

It is possible to show that $\mathsf{K}^t(x)$, $\mathsf{rK}^t(x)$, and $\mathsf{pK}^t(x)$ correspond essentially to the same time-bounded measure, under standard derandomization assumptions [GKLO22].[5] One of the main benefits of pK^t is that it allows us to establish unconditional results that are currently unknown in the case of the other measures.[6]

Theorem 5. *Suppose there is a randomized algorithm* A *for sampling strings such that* $A(1^n)$ *runs in time* $T(n) \geq n$ *and outputs a string* $x \in \{0,1\}^n$ *with probability at least* $\delta > 0$*. Then*

$$\mathsf{pK}^t(x) \;=\; \log(1/\delta) + O(\log T(n))\,,$$

where $t(n) = \mathrm{poly}(T(n))$ *and the constant behind the* $O(\cdot)$ *depends on* $|A|$ *and is independent of the remaining parameters.*

[5] More precisely, under standard derandomization assumptions, $\mathsf{pK}^t(x)$ and $\mathsf{rK}^{t'}(x)$ coincide up to an additive term of $O(\log|x|)$, provided that $t' = \mathrm{poly}(t)$. A similar relation holds between K^t and rK^t.

[6] While in this work we focus on coding theorems, we stress that pK^t is a key notion introduced in [GKLO22] that enables the investigation of meta-complexity in the setting of probabilistic computations. It has applications in worst-case to average-case reductions and in learning theory.

Theorem 5 provides a time-bounded coding theorem that can be used in settings where the optimal dependence on δ is crucial. As an immediate application, it is possible to show an equivalence between efficiently sampling a fixed sequence $w_n \in \{0,1\}^n$ of objects (e.g., n-bit prime numbers) with probability at least $\delta_n/\mathsf{poly}(n)$ and the existence of bounds for the corresponding objects of the form $\mathsf{pK}^{\mathrm{poly}}(w_n) = \log(1/\delta_n) + O(\log n)$.[7] This is the first tight equivalence of this form in time-bounded Kolmogorov complexity that does not rely on an unproven assumption.

As a more sophisticated application of Theorem 5, we establish an unconditional form of the main theorem from Antunes and Fortnow [AF09], which provides a characterization of the worst-case running times of languages that are in average polynomial-time over all P-samplable distributions.

We recall the following standard notion from average-case complexity (see, e.g., [BT06]). For an algorithm A that runs in time $T_A\colon \{0,1\}^* \to \mathbb{N}$ and for a distribution $\mathcal{D}$ supported over $\{0,1\}^*$, we say that A runs in polynomial-time on average with respect to $\mathcal{D}$ if there is some constant $\varepsilon > 0$ such that

$$\mathop{\mathbf{E}}_{x\sim\mathcal{D}}\left[\frac{T_A(x)^\varepsilon}{|x|}\right] < 1.$$

As usual, we say that a distribution $\mathcal{D}$ is P-samplable if it can be sampled in polynomial time.

Theorem 6. *The following conditions are equivalent for any language $L \subseteq \{0,1\}^*$.*

(*i*) *For every P-samplable distribution $\mathcal{D}$, L can be solved in polynomial-time on average with respect to $\mathcal{D}$.*

(*ii*) *For every polynomial p, there exists a constant $b > 0$ such that the running time of some algorithm that computes L is bounded by $2^{O(\mathsf{pK}^p(x) - \mathsf{K}(x) + b\log(|x|))}$ for every input x.*

In contrast, [AF09] shows a *conditional* characterisation result that employs K^t complexity in the expression that appears in Item (*ii*).

1.3 Techniques

In this section, we provide an informal overview of our proofs and techniques.

Efficient Coding Theorem for rKt (Theorem 1). Breaking down the result into its components, Theorem 1 shows that for any polynomial-time sampler A, there exist a probabilistic polynomial-time algorithm $\mathsf{Compress}$ and an algorithm $\mathsf{Decompress}$ with the following properties: $\mathsf{Compress}$ on input an n-bit string x and δ (which estimates from below the probability with which A samples x), returns a codeword c_x of length $\log(1/\delta) + \mathrm{poly}(\log n)$ such that $\mathsf{Decompress}$ with probability ≥ 0.99 reconstructs x in time $1/\delta \cdot \exp(\mathrm{poly}(\log n))$. Note that the probabilistic representation of x certifying the rKt bound in Theorem 1 can be obtained from the codeword c_x and $\mathsf{Decompress}$, and that obtaining a running time with a factor of $(1/\delta)^{1+o(1)}$ is crucial in order to get a final rKt bound of the form $(2+o(1))\cdot\log(1/\delta)$. (Actually, $\mathsf{Compress}$ does not have to depend on A, the 0.99 can be $1-\varepsilon$ for arbitrary $\varepsilon > 0$,

[7] An efficient sampler immediately implies the corresponding pK^t bounds via Theorem 5. On the other hand, objects of bounded pK^t complexity can be sampled by considering a random sequence of bits and a random program of appropriate length. We refer to [LO21, Theorem 6] for a weaker relation and its proof. Since the argument is essentially the same, we omit the precise details.

and the poly$(\log n)$ term is $O(\log n + \log^2 \log(1/\delta))$, but we omit these details in our discussion). We explain what are the challenges in obtaining Compress and Decompress and how they are overcome. We remark that the construction is different from the approaches described in [LO21].

Decompress can run the sampler $K := O(1/\delta)$ times and obtain a list of elements S^* (the list of suspects) that with high probability contains x. Compress has to provide information that allows Decompress to prune S^* and find x. Since the algorithms do not share randomness, Compress does not know S^*, and so compression has to work for any $S \subseteq \{0,1\}^n$ of size K, only assuming that $x \in S$. Compress can use a bipartite lossless expander graph G, which is a graph with the property that any set S of left nodes with size $|S| \leq K$ has at least $(1-\varepsilon)D|S|$ neighbors, where D is the left degree. Such graphs are called $((1-\varepsilon)D, K)$ lossless expanders and they have numerous applications (see e.g., [CRVW02, HLW06]). An extension of Hall's matching theorem shows that for any set S of K left nodes, there is a matching that assigns to each $x \in S$, $(1-\varepsilon)D$ of its neighbors, so that no right node is assigned twice (i.e., the matching defines a subgraph with no collisions). Compress can just pick the codeword c_x to be one random neighbor of x. Then, Decompress can do the pruning of S^* as follows. Having S^* and c_x, it does the matching, and, since with probability $1-\varepsilon$, c_x is only assigned to x, Decompress can find x. There is one problem though. The algorithms for maximum matching in general bipartite graphs do not run in linear time (see [CKL+22, Mad13], and the references therein). Therefore, the decompression time would have a dependency on δ too large for us. Fortunately, lossless expanders can be used to do "almost" matching faster. [BZ19] introduces *invertible functions* (see Definition 12) for the more demanding task in which the elements of S appear one-by-one and the matching has to be done in the online manner. We do not need online matching, but we take advantage of the construction in [BZ19] to obtain a fast matching algorithm. It follows from [BZ19], that in a lossless expander it is possible to do a greedy-type of "almost" matching, which means that every left node in S is matched to $(1-\varepsilon)D$ of its neighbors (exactly what we need), but with poly$(\log n)$ collisions. The collisions can be eliminated with some additional standard hashing (see the discussion on page 14 for details). As we explain on page 14, this leads to decompression time $K \cdot D \cdot \text{poly}(n)$ and the length of the codeword c_x is $\log |R| + |\text{hash-code}|$, where R is the right set of the lossless expander. To obtain our result, the degree D has to be $2^{\text{poly}(\log n)}$ and $|R|$ has to be $K \cdot 2^{\text{poly}(\log n)}$.

Building on results and techniques from [GUV09], [BZ19] constructs a $((1-\varepsilon)D, K)$ explicit lossless expander with left side $\{0,1\}^n$, degree $D = 2^d$ for $d = O(\log(n/\varepsilon) \cdot \log k)$, and right side R, with size verifying $\log |R| = k + \log(n/\varepsilon) \cdot \log k$ (where $k := \log K$). To obtain in Theorem 1 the dependency on n to be $O(\log n)$ (which is optimal up to the constant in $O(\cdot)$), we show the existence of a $((1-\varepsilon)D, K)$ explicit expander with $d = O(\log n + \log(k/\varepsilon) \cdot \log k)$ and $\log |R| = k + O(\log n + \log(k/\varepsilon) \cdot \log k)$. This lossless expander is constructed by a simple composition of the above lossless expander from [BZ19] with a lossless expander from [GUV09], with an appropriate choice of parameters (see Section 3.1.1).

Conditional Lower Bound for Efficient Coding Theorems (Theorem 3). Our goal is to show that there is no *efficient* coding theorem for rKt that achieves bounds of the form $(1+\gamma-o(1)) \cdot \log(1/\delta) + \text{poly}(\log n)$, under the assumption that γ-Crypto-ETH holds for $\gamma \in (0,1)$. We build on an idea attributed to L. Levin (see e.g. [Lee06, Section 5.3]). To provide an overview of the argument, let $G_n \colon \{0,1\}^{\ell(n)} \to \{0,1\}^n$ be a cryptographic generator of seed length $\ell(n) = n/2$ witnessing that γ-Crypto-ETH holds. In other words, G_n has security $2^{\gamma \cdot \ell(n)}$

against uniform adversaries. We define a sampler S_n as follows. On input $x \in \{0,1\}^n$, which we interpret as a random string, it outputs $G_n(x')$, where x' is the prefix of x of length $\ell(n)$. We argue that if an *efficient* algorithm F is able to compress every string y in the support of $\mathsf{Dist}(S_n)$, the distribution induced by the sampler S_n, to an rKt encoding of complexity $(1+\gamma-\varepsilon)\cdot\log(1/\delta'(y)) + C\cdot(\log n)^C$, where $\delta'(y)$ is a lower bound on $\delta(y)$ (the probability of y under $\mathsf{Dist}(S_n)$), we can use F to break G_n. (Note that F expects as input n, y, δ', and $\mathsf{code}(S)$.)

The (uniform) distinguisher D computes roughly as follows. Given a string $z \in \{0,1\}^n$, which might come from the uniform distribution U_n or from $G_n(U_{\ell(n)}) \equiv \mathsf{Dist}(S_n)$, D attempts to use F to *compress* z to a "succinct" representation, then checks if the computed representation *decompresses* to the original string z. If this is the case, it outputs 1, otherwise it outputs 0. (Note that we haven't specified what "succinct" means, and it is also not immediately clear how to run F, since it assumes knowledge of a probability bound δ'. For simplicity of the exposition, we omit this point here.) We need to argue that a test of this form can be implemented in time $2^{\gamma\cdot\ell(n)}$, and that it distinguishes the output of G from a random string.

To achieve these goals, first note that a typical random string cannot be compressed to representations of length, say, $n-\mathsf{poly}(\log n)$, even in the much stronger sense of (time-unbounded) Kolmogorov complexity. Therefore, with some flexibility with respect to our threshold for succinctness, the proposed distinguisher is likely to output 0 on a random string. On the other hand, if F implements an efficient coding theorem that achieves rKt encodings of complexity $(1+\gamma-\varepsilon)\cdot\log(1/\delta'(y)) + \mathsf{poly}(\log n)$, the following must be true. Using that the expected *encoding length* of *any* (prefix-free) encoding scheme is at least $H(\mathsf{Dist}(S_n))$, where $\mathsf{Dist}(S_n)$ is the distribution of strings sampled by S_n and H is the entropy function, we get (via a slightly stronger version of this result) that a non-trivial measure of strings y in the support of $\mathsf{Dist}(S_n)$ have rKt *encoding length* at least $(1-\varepsilon/4)\cdot\log(1/\delta(y))$. Consequently, for such strings, an upper bound on rKt complexity of $(1+\gamma-\varepsilon)\cdot\log(1/\delta'(y)) + \mathsf{poly}(\log n)$ when $\delta'(y)$ is sufficiently close to $\delta(y)$ implies that the running time t of the underlying machine satisfies $\log t \le (\gamma-\varepsilon/2)\log(1/\delta(y)) + \mathsf{poly}(\log n)$. Using that $\ell(n) = n/2$ and $\delta(y) \ge 2^{-\ell(n)}$ for any string y in the support of $\mathsf{Dist}(S_n)$, it is easy to check that (asymptotically) $t \le 2^{(\gamma-\varepsilon/4)\cdot\ell(n)}$. For this reason, we can implement a (slightly modified) distinguisher D in time less than $2^{\gamma\cdot\ell(n)}$, by trying different approximations $\delta'(z)$ for an input string z and by running the decompressor on the produced representation for at most t steps on each guess for $\delta(z)$. By our previous discussion, a non-trivial measure of strings from $\mathsf{Dist}(S_n)$ will be accepted by D, while only a negligible fraction of the set of all strings (corresponding to the random case) will be accepted by D.

Implementing this strategy turns out to be more subtle than this. This happens because F is a *probabilistic* algorithm which does not need to commit to a *fixed* succinct encoding. We refer to the formal presentation in Section 4 for details, where we also discuss the bound on the seed length $\ell(n)$.

Coding Theorem for pK^t (Theorem 5) and Unconditional [AF09] (Theorem 6). The proof of our optimal coding theorem for pK^t builds on that of the *conditional* coding theorem for K^t from [AF09], which can be viewed as a two-step argument. Roughly speaking, the first step is to show that if there is a polynomial-time sampler that outputs a string $x \in \{0,1\}^n$ with probability δ, then the polynomial-time-bounded Kolmogorov complexity of x is about $\log(1/\delta) + O(\log n)$ *if we are given a random string.* After this, they "derandomize" the use

of random strings using a certain pseudorandom generator, which exists under a strong derandomization assumption. Our key observation is that the use of random strings arises naturally in probabilistic Kolmogorov complexity, and particularly in this case the random strings can be "embedded" into the definition of pK^t. As a result, we don't need to perform the afterward derandomization as in original proof of [AF09], and hence get rid of the derandomization assumption.

Next, we describe how to use Theorem 5, together with other useful properties of pK^t, to obtain an unconditional version of Antunes and Fortnow's main result. Let μ be a Kolmogorov complexity measure, such as $\mathsf{K}^{\mathsf{poly}}$, $\mathsf{rK}^{\mathsf{poly}}$ or $\mathsf{pK}^{\mathsf{poly}}$. The key notion in the proof is the distribution (in fact, a class of semi-distributions) called m_μ, which is defined as $m_\mu(x) := 1/2^{\mu(x)}$. More specifically, following [AF09], it is not hard to show that, for every language L, L can be decided in polynomial-time on average with respect to m_μ if and only if its worst-case running time is $2^{O(\mu(x)-\mathsf{K}(x))}$ on input x (see Lemma 35). Then, essentially, to show our result we argue that L can be decided in polynomial time on average with respect to m_μ if and only if the same holds with respect to all P-samplable distributions.

Recall that if a distribution $\mathcal{D}$ *dominates* another distribution $\mathcal{D}'$ (i.e., $\mathcal{D}(x) \gtrsim \mathcal{D}'(x)$ for all x) and L is polynomial-time on average with respect to $\mathcal{D}$, then the same holds with respect to $\mathcal{D}'$ (see Definition 9 and Fact 10). Therefore, to replace m_μ above with P-samplable distributions, it suffices to show that m_μ is "universal" with respect to the class of P-samplable distributions, in the following sense.

1. m_μ dominates *every* P-samplable distribution. (This is essentially an optimal source coding theorem for the Kolmogorov measure μ.)

2. m_μ is dominated by *some* P-samplable distribution.

The above two conditions require two properties of the Kolmogorov measure μ that are somewhat conflicting: the first condition requires the notion of μ to be *general* enough so that m_μ can "simulate" *every* P-samplable distribution, while the second condition needs μ to be *restricted* enough so that m_μ can be "simulated" by *some* P-samplable (i.e., simple) distribution. For example, if μ is simply the time-unbounded Kolmogorov complexity K (or even the polynomial-space-bounded variant), then it is easy to establish an optimal source coding theorem for such a general Kolmogorov measure; however it is unclear how to sample in polynomial-time a string x with probability about $1/2^{\mathsf{K}(x)}$, so in this case μ does not satisfy the second condition. On the other hand, if μ is some restricted notion of time-bounded Kolmogrov complexity measure such as $\mathsf{K}^{\mathsf{poly}}$ or $\mathsf{rK}^{\mathsf{poly}}$, then one can obtain polynomial-time samplers that sample x with probability about $1/2^{\mathsf{K}^{\mathrm{poly}}(x)}$ or $1/2^{\mathsf{K}^{\mathrm{poly}}(x)}$ (up to a polynomial factor); however, as in [AF09], we only know how to show an *optimal* source coding theorem for $\mathsf{K}^{\mathsf{poly}}$ (or $\mathsf{rK}^{\mathsf{poly}}$) under a derandomization assumption. Therefore, in this case μ does not satisfies the first condition. Our key observation is that the notion $\mathsf{pK}^{\mathsf{poly}}$, which sits in between K and $\mathsf{K}^{\mathrm{poly}}$ (or $\mathsf{rK}^{\mathrm{poly}}$),[8] satisfies both conditions described above (see Lemmas 36 and 37).

[8] We can show that for every $x \in \{0,1\}^*$ and every computable time bound $t\colon \mathbb{N} \to \mathbb{N}$, $\mathsf{K}(x) \lesssim \mathsf{pK}^t(x) \leq \mathsf{rK}^t(x) \leq \mathsf{K}^t(x)$.

2 Preliminaries

Time-bounded Kolmogorov complexity. For a function $t\colon \mathbb{N} \to \mathbb{N}$, a string x, and a universal Turing machine U, let the time-bounded Kolmogorov complexity be defined as

$$\mathsf{K}^t_U(x) = \min_{p\in\{0,1\}^*} \{|p| \mid U(p) \text{ outputs } x \text{ in at most } t(|x|) \text{ steps}\}.$$

A machine U is said to be *time-optimal* if for every machine M there exists a constant c such that for all $x \in \{0,1\}^n$ and $t\colon \mathbb{N} \to \mathbb{N}$ satisfying $t(n) \geq n$,

$$\mathsf{K}^{ct\log t}_U(x) \leq \mathsf{K}^t_M(x) + c,$$

where for simplicity we write $t = t(n)$. It is well known that there exist time-optimal machines [LV19, Th. 7.1.1]. In this paper, we fix such a machine U, and drop the index U when referring to time-bounded Kolmogorov complexity measures. It is also possible to consider prefix-free notions of Kolmogorov complexity. However, since all our results hold up to additive $O(\log|x|)$ terms, we will not make an explicit distinction.

Henceforth we will not distinguish between a Turing machine $\mathcal{M}$ and its encoding p according to U. If p is a probabilistic Turing machine, we define $t_p \in \mathbb{N} \cup \{\infty\}$ to be the maximum number steps it takes p to halt on input λ (the empty string), where the maximum is over all branches of the probabilistic computation.

rKt complexity and probabilistic representations. A probabilistic representation of a string x is a probabilistic Turing machine p that on input λ halts with x on the output tape with probability at least $2/3$. The rKt-complexity of a string x is the minimum, over all probabilistic representations p of x, of $p + \log t_p$. A *probabilistic representation* p of x *certifies* rKt*-complexity bounded by* Γ if $|p| + \log t_p \leq \Gamma$.

Distributions and semi-distributions. We consider distributions over the set $\{0,1\}^*$. We will identify a distribution with its underlying probability density function of the form $\mathcal{D}\colon \{0,1\}^* \to [0,1]$. A distribution $\mathcal{D}$ is a *semi-distribution* if $\sum_{x\in\{0,1\}^*} \mathcal{D}(x) \leq 1$, and is simply called a *distribution* if the sum is exactly 1. In this subsection and Section 5.2, we will use the word "distribution" to refer to both distribution and semi-distribution.

Samplers. A *sampler* is a probabilistic algorithm A with inputs in $\{1\}^n$ such that $A(1^n)$ outputs a string $x \in \{0,1\}^n$.[9] It defines a family of distributions $\{\mu_{A,n}\}_{n\in\mathbb{N}}$, where $\mu_{A,n}$ is the distribution on $\{0,1\}^n$ defined by $\mu_{A,n}(x) = \mathbf{Pr}_A[A(1^n) = x]$.

Average-case complexity. We now review some standard definitions and facts from average-case complexity. We refer to the survey [BT06] for more details.

Definition 7 (Polynomial-time Samplable [BCGL92])**.** *A distribution $\mathcal{D}$ is called* P*-samplable if there exists a polynomial p and a probabilistic algorithm M such that for every $x \in \{0,1\}^*$, M outputs x with probability $\mathcal{D}(x)$ within $p(|x|)$ steps.*

[9] For simplicity, we assume that $A(1^n)$ samples a string of length n. Our coding theorems also hold for algorithms used to define P-samplable distributions, see Definition 7, with obvious changes in the proofs. Also, as in [LO21], our results can be easily generalised to samplers that on 1^n output strings of arbitrary length. In this case, while the length of x might be significantly smaller than n, an additive overhead of $\log n + O(1)$ is necessary in our coding theorems, as we need to encode 1^n.

Definition 8 (Polynomial Time on Average [Lev86])**.** *Let A be an algorithm and $\mathcal{D}$ be a distribution. We say that A runs in polynomial-time on average with respect to $\mathcal{D}$ if there exist constants ε and c such that,*

$$\sum_{x\in\{0,1\}^*} \frac{t_A(x)^\varepsilon}{|x|}\mathcal{D}(x) \leq c,$$

where $t_A(x)$ denotes the running time of A on input x. For a language L we say that L can be solved in polynomial time on average with respect to $\mathcal{D}$ if there is an algorithm that computes L and runs in polynomial-time on average with respect to $\mathcal{D}$.

Definition 9 (Domination)**.** *Let $\mathcal{D}$ and $\mathcal{D}'$ be two distributions. We say that $\mathcal{D}$* dominates *$\mathcal{D}'$ if there is a constant $c > 0$ such that for every $x \in \{0,1\}^*$,*

$$\mathcal{D}(x) \geq \frac{\mathcal{D}'(x)}{|x|^c}.$$

Fact 10 (See e.g., [AF09, Lemma 3.3])**.** *Let $\mathcal{D}, \mathcal{D}'$ be two distributions, and let A be an algorithm. If*

- *A runs in polynomial time on average with respect to $\mathcal{D}$, and*
- *$\mathcal{D}$ dominates $\mathcal{D}'$*

Then A also runs in polynomial time on average with respect to $\mathcal{D}'$.

3 Coding Theorems for rKt Complexity

3.1 Efficient Coding Theorem with Tighter Parameters

We prove the result stated in Theorem 1. It involves a function

$$\alpha(n, 1/\delta, \varepsilon) = O(\log n + (\log\log 1/\delta + \log(1/\varepsilon)) \cdot \log\log 1/\delta\,), \tag{1}$$

which bounds the additive precision term in the length of codewords. The constant hidden in $O(\cdot)$ is derived from the proof of Theorem 13 (stated below). The key fact is stated in the following result.

Theorem 11. *There exist a probabilistic polynomial-time algorithm* Compress *and a probabilistic algorithm* Decompress *such that for every n-bit string x, and every rationals $\delta > 0$ and $\varepsilon > 0$,*

- Compress *on input $x, 1/\delta, \varepsilon$ outputs a string p that has with probability 1 length $\log(1/\delta) + \alpha(n, 1/\delta, \varepsilon)$, and*
- *If x can be sampled by a polynomial-time sampler A with probability at least δ, then, with probability at least $1 - \varepsilon$,* Decompress *on input A and p outputs x. Moreover,* Decompress *on input A and p halts in $t_D := 1/\delta \cdot 2^{O(\alpha(n,1/\delta,\varepsilon))}$ steps with probability 1. The constant in $O(\cdot)$ depends on A, and the two probabilities are over the randomness of* Compress *and the randomness of* Decompress*.*

Proof. The proof uses the *invertible functions* from [BZ19]. A (k,ε)-invertible function is a probabilistic function that on input x produces a random fingerprint of x. The invertibility property requires that there exists a deterministic algorithm that on input a random fingerprint of x and a set S (the "list of suspects") of size at most 2^k that contains x, with probability $1-\varepsilon$ identifies x among the suspects.

Definition 12. *A function $F:\{0,1\}^n\times\{0,1\}^d\to\{0,1\}^{k+\Delta}$ is (k,ε)-invertible if there exists a partial function g mapping a set S of n-bit strings and a $(k+\Delta)$-bit string y into $g_S(y)\in\{0,1\}^n$ such that for every set S of size at most 2^k strings and every x in S*

$$\Pr_{\rho}[g_S(F(x,\rho))=x]\geq 1-\varepsilon. \tag{2}$$

The actual (k,ε)-invertible function that we use is given in the following theorem, which is essentially Theorem 2.1 in [BZ19], with an improvement of the precision term. (The result in [BZ19] is stronger, because the "list of suspects" S can be presented to the inverter g from Equation (2) in an *online* manner, but in our application we do not need the online feature.)

Theorem 13. *There exists a probabilistic algorithm* F *that on input $\varepsilon>0$, k and string x, uses $d=\alpha(|x|,2^k,\varepsilon)$ random bits and outputs in time polynomial in $|x|$ a string $\mathrm{F}_{\varepsilon,k}(x)$ of length $k+\alpha(|x|,2^k,\varepsilon)$, such that for all $\varepsilon>0$ and k, the function $x\mapsto\mathrm{F}_{\varepsilon,k}(x)$ is (k,ε)-invertible. Moreover, the inverter g that satisfies Equation (2) runs in time $|S|\cdot 2^d\cdot\mathrm{poly}(|x|)$.*

This theorem is proven in [BZ19] (for a somewhat larger d, but this does not affect the arguments), however the time bound is not explicitly stated. Therefore, we describe below the invertible function and verify the bound. We also need to explain how to obtain the value $d=\alpha(|x|,2^k,\varepsilon)=O(\log n+\log(k/\varepsilon)\cdot\log k)$ claimed in Theorem 13 (Theorem 2.1 in [BZ19] has $d=O(\log(n/\varepsilon)\cdot\log k)$). This is done in Section 3.1.1.

The function $F(x,\rho)$ (the random fingerprint of x) is formed by concatenating 2 strings, $F_1(x,\rho_1)$ and $F_2(x,\rho_2)$ (here, $\rho=(\rho_1,\rho_2)$ is the randomness of F). The first one is obtained by evaluating an explicit conductor (see further), and the second one is a standard hash code. The reconstruction of x from the fingerprint is done in two pruning stages. In the first stage, the conductor part will be used to reduce the list of suspects S (which includes x) of size 2^k to a list $\tilde{S}$ of size $2^{\alpha(n,2^k,\varepsilon)}$ (which also includes x w.h.p), and in the second stage, the second component of the fingerprint is used to select one string in $\tilde{S}$, which with probability $\geq 1-\varepsilon$ is x. The second reduction is simple: a greedy algorithm is used that selects the first string in $\tilde{S}$ for which the hash code matches. To distinguish a string x from s other strings in this way, one can use a hash code with prime numbers, which has size $2\log s+O(\log n)$.

The non-trivial part is the first stage, which reduces the list S of size 2^k to a list of quasi polynomial size. (A greedy algorithm would require hash codes of bitlength $2k$ instead of k.) Let $\mathcal{X}=\{0,1\}^n$ and $\mathcal{Y}=\{0,1\}^k$. A hash code $H:\mathcal{X}\times\{0,1\}^r\to\mathcal{Y}$ defines a bipartite graph with left set $\mathcal{X}$, right set $\mathcal{Y}$, and left degree 2^r. (We will use as the bipartite graph an explicit conductor graph with $r\leq\alpha(n,1/\delta,\varepsilon)$, this is the above $F_1(x,\rho_1)$.)

We explain now the first reduction of the list of suspects. Given a set S and a right node $y\in\mathcal{Y}$ (which in our application is $F_1(x,\rho_1)$), the algorithm will output at most $(1+\log|S|)2^{r+1}$ strings from S as follows. It iterates through all elements in S and selects for the output list the first 2^{r+1} left neighbors of y. If there are no more such neighbors, the algorithm is finished. Otherwise, it will compute the set S' of all left nodes for which more than a fraction 2ε of right

nodes have more than 2^{r+1} collisions. Then it will run the reduction algorithm recursively on S' and append the output list of the recursive call.

This is applied to a function H with $r \leq \alpha(n, 1/\delta, \varepsilon)$ that satisfies a conductor property. More precisely, we need the properties of a lossless conductor, an object which is essentially equivalent to a bipartite lossless expander, that we used in the high-level description in Section 1.3. This property implies that $|S'| \leq |S|/2$, and hence, by induction, we obtain the bound $(1 + \log |S|)2^{r+1}$ for the size of the output list, which is passed to the second reduction.

Let us now evaluate the runtime. In each recursive call we iterate over all elements in S and compute the list of right neighbors, which takes time $|S| \cdot 2^r \cdot \mathrm{poly}(|x|)$. To calculate S', one maintains a counter for each right node. Then we iterate through all elements of S and for each element, increment the counters of its right neighbors by one. Then we collect all right nodes with counters that exceed 2^r into the set S' and start the recursion. Thus, since a left node has 2^r neighbors, one recursive call takes $|S| \cdot 2^r \cdot \mathrm{poly}(|x|)$ steps. There may be $\log |S|$ recursive calls, but $\log |S| \leq |x|$. Thus, the total time for the first reduction is bounded by $|S| \cdot 2^r \cdot \mathrm{poly}(|x|) \leq |S| \cdot 2^d \cdot \mathrm{poly}(|x|)$. The second reduction, using standard hashing, takes time bounded by $2^d \cdot \mathrm{poly}(|x|)$. Thus, we got the claimed runtime.

We now define Compress and Decompress with the properties required in Theorem 11. We assume that $\delta \geq 2^{-|x|}$, because otherwise the trivial compressor, that compresses x to x itself, satisfies the conditions.

The function Compress on input $x, 1/\delta, \varepsilon$, takes $K := \lceil \ln(1/\varepsilon) \cdot (1/\delta) \rceil$ and runs F from Theorem 13 on input $x, \log K, \varepsilon$, which produces a string p of length $\log K + \alpha(|x|, K, \varepsilon)$. We can assume that K is a power of 2 (otherwise we replace it in the following arguments with the smallest power of 2 larger than it), and thus it can be described with $\log \log K$ bits. Compress also appends $|x|$ and the short description of K to p encoded in a self-delimited way. This takes only an additional $O(\log |x| + \log \log K)$ bits. By scaling up the constant in the definition of $\alpha(\cdot, \cdot, \cdot)$, we get that the length of p is bounded by

$$\log(1/\delta) + \alpha(|x|, K, \varepsilon).$$

Decompress first produces a list of suspects S by running the sampler A on input $1^{|x|}$ (with $|x|$ extracted from p), K times (K is also extracted from p) and taking S to be the set of samples that are obtained. We have $|S| \leq K$ and, since x is sampled with probability at least δ, the probability that x is not in S is bounded by $(1 - \delta)^K < \varepsilon$. This step takes time $K \cdot \mathrm{poly}(|x|)$, with the degree of the polynomial depending on A.

Next, Decompress runs the inverter g on input S and p. From Equation (2) we infer that if $x \in S$, then, with probability $1 - \varepsilon$, this computation reconstructs x. Overall, Decompress reconstructs x with probability $1 - 2\varepsilon$, and by the "Moreover ..." part of Theorem 13, its runtime is

$$t_D := |S| \cdot 2^d \cdot \mathrm{poly}(|x|) + K \cdot \mathrm{poly}(|x|) = K \cdot 2^{\alpha(|x|,K,\varepsilon)} \cdot \mathrm{poly}(|x|) = (1/\delta) \cdot 2^{O(\alpha(|x|,1/\delta,\varepsilon))}.$$

The conclusion follows after a rescaling of ε. □

We now readily obtain the announced result.

Corollary 14 (Efficient coding for rKt)**.** *Let $x \in \{0,1\}^n$. Suppose there is a polynomial-time sampler A such that $A(1^n)$ outputs x with probability at least $\delta > 0$. Then, for every $\varepsilon > 0$*

$$\mathsf{rKt}_\varepsilon(x) \;\leq\; 2\log(1/\delta) + O(\log(\alpha(|x|, 1/\delta, \varepsilon)),$$

where the constant hidden in $O(\cdot)$ depends on A.

Moreover, there is a probabilistic polynomial time algorithm that on input $x, 1/\delta, \varepsilon$ and the code of A, outputs with probability $1-\varepsilon$, a probabilistic representation of x certifying the above rKt*-complexity.*

Proof. Indeed, in the proof of Theorem 11, we have seen that Compress, with probability $1-\varepsilon$, produces a string p, which, given the sampler, allows the reconstruction of x with probability at least $1-\varepsilon$. More precisely, for every list of suspects S produced by Decompress with $x \in S$, a fraction of $(1-\varepsilon)$ of p's allow Decompress to reconstruct x. With a standard averaging argument, we can change the order of quantifiers, and show that for a fraction of $1-\sqrt{\varepsilon}$ of p's, it holds that for $1-\sqrt{\varepsilon}$ of S's as above, Decompress can reconstruct x.[10] Since Compress is a polynomial-time probabilistic algorithm and both the length of p and the logarithm of the reconstruction time t_D are bounded by $\log(1/\delta) + O(\alpha(|x|, 1/\delta, \varepsilon))$, the conclusion follows by scaling ε and taking the probabilistic representation of x to be $\langle p, \mathsf{code}(A)\rangle$, where $\langle\cdot,\cdot\rangle$ is some canonical self-delimiting pairing of strings. □

We can assume that $\log(1/\delta) \leq n$, since otherwise, as we have already noted, Compress can trivially simply return x. Therefore, the overhead in Corollary 14 is bounded by the simpler term $O(\log(n/\varepsilon) \cdot \log n)$.

3.1.1 Improving the Precision Term in the Invertible Function in [BZ19]

In Theorem 2.1 in [BZ19], an invertible function is constructed, in which the precision term is $O(\log(n/\varepsilon) \cdot \log k)$. We need to improve it to $O(\log n + (\log k/\varepsilon) \cdot \log k)$, to obtain the precision claimed in Theorem 13. The same idea is used in [GUV09, Th. 4.21].

We start with some standard concepts from the theory of pseudo-randomness (see [Vad12]). A *source* is a random variable whose realizations are binary strings. A source has *min-entropy* t if each value has probability at most 2^{-t}. The statistical distance between two measures P and Q with the same domain is $\sup|P(S) - Q(S)|$ (supremum is over all subsets S of the common domain of P and Q). Given a set B, we denote U_B to be a random variable that is uniformly distributed on B.

Definition 15 (Condensers and lossless conductors)**.**

(*a*) *A function $C : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a $t \to_\varepsilon t'$ condenser, if for every $S \subseteq \{0,1\}^n$ of size at least 2^t, the random variable $X = C\left(U_S, U_{\{0,1\}^d}\right)$ is ε-close to a random variable $\widetilde{X}$ that has min-entropy at least t'.*

(*b*) *A function $C : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a (t_{max}, ε) lossless conductor if it is a $t \to_\varepsilon t + d$ condenser for all $t \leq t_{max}$.*

In the proof in [BZ19], the following lossless conductor is obtained, which is next composed with a certain hash function to obtain the invertible function.

Theorem 16 ([BZ19], implicit in Theorem 2.1)**.** *For every n, $t_{max} \leq n$, $\varepsilon > 0$, there exists an explicit (t_{max}, ε) lossless conductor $C_{BZ} : \{0,1\}^n \times \{0,1\}^{d_{BZ}} \to \{0,1\}^{m_{BZ}}$ with $d_{BZ} = O(\log(n/\varepsilon) \cdot \log(t_{max}))$ and $m_{BZ} \leq t_{max} + O(\log(n/\varepsilon) \cdot \log(t_{max}))$.*

[10] The averaging argument is done with respect to the distribution over sets S induced by Decompress.

To obtain the desired improvement, we need to replace C_{BZ} in the construction in Theorem 2.1 in [BZ19] with a (t_{max}, ε) lossless conductor C with seed length $d_C = O((\log n + \log(t_{max}/\varepsilon)) \cdot \log t_{max})$ and output length $m_C \le t_{max} + O((\log n + \log(t_{max}/\varepsilon)) \cdot \log t_{max})$.

This is obtained by composing C_{BZ} with the following lossless conductor of Guruswami, Umans, and Vadhan.

Theorem 17 ([GUV09], Theorem 4.4)**.** *For every n, $t_{max} \le n$, $\varepsilon > 0$, there exists an explicit (t_{max}, ε) lossless conductor $C_{GUV} : \{0,1\}^n \times \{0,1\}^{d_{GUV}} \to \{0,1\}^{m_{GUV}}$ with $d_{GUV} \le \log n + \log(t_{max}) + \log(1/\varepsilon) + 1$ and $m_{GUV} \le d_{GUV} \cdot (t_{max} + 2)$.*

Now, as announced, we obtain the lossless conductor C, by composing C_{GUV} and C_{BZ}. Namely, $C : \{0,1\}^n \times \{0,1\}^{d_{\mathrm{GUV}} + d_{\mathrm{BZ}}} \to \{0,1\}^{m_{\mathrm{BZ}}}$ is defined by

$$C(x, (y_1, y_2)) = C_{\mathrm{BZ}}(C_{\mathrm{GUV}}(x, y_1), y_2).$$

Let us specify the parameters. We first condense with C_{GUV} for sources X with min-entropy $t \le t_{max}$ using a seed of length $d_{\mathrm{GUV}} \le \log n + \log(t_{max}) + \log(1/\varepsilon) + 1$ and obtain an output X_1 of length $m_{\mathrm{GUV}} \le d_{\mathrm{GUV}} \cdot (t_{max} + 2)$ and min-entropy at least $t + d_{\mathrm{GUV}}$. Then we further condense X_1 with C_{BZ} with parameters d_{BZ} and m_{BZ} set-up for sources with input length m_{GUV} and min-entropy bounded by $t'_{max} = t_{max} + d_{\mathrm{GUV}}$. In this way, we obtain a $(t_{max}, 2\varepsilon)$ lossless conductor for all $t \le t_{max}$, with the parameters d_C and m_C announced above, except for the case $t_{max} = o(\log n)$. But for such small t_{max}, we can take the conductor that simply outputs the seed.

Thus, we have obtained the following lossless conductor.

Theorem 18 (The new conductor)**.** *For every n, $t_{max} \le n$, $\varepsilon > 0$, there exists an explicit (t_{max}, ε) lossless conductor $C : \{0,1\}^n \times \{0,1\}^{d_C} \to \{0,1\}^{m_C}$ with $d_C = O(\log n + \log(t_{max}/\varepsilon) \cdot \log(t_{max}))$ and $m_C \le t_{max} + O(\log n + \log(t_{max}/\varepsilon) \cdot \log(t_{max}))$.*

Remark 19. *It has been observed in [TUZ07] that lossless conductors are essentially equivalent to lossless bipartite expanders, which have numerous applications. Therefore the conductor in Theorem 18 is of independent interest (it is already being used in a work in progress of some of the authors). The main lossless conductor (or, equivalently, lossless bipartite expander) of Guruswami, Umans, and Vadhan [GUV09] (different from the one in Theorem 17) has a better seed length of $d = (1 + 1/\alpha)(\log n + \log t_{max} + \log(1/\varepsilon)) + O(1)$, for any $\alpha \in (0,1)$, but the output length $m = (1+\alpha)t_{max} + 2d$ is larger, and would not have produced the optimal coding theorem.*

3.2 Existential Coding Theorem Under a Derandomization Assumption

The argument is a straightforward adaptation of a proof from [AF09], and we include a sketch here for completeness.

Theorem 20. *Assume there is a language $L \in \mathsf{BPTIME}\left[2^{O(n)}\right]$ that requires nondeterministic circuits of size $2^{\Omega(n)}$ for all but finitely many n. Suppose there is an algorithm A for sampling strings that runs in time $T(n)$ such that $A(1^n)$ outputs a string $x \in \{0,1\}^n$ with probability at least $\delta > 0$. Then*

$$\mathsf{rKt}(x) \le \log(1/\delta) + O(\log T(n)),$$

where the constant behind the $O(\cdot)$ depends on $|A|$ and is independent of the remaining parameters.

Proof Sketch. As briefly described in Section 1.3, the proof of the coding theorem in [AF09] first shows (unconditionally) that if we are given a typical random string r of length $\mathrm{poly}(T(n))$, then there exists some string $\alpha \in \{0,1\}^{\ell}$, where $\ell \leq \log(1/\delta) + O(1)$, from which one can recover the string x in time $\mathrm{poly}(T(n))$. Roughly speaking, the string r encodes a good *hitting set generator* $H\colon \{0,1\}^{\ell} \to \{0,1\}^{T(n)}$, and α is an input to H so that using $H(\alpha)$ as its internal randomness, $A(1^n)$ outputs x. Then it was observed in [AF09] (attributed to van Melkebeek) that one can further encode such an r using some string $r_0 \in \{0,1\}^{O(\log T(n))}$ if we have an optimal PRG $G\colon \{0,1\}^{O(\log T(n))} \to \{0,1\}^{|r|}$ that fools *nondeterministic circuits*. That is, $G(r_0) = r$. Then given r_0, we can eventually obtain the string r and use it along with α to recover x *deterministically* in time $\mathrm{poly}(T(n)$, which implies

$$\mathsf{Kt}(x) \leq \log(1/\delta) + O(\log T(n))\,.$$

The existence of such a PRG follows from the assumption that there exists a language $L \in \mathsf{DTIME}\big[2^{O(n)}\big]$ that requires nondeterministic circuits of size $2^{\Omega(n)}$ for all but finitely many n [SU05]. It turns out that if the language above is in $\mathsf{BPTIME}\big[2^{O(n)}\big]$ instead of $\mathsf{DTIME}\big[2^{O(n)}\big]$, we can still obtain an optimal *pseudodeterministic* PRG; this again follows from the construction in [SU05]. A PRG G is pseudodeterministic if there is a randomized algorithm that, given a seed, computes the output of G on this seed with high probability. That is, if we have such a PRG, then in the above argument, we can obtain $r = G(r_0)$ with high probability, which then allows us to recover x in the same way but *probabilistically*. Therefore, we get

$$\mathsf{rKt}(x) \leq \log(1/\delta) + O(\log T(n))\,,$$

as desired. □

4 Lower Bounds for Efficient Coding Theorems

4.1 Conditional Optimality of the Efficient Coding Theorem for rKt

We introduce the following hypothesis, which postulates the existence of a cryptographic PRG G of exponential security.

Hypothesis 21 (Cryptographic Exponential Time Hypotheses)**.** *For a constant $\gamma \in (0,1)$, we let γ-Crypto-ETH be the following statement. There is a function family $G = \{G_n\}_{n\geq 1}$, where $G_n\colon \{0,1\}^{\ell(n)} \to \{0,1\}^n$, such that the following holds:*

- *The seed length $\ell(n)$ can be computed in time polynomial in n and $(\log n)^{\omega(1)} \leq \ell(n) \leq n/2$.*
- *G is efficiently computable in the output length, i.e., there is a deterministic polynomial time algorithm A that, given 1^n and an input $x \in \{0,1\}^{\ell(n)}$, outputs $G_n(x)$ in time $\mathrm{poly}(n)$,*
- *G_n has security $2^{\gamma\cdot\ell(n)}$, i.e., for every probabilistic algorithm D that runs in time $O(2^{\gamma\cdot\ell(n)})$ on inputs of length n, there exists $n_0 \in \mathbb{N}$ such that, for every $n \geq n_0$,*

$$\left| \Pr_{\boldsymbol{D},\boldsymbol{x}\sim\{0,1\}^{\ell(n)}}[\boldsymbol{D}(G_n(\boldsymbol{x})) = 1] - \Pr_{\boldsymbol{D},\boldsymbol{y}\sim\{0,1\}^{n}}[\boldsymbol{D}(\boldsymbol{y}) = 1] \right| \leq 1/n.$$

Note that in γ-Crypto-ETH it is necessary to fool algorithms that run in time $2^{\gamma \cdot \ell(n)}$ in the *seed length* $\ell(n)$. We say that Crypto-ETH holds if γ-Crypto-ETH is true for some $\gamma > 0$, and that Crypto-SETH holds if γ-Crypto-ETH is true for every $\gamma \in (0,1)$.

The next result formalizes Theorem 3 from Section 1.2.2. It shows that if Crypto-ETH holds then the best parameter achieved by an *efficient* coding theorem for rKt is $(1+\Omega(1)) \cdot \log(1/\delta) + \mathsf{poly}(\log n)$. On the other hand, if the stronger Crypto-SETH hypothesis holds, then no efficient coding theorem for rKt achieves parameter $(2-o(1)) \cdot \log(1/\delta) + \mathsf{poly}(\log n)$.

Theorem 22 (Conditional rKt lower bound of efficient compression for poly-time samplers)**.** *Suppose that γ-Crypto-ETH holds for some constant $\gamma \in (0,1)$, and let $\ell(n)$ be the corresponding seed length function. There is a polynomial-time sampler $S = \{S_n\}_{n\geq 1}$ such that, for every $\varepsilon > 0$ and $C \geq 1$, the following holds. For every probabilistic polynomial time algorithm F, there is a sequence $\{y_n\}_{n\geq 1}$ of strings $y_n \in \{0,1\}^n$ such that:*

(i) Each string y_n is sampled by S_n with probability $\delta_n \geq 2^{-\ell(n)}$.

(ii) On some input parameter $\delta'_n \leq \delta_n$, F fails to output a probabilistic representation of y_n of certifying an rKt complexity

$$k_n(\delta'_n) = (1+\gamma-\varepsilon) \cdot \log(1/\delta'_n) + C(\log n)^C.$$

More precisely, there is a sequence $\{\delta'_n\}_{n\geq 1}$ with $(1/2)\delta_n \leq \delta'_n \leq \delta_n$ such that

$$\Pr_{\boldsymbol{F}}\left[\, \boldsymbol{F}(n, y_n, \delta'_n, \mathsf{code}(S)) \text{ outputs an rKt encoding of } y_n \text{ of complexity } \leq k_n(\delta'_n) \,\right] \to_n 0.$$

We remark that the *lower bound* on the seed length $\ell(n)$ present in γ-Crypto-ETH is a consequence of the $\mathsf{poly}(\log n)$ additive term in the definition of k_n in Theorem 22. This makes the negative result more robust. On the other hand, the *upper bound* on $\ell(n)$ is needed in our argument when considering an arbitrary $\gamma \in (0,1)$.

Proof. We implement the strategy described in Section 1.3.

Consider the function family $\{G_n\}_{n\geq 1}$ of PRG's witnessing that γ-Crypto-ETH holds, where $G_n : \{0,1\}^{\ell(n)} \to \{0,1\}^n$. We fix a sufficiently large length n (so that the forthcoming argument works), and for convenience we refer to G_n simply as G.

Let $Y = \{y_1, y_2, \ldots, y_m\}$ be the strings in the support of the PRG G and let $p_1, p_2, \ldots, p_m$ be their corresponding probabilities. Also, let δ_i be the unique number of the form $2^q/2^{\ell(n)}$, where $q = 0, 1, 2, \ldots, \ell(n)$, such that $(1/2)p_i < \delta_i \leq p_i$.

We define our sampler S_n as follows. S_n on input 1^n flips a coin $\ell(n)$ times obtaining a random string $z \in \{0,1\}^{\ell(n)}$ and next outputs $G(z)$. Thus, S_n runs in polynomial time, and for each $i \in [m]$, S_n generates y_i with probability p_i.

We must prove that for every choice of $\varepsilon > 0$ and $C \geq 1$, and for every probabilistic polynomial time algorithm F, there is a sequence $\{y_n\}_{n\geq 1}$ of strings $y_n \in \{0,1\}^n$ and a sequence $\{\delta'_n\}_{n\geq 1}$ of probability bounds δ'_n with the properties described above. Note that Item (i) is true by the definition of the sampler.

Suppose there is a probabilistic polynomial-time algorithm F that violates the assumption of the theorem. Then for every $i \in [m]$, there is a set E_{y_i} of valid probabilistic representations of y_i certifying rKt complexity at most $k(\delta_i)$ such that

$$\Pr_{\boldsymbol{F}}[\boldsymbol{F}(y_i, \delta_i) \in E_{y_i}] \geq \zeta,$$

for some constant $\zeta > 0$. We view the elements of E_{y_i} as the *good* fingerprints of y_i. The sets E_{y_i} are pairwise disjoint, because no element can be a good fingerprint of two strings. For each $i \in [m]$, let $x_{i,j}$, where $j \in [|E_{y_i}|]$, be the j-th element in E_{y_i}. Let

$$C \stackrel{\text{def}}{=} \{x_{i,j} \,|\, i \in [m] \text{ and } j \in [|E_{y_i}|]\},$$

i.e., C is the event that a fingerprint is good for some y_i. We define

$$p_{i,j} \stackrel{\text{def}}{=} \Pr_{\boldsymbol{F},\, \boldsymbol{z} \sim \{0,1\}^{\ell(n)}} [G(\boldsymbol{z}) = y_i \text{ and } \boldsymbol{F}(y_i, \delta_i) = x_{i,j}].$$

Note that we have $p_i \geq p_{i,j}$, for every i and j. Also, C has probability at least ζ because

$$\sum_{x_{i,j} \in C} p_{i,j} = \sum_{i \in [m]} \Pr_z[G(z) = y_i] \cdot \sum_{j \in [|E_i|]} \Pr_{F,z}[F(y_i, \delta_i) = x_{i,j} \mid G(z = y_i)] \geq \sum_{i \in [m]} \Pr_z[G(z) = y_i] \cdot \zeta = \zeta.$$

We say that a fingerprint $x_{i,j} \in C$ is *not short* if

$$|x_{i,j}| \geq \log(1/p_i) - 2\lceil \log(|x_{i,j}| + 1) \rceil - 2 - \log(2/\zeta).$$

Let $\mathcal{E} \stackrel{\text{def}}{=} \{x_{i,j} \mid i \in [\ell], j \in [|E_i|],\ x_{i,j} \text{ is not short}\}$ (i.e., $\mathcal{E}$ is the event that a fingerprint is good and not short).

Claim 23. $\Pr_{z, \rho_C}(\mathcal{E}) \geq \zeta/2$.

Proof of Claim 23. Let

$$C_{\text{pf}} \stackrel{\text{def}}{=} \left\{ x'_{i,j} \,\middle|\, i \in [m] \text{ and } j \in [|E_{y_i}|] \right\}$$

be a prefix-free encoding for the strings in C, where $x'_{i,j}$ is obtained from $x_{i,j}$ using the standard trick that inserts in front of $x_{i,j}$ its length written in binary with every bit doubled followed by 01 to delimit this addition from $x_{i,j}$. Note that $|x'_{i,j}| = |x_{i,j}| + 2\lceil \log(|x_{i,j}| + 1) \rceil + 2$.

Consider the set

$$C'_{\text{pf}} \stackrel{\text{def}}{=} \left\{ x'_{i,j} \in C_{\text{pf}} \,\middle|\, \left|x'_{i,j}\right| \leq \log(1/p_i) - \log(2/\zeta) \right\},$$

which is the prefix-free encoding of the complement (with respect to C) of the event $\mathcal{E}$.

Since C'_{pf} is a prefix-free set, we can use Kraft's inequality and we obtain

$$\begin{aligned}
1 &\geq \sum_{x'_{i,j} \in C'_{\text{pf}}} 2^{-|x'_{i,j}|} \\
&\geq \sum_{x'_{i,j} \in C'_{\text{pf}}} 2^{-\log(1/p_i) + \log(2/\zeta)} \\
&= \sum_{x'_{i,j} \in C'_{\text{pf}}} p_i \cdot (2/\zeta) \\
&\geq \sum_{x'_{i,j} \in C'_{\text{pf}}} p_{i,j} \cdot (2/\zeta),
\end{aligned}$$

which implies

$$\mu(C_{\mathrm{pf}}) = \sum_{x'_{i,j} \in C'_{\mathrm{pf}}} p_{i,j} \leq \zeta/2.$$

The event $\mathcal{E}$ has μ-probability $\mu(C) - \mu(C'_{\mathrm{pf}}) \geq \zeta - \zeta/2 = \zeta/2$. This ends the proof of Claim 23. □

Claim 23 means that with probability at least $\zeta/2$, over a random seed z and the internal randomness of F, $G(z)$ outputs some y_i and $F(y_i, \delta_i)$ gives a valid probabilistic representation x of y_i certifying rKt complexity $|x| + \log t_x$ at most $k(\delta_i)$ and its length $|x|$ is at least

$$s \stackrel{\mathrm{def}}{=} \log(1/p_i) - \log(2/\zeta) - (2\lceil \log(|x|+1) \rceil + 2).$$

We can implement a distinguisher D that, given $y \in \{0,1\}^n$, does the following.

1. For every
$$\delta'_q := 2^q / 2^{\ell(n)},$$
where $q = 0, 1, 2, \ldots, \ell(n)$, D runs $F\left(y, \delta'_q\right)$ (using the same randomness of F for all the δ'_q), and obtain a collection of encodings $S := \{x_q\}_q$.

2. D outputs 1 if both of the following conditions hold for *at least one* $x_q \in S$:
   - $|x_q| \leq k(\delta'_q)$.
   - x_q can be decoded (probabilistically) in $2^{(\gamma - \varepsilon/2)\cdot \ell(n)}$ steps and the decoded string is equal to y.

   Again, for decoding, we can use the same randomness for all the x_q.

It is easy to see that the running time of D is $2^{(\gamma - \Omega(1))\cdot \ell(n)}$. Next, we argue that D is a distinguisher for G.

Claim 24. *We have*

$$\Pr_{\boldsymbol{D},\, \boldsymbol{z} \sim \{0,1\}^{\ell(n)}} [\boldsymbol{D}(G(\boldsymbol{z})) = 1] \geq \zeta/3 = \Omega(1),$$

and

$$\Pr_{\boldsymbol{D},\, \boldsymbol{y} \sim \{0,1\}^{n}} [\boldsymbol{D}(y) = 1] = o(1).$$

Proof of Claim 24. For the first item, note that from the discussion above, we have that with probability at least $\zeta/2$ (over a random z and the internal randomness of F), $G(z)$ outputs some y_i and for $\delta'_q = \delta_i$, $F(y_i, \delta'_q)$ will output some probabilistic representation x_q of y_i certifying rKt complexity $|x_q| + \log t_{x_q}$ at most $k(\delta'_q)$ and length $|x_q|$ at least

$$\begin{aligned} s &:= \log(1/p_i) - \log(2/\zeta) - (2\lceil \log(|x_q|+1) \rceil + 2) \\ &\geq \log(1/2\delta_i) - \log(2/\zeta) - (2\lceil \log(|x_q|+1) \rceil + 2) \\ &\geq \log(1/\delta'_q) - 3\log(n). \end{aligned}$$

Whenever we have such an encoding, we can decode probabilistically in time

$$t_{x_q} \leq 2^{k(\delta'_q) - s} = 2^{(1+\gamma-\varepsilon)\log(1/\delta'_q) + C(\log n)^C - s} \leq 2^{(\gamma - \varepsilon/2)\cdot \ell(n)},$$

(we have used $\ell(n) = (\log n)^{\omega(1)}$) and with error probability at most $1/3$. By the definition of D, we conclude that D outputs 1 with probability at least $(\zeta/2) \cdot (2/3) = \zeta/3$.

We now show the second item. Fix any $\delta'_q \geq 1/2^{\ell(n)}$, where $q = 0, 1, \ldots, \ell(n)$. We will show that

$$\Pr_{\boldsymbol{F}, \mathbf{Dec}, \boldsymbol{y} \sim \{0,1\}^n} \left[\left| \boldsymbol{F}(\boldsymbol{y}, \delta'_q) \right| \leq k(\delta'_q) \text{ and } \mathbf{Dec}\big(\boldsymbol{F}(\boldsymbol{y}, \delta'_q)\big) = \boldsymbol{y} \right] = o\left(\frac{1}{\ell(n)}\right). \tag{3}$$

Then the second item follows from a union bound. For the sake of contradiction, suppose Equation (3) is false. Then by averaging, there exist circuits F' and Dec', which are obtained by fixing the randomness of $\boldsymbol{F}$ and $\mathbf{Dec}$ and by hard-wiring δ'_q, such that

$$\Pr_{\boldsymbol{y} \sim \{0,1\}^n} \left[\left| F'(\boldsymbol{y}) \right| \leq k(\delta'_q) \text{ and } \mathsf{Dec}'\left(F'(\boldsymbol{y})\right) = \boldsymbol{y} \right] = \Omega\left(\frac{1}{\ell(n)}\right).$$

However, this is not possible, because by a counting argument, the probability on the left side is at most

$$\frac{2^{k(\delta'_q)}}{2^n} = \frac{2^{(1+\gamma-\varepsilon)\log(1/\delta'_q) + C(\log n)^C}}{2^n} \leq \frac{2^{(1+\gamma-\varepsilon)\ell(n) + C(\log n)^C}}{2^n} \leq \frac{2^{(2-\varepsilon)\ell(n) + C(\log n)^C}}{2^n} \leq 2^{-\Omega(n)},$$

where we used that $\gamma < 1$, $\ell(n) \leq n/2$, and $\delta'_q \geq 2^{-\ell(n)}$. This completes the proof of Claim 24. □

The theorem now follows from Claim 24. □

4.2 Fine-Grained Complexity of Coding Algorithms for Poly-Time Samplers

We prove the results stated informally in Theorem 4. The rKt-complexity of a string adds together the length of a compressed codeword and the logarithm of the time it takes to decompress the codeword. In 2-sided-rKt complexity we also consider the time to compress the string.

Definition 25 (2-sided-rKt)**.** *A sampler A admits coding with 2-sided-rKt_ε complexity bounded by Γ if there exists a pair of probabilistic Turing machines* (Compress, Decompress) *such that for all n, all $y \in \{0,1\}^n$, and all $\delta > 0$ satisfying $\mathbf{Pr}[A(1^n) = y] \geq \delta$, it holds with probability $1 - \varepsilon$ (over the randomness of* Compress *and the randomness of* Decompress*) that*

- $\mathsf{Compress}(y, \delta)$ *outputs a string x in t_C steps,*
- $\mathsf{Decompress}(x)$ *outputs y in t_D steps, and*
- $|x| + \log(t_C + t_D) \leq \Gamma$.

Here, Γ is a function of n and δ. In case $\varepsilon < 1/3$, we drop the subscript in the notation rKt_ε.

The following result follows from Theorem 11 in the same way as Corollary 14.

Corollary 26 (Formal statement of Theorem 4, (a))**.** *Let S be a polynomial-time sampler. Then, for every $\varepsilon > 0$, S admits coding with 2-sided-rKt_ε complexity bounded by $2\log(1/\delta) + O(\alpha(|x|, 1/\delta, \varepsilon))$, where the constant hidden in $O(\cdot)$ depends on A and $\alpha(|x|, 1/\delta, \varepsilon)$ is the function from Equation (1).*

Theorem 27 (Formal statement of Theorem 4, (b))**.** *Assume* γ-Crypto-ETH *holds for some* $\gamma \in (0,1)$*. Then there is a polynomial-time sampler* S *that, for any* $\varepsilon > 0$ *and any* $C > 0$*, does not admit coding with* 2*-sided-*$\mathsf{rKt}_{1/7}$ *complexity bounded by* $k_n(\delta) := (1+\gamma-\varepsilon)\log(1/\delta)+C(\log n)^C$*.*

Proof. The proof is similar to the proof of Theorem 22, but there are differences caused by the fact that in 2-sided-rKt, t_C and t_D (i.e., the runtimes of compression and decompression) are random variables, whereas in rKt, t_p is a fixed value (being the maximum decompression time over all the probabilistic branches).

The PRG G, the set $Y = \{y_1, \ldots, y_m\}$, and the sampler S are exactly like in the proof of Theorem 22.

Suppose there is a pair $(\mathsf{Compress}, \mathsf{Decompress})$ of probabilistic Turing machines that violates the conclusion of the theorem for the sampler S, i.e., the pair certifies that the sampler S has coding with 2-sided-$\mathsf{rKt}_{1/7}$ complexity bounded by $(1+\gamma-\varepsilon)\log(1/\delta) + C(\log n)^C$ for some $\varepsilon > 0$ and $C > 0$. We show that this assumption implies the existence of a distinguisher D that breaks the security of G stipulated by γ-Crypto-ETH, and this contradiction proves the theorem.

Since for every $i \in [m]$, $\mathbf{Pr}_{\rho_C,\rho_D}[\mathsf{Decompress}(\mathsf{Compress}(y_i,\delta_i)) = y_i] \geq (1-1/7)$, where the probability is over the random coins ρ_C of Compress and ρ_D of Decompress, a simple argument that considers separately the random coins of the two algorithms implies that that for every $i \in [m]$ there is a set E_{y_i} of strings (which we view as the *good* fingerprints of y_i) such that

(a) $\mathbf{Pr}_{\rho_C}[\mathsf{Compress}\,(y_i,\delta_i) \in E_{y_i}] \geq 5/7$, and

(b) for every $x \in E_{y_i}$, $\mathbf{Pr}_{\rho_D}[\mathsf{Decompress}(x) = y_i] > 1/2$.

By (b), the sets $E_{y_i}, i \in [m]$ are pairwise disjoint. For each $i \in [m]$ and $j \in [|E_{y_i}|]$, let $x_{i,j}$ be the j-th element in E_{y_i}. Let $C \stackrel{\text{def}}{=} \{x_{i,j} \mid i \in [\ell] \text{ and } j \in [|E_{y_i}|]\}$ (i.e., C is the event that a fingerprint is good for some y_i).

Let $\zeta = 5/7$. Taking into account (a), it follows, exactly like in Theorem 22, that $\mathbf{Pr}_{z,\rho_C}(C)$ is at least ζ. As before, we say that a fingerprint $x_{i,j} \in C$ is *not short* if $|x_{i,j}| \geq \log(1/p_i) - 2\lceil\log(|x_{i,j}|+1)\rceil - 2 - \log(2/\zeta)$. Let $\mathcal{E} \stackrel{\text{def}}{=} \{x_{i,j} \mid i \in [m], j \in [|E_i|], x_{i,j} \text{ is not short}\}$ (i.e., $\mathcal{E}$ is the event that a fingerprint is good and not short).

Claim 28. $\mathbf{Pr}_{z,\rho_C}[\mathcal{E}] \geq \zeta/2$*.*

Proof of Claim 28. Identical to the proof of Claim 23. □

Claim 28 means that, conditioned on the event $\mathcal{E}$ (so with probability of (z,ρ_C) at least $\zeta/2$), $G(z)$ outputs some y_i and $\mathsf{Compress}(y_i,\delta_i)$ outputs a good fingerprint x that is not short.

We implement a distinguisher D that, on input $y \in \{0,1\}^n$, does the following.

For every $q = 0, 1, \ldots, \ell(n)$:

1. Let $\delta'_q := 2^q/2^{\ell(n)}$.
2. D runs $\mathsf{Compress}\left(y, \delta'_q\right)$ (using the same randomness ρ_C of Compress for all the δ'_q), which outputs a fingerprint x_q.
3. D runs Decompress on input x_q, which outputs y'. As above, D uses the same randomness ρ_D of Decompress for all the x_q.

4. D outputs 1 (and exits the for loop) if

   - $y' = y$ (i.e., $\mathsf{Decompress}(\mathsf{Compress}(y, \delta'_q)) = y$), and
   - $t_C + t_D$ is at most $2^{(\gamma-\varepsilon/2)\cdot\ell(n)}$, where t_C is the number of steps executed by $\mathsf{Compress}$ on input (y, δ'_q) and t_D is the number of steps executed by $\mathsf{Decompress}$ on input x_q.

If the for loop ends without the conditions in Item 4 being satisfied at any iteration, D outputs 0.

Since randomness is re-used at every iteration, overall, D is using randomness (ρ_C, ρ_D).

We now argue that D is a distinguisher for G with runtime bounded by $2^{(\gamma-\Omega(1))\cdot\ell(n)}$, which yields the desired contradiction and finishes the proof.

First, since the execution of D consists of $\ell(n)+1$ iterations and it can be arranged that each iteration takes at most $2^{(\gamma-\varepsilon/2)\ell(n)}$ steps (by halting the iteration when $t_C + t_D$ gets larger than this value), the runtime of D is, as claimed, bounded by $2^{(\gamma-\Omega(1))\cdot\ell(n)}$. Next we show that D distinguishes the distributions $G(U_{\ell(n)})$ and U_n.

Claim 29. *We have*

$$\mathop{\mathbf{Pr}}_{\rho_C,\rho_D,z\sim\{0,1\}^{\ell(n)}}[D(G(z)) = 1] \geq \Omega(1),$$

and

$$\mathop{\mathbf{Pr}}_{\rho_C,\rho_D,y\sim\{0,1\}^{n}}[D(y) = 1] < o(1).$$

Proof of Claim 29. We start with the first inequality. By the discussion above, conditioned on the event $\mathcal{E}$ (which by Claim 28 has probability of (z, ρ_C) at least $\zeta/2$), $G(z)$ outputs some y_i and for $\delta'_q = \delta_i$, $\mathsf{Compress}(y_i, \delta'_q)$ outputs a good fingerprint x_q that is not short. Since x_q is a good fingerprint, conditioned on an event $\mathcal{E}' \subseteq \mathcal{E}$, $\mathsf{Decompress}$ on input x_q reconstructs y_i. By (b), $\mathcal{E}'$ has probability of (z, ρ_C, ρ_D) at least $1/2 \cdot \mathbf{Pr}[\mathcal{E}] \geq \zeta/4$. Recall that $(\mathsf{Compress}, \mathsf{Decompress})$ are assumed to certify that the sampler A is compressible with 2-sided-$\mathsf{rKt}_{1/7}$ complexity bounded by $(1+\gamma-\varepsilon)\log(1/\delta)$. This means that conditioned by an event $\mathcal{V}$ that has probability of (z, ρ_C, ρ_D) at least $1 - 1/7$, it holds that

$$|x_q| + \log(t_C + t_D) \leq k_n(\delta'_q).$$

Now, conditioned by $\mathcal{E}'$, x_q is not short, and so (like in Theorem 22)

$$|x_q| \geq s := \log(1/p_i) - \log(2/\zeta) - (2\lceil\log(|x_q|+1)\rceil + 2) \geq \log(1/\delta'_q) - 3\log(n).$$

By combining the above two inequalities, it follows that conditioned by $\mathcal{E}' \cap \mathcal{V}$, which has probability of (z, ρ_C, ρ_D) at least $\zeta/4 - 1/7$, it holds that

$$t_C + t_D \leq 2^{k_n(\delta'_q)-s} \leq 2^{(\gamma-\varepsilon/2)\cdot\ell(n)}.$$

By the definition of D, we conclude that D outputs 1 with probability greater than $\zeta/4 - 1/7 = \Omega(1)$ (recall that $\zeta = 5/7$).

The second inequality is shown in the same way as the second inequality of Claim 24. □

Thus, D is a distinguisher that contradicts that the PRG G has the security stipulated by γ-Crypto-ETH, and this finishes the proof of Theorem 27. □

5 A Coding Theorem for $\mathsf{pK^t}$ Complexity and Its Consequences

5.1 Optimal Coding Theorem for $\mathsf{pK^t}$

In this section, we prove our optimal coding theorem for pK^t.

Theorem 30 (Reminder of Theorem 5)**.** *Suppose there is a randomized algorithm A for sampling strings such that $A(1^n)$ runs in time $T(n)$ and outputs a string $x \in \{0,1\}^n$ with probability at least $\delta > 0$. Then*

$$\mathsf{pK}^t(x) \;=\; \log(1/\delta) + O(\log T(n))\,,$$

where $t(n) = \mathrm{poly}(T(n))$ and the constant behind the $O(\cdot)$ depends on $|A|$ and is independent of the remaining parameters.

For a function $H\colon \{0,1\}^\ell \to \{0,1\}^T$, we will sometimes identify it with a string $H \in \{0,1\}^{2^\ell \cdot T}$.

Lemma 31. *For any $T \in \mathbb{N}$ and $\delta \in [0,1]$, there exists a family of functions*

$$\mathcal{H} = \left\{H_w\colon \{0,1\}^\ell \to \{0,1\}^T\right\}_{w\in\{0,1\}^k}$$

where $k = \mathrm{poly}(T)$ and $\ell = \log(1/\delta) + O(1)$ such that the following holds. Let $M\colon \{0,1\}^T \to \{0,1\}^*$ be a function computable in time T and let $x \in \mathrm{Range}(M)$ be such that

$$\Pr_{z\sim\{0,1\}^T}[M(z) = x] \geq \delta.$$

It holds that

$$\Pr_{w\sim\{0,1\}^k}\left[\exists\, v \in \{0,1\}^\ell \text{ such that } M(H_w(v)) = x\right] \geq 2/3.$$

Moreover, given $w \in \{0,1\}^k$ and $v \in \{0,1\}^\ell$, $H_w(v)$ can be computed in time $\mathrm{poly}(T)$.

Proof. Consider arbitrary M and x. Let us call a function $H\colon \{0,1\}^\ell \to \{0,1\}^T$ *good* (with respect to M and x) if there exists some $v \in \{0,1\}^\ell$ such that $M(H(v)) = x$. First note that a random $H \in \{0,1\}^{2^\ell \cdot T}$ is good with high probability. In particular, the probability that a random H is not good is at most $(1-\delta)^{2^\ell}$, which is at most $o(1)$ for our choice of ℓ.

Next, we show that checking whether a given H is good can be implemented as a constant-depth circuit. More specifically, note that given M and x, and using oracle access to H, checking whether there exists some $v \in \{0,1\}^\ell$ such that $M(H(v)) = x$ can be done in NP. By the standard connection between the computation of an oracle-taking machine in PH and constant-depth circuits (see e.g., [RST15]), we get that there is an AC^0 circuit of size at most $2^{\mathrm{poly}(T)}$ that takes H as input and checks whether it is good.

Now we will try to generate a good H using a pseudorandom generator for AC^0 circuits. It is known that there is a pseudorandom generator $G\colon \{0,1\}^r \to \{0,1\}^N$ that $(1/10)$-fools AC^0 circuits on N bits of size at most s, where the seed length r is at most $\mathrm{polylog}(Ns)$. Moreover, given $z \in \{0,1\}^r$ and $i \in [N]$, the i-th bit of $G(z)$ can be computed in time $\mathrm{poly}(r)$ (see e.g., [Nis91, TX13, Tal17, ST19]). Let $N := 2^\ell \cdot T$ and let $s := 2^{\mathrm{poly}(T)}$. We get a generator that takes $w \in \{0,1\}^{\mathrm{poly}(T)}$ and outputs a function $H_w \in \{0,1\}^{2^\ell \cdot T}$, such that with probability at least $1 - o(1) - 1/10 > 2/3$ over w, H_w is good. Finally, note that given w and v, we can compute $H_w(v)$ in time $\mathrm{poly}(T)$ because we can compute any single output bit of the generator in time $\mathrm{poly}(T)$. □

We are now ready to show Theorem 30.

Proof of Theorem 30. Let us view $M := A(1^n)$ as a function that takes $T := T(n)$ random bits and outputs $x \in \{0,1\}^n$ with probability at least δ.

By Lemma 31, for at least 2/3 of $w \in \{0,1\}^{\mathrm{poly}(T)}$, we get a function $H_w \colon \{0,1\}^\ell \to \{0,1\}^T$, where $\ell = \log(1/\delta) + O(\log T)$, with the property that there is some "good" $v \in \{0,1\}^\ell$ such that $M(H_w(v)) = x$. Also, given w and v, $H_w(v)$ can be computed in time $\mathrm{poly}(T)$. This means that for at least 2/3 of $w \in \{0,1\}^{\mathrm{poly}(T)}$, there is some advice string $\alpha \in \{0,1\}^{\log(1/\delta)+O(\log T)}$, which encodes the number T, the code for $A(1^n)$, the code for computing H_w using w, and some good v (which could depend on w), such that using α together with w we can recover x in time $\mathrm{poly}(T)$. This implies that

$$\mathsf{pK}^t(x) \le \log(1/\delta) + O(\log T),$$

where $t \colon \mathbb{N} \to \mathbb{N}$ is such that $t(n) = \mathrm{poly}(T(n))$. □

5.2 Application: An Unconditional Version of Antunes-Fortnow

In this subsection, we prove an unconditional version of a result in [AF09], which is stated in Theorem 6. We start with some useful lemmas.

5.2.1 Useful Lemmas

The following lemma lower bounds the pK^t complexity of a string by its (time-unbounded) Kolmogorov complexity.

Lemma 32. *For every computable time bound $t \colon \mathbb{N} \to \mathbb{N}$, there is a constant $b > 0$ (which depends only on t) such that for every $x \in \{0,1\}^*$,*

$$\mathsf{K}(x) \le \mathsf{pK}^t(x) + b\log(|x|).$$

Proof. Recall the following source coding theorem for (time-unbounded) prefix-free Kolmogorov complexity. There is a universal constant $c > 0$ such that, if there exists a randomized algorithm D that uses randomness chosen from a prefix-free set and that generates x with probability δ, then

$$\mathsf{K}(x \mid D) \le \log(1/\delta) + c.$$

Fix a computable function t and a string x. Given the integers $n := |x|$ and $k := \mathsf{pK}^t(x)$, consider the algorithm D that randomly picks $w \in \{0,1\}^{t(n)}$ and $\mathcal{M} \in \{0,1\}^k$, and then outputs whatever $\mathcal{M}(w)$ outputs within $t(n)$ steps. Note that the random strings used by D all have the same length and thus they form a prefix-free set as required by the above coding theorem. By the definition of pK^t, D will output x with probability at least $\frac{2}{3\cdot 2^k}$. Consequently, using the above source coding theorem and the fact that D can be encoded using $O_t(\log(|x|))$ bits (because $k \le |x| + O(1)$), we obtain

$$\mathsf{K}(x) \le k + b\log(|x|),$$

where $b > c$ is some constant that depends only on t. □

For technical reasons, we introduce the following measure.

Definition 33. *For a time bound $t\colon \mathbb{N} \to \mathbb{N}$ and $x \in \{0,1\}^*$, define*

$$\mathsf{pK}^t_*(x) \stackrel{\text{def}}{=} \mathsf{pK}^t(x) + b\log(|x|),$$

where $b > 0$ is the constant from Lemma 32.

We define the following (semi-)distribution which will be a key notion used in the proofs later.

Definition 34. *For a time bound $t\colon \mathbb{N} \to \mathbb{N}$, let m^t be the distribution over $\{0,1\}^*$ defined as*

$$m^t(x) \stackrel{\text{def}}{=} 2^{-\mathsf{pK}^t_*(x)}.$$

Equivalence between polynomial time on m^{poly}-average and worst-case time using pK^t.

Lemma 35. *For any algorithm A and any computable time bound $t\colon \mathbb{N} \to \mathbb{N}$, the following are equivalent.*

1. *A runs in polynomial time on average with respect to m^t.*
2. *The running time of A is bounded by $2^{O\left(\mathsf{pK}^t_*(x) - \mathsf{K}(x) + \log(|x|)\right)}$ for every input x.*

Proof. The proof follows closely that of [AFV03, Theorem 4]. Let $t_A(x)$ denote the running time of A on input x.

($2 \implies 1$). Let $c > 0$ be a constant such that $t_A(x) \le 2^{c\cdot\left(\mathsf{pK}^t_*(x) - \mathsf{K}(x) + \log(|x|)\right)}$. We have

$$\begin{aligned}\sum_{x\in\{0,1\}^*} \frac{t_A(x)^{1/c}}{|x|} \cdot m^t(x) &\le \sum_x \frac{2^{\mathsf{pK}^t_*(x) - \mathsf{K}(x) + \log(|x|)}}{|x|} \cdot 2^{-\mathsf{pK}^t_*(x)} \\ &\le \sum_x 2^{-\mathsf{K}(x)} < 1,\end{aligned}$$

where the last line follows from Kraft's inequality.

($1 \implies 2$). For $n, i, j \in \mathbb{N}$ with $i, j \le n^2$, define

$$S_{i,j,n} \stackrel{\text{def}}{=} \left\{x \in \{0,1\}^n \mid 2^i \le t_A(x) \le 2^{i+1} \text{ and } \mathsf{pK}^t_*(x) = j\right\}.$$

Let r be such that $2^r \le |S_{i,j,n}| \le 2^{r+1}$. We claim that for every $x \in S_{i,j,n}$,

$$\mathsf{K}(x) \le r + O(\log n). \tag{4}$$

To see this, note that given i, j, n, we can first enumerate all the elements in $S_{i,j,n}$, which can be done since t is computable, and then using additional $r+1$ bits, we can specify x in $S_{i,j,n}$.

Now, fix $i, j \le n^2$, and let r be such that $2^r \le |S_{i,j,n}| \le 2^{r+1}$. Then by assumption and by the definition of $S_{i,j,n}$, we have for some constants $\varepsilon, c > 0$ (which may depend on m^t and hence the time bound function t),

$$c > \sum_{x\in S_{i,j,n}} \frac{t_A(x)^\varepsilon}{|x|} \cdot m^t(x) \ge 2^r \cdot \frac{2^{\varepsilon\cdot i}}{n} \cdot 2^{-j} = 2^{\varepsilon\cdot i + r - j - \log(n)},$$

which yields

$$\varepsilon \cdot i + r - j - \log(n) < c.$$

By Equation (4), this implies that for every $x \in S_{i,j,n}$,

$$\varepsilon \cdot i \leq \mathsf{pK}^t_*(x) - \mathsf{K}(x) + O(\log n).$$

Therefore, we have that for every $x \in S_{i,j,n}$,

$$t_A(x) \leq 2^{i+1} \leq 2^{\varepsilon^{-1} \cdot \left(\mathsf{pK}^t_*(x) - \mathsf{K}(x) + O(\log n)\right)} = 2^{O\left(\mathsf{pK}^t_*(x) - \mathsf{K}(x) + \log(|x|)\right)},$$

as desired. □

A P-samplable distribution that dominates m^{poly}.

Lemma 36. *For any polynomial p, there is a P-samplable distribution $\mathcal{D}$ that dominates m^p.*

Proof. First note that there is a universal constant $d > 0$ such that for every $x \in \{0,1\}^n$, $\mathsf{pK}^p(x) \leq n + d$. For a polynomial p, we define a distribution $\mathcal{D}$ over $\{0,1\}^*$ as follow:

1. Pick n with probability $\frac{1}{n \cdot (n+1)}$.
2. Pick uniformly at random $j \in [n + d]$.
3. Pick uniformly at random $w \in \{0,1\}^{p(n)}$.
4. pick uniformly at random $\mathcal{M} \in \{0,1\}^j$.
5. Run $\mathcal{M}(w)$ for $p(n)$ steps and output whatever is on its output tape.

By the definition of pK^t, for every $x \in \{0,1\}^n$, $\mathcal{D}$ outputs x with probability at least

$$\frac{1}{n \cdot (n+1)} \cdot \frac{1}{n+d} \cdot \frac{2}{3} \cdot 2^{-\mathsf{pK}^p(x)} \geq \frac{m^p(x)}{|x|^{O(1)}},$$

as desired. □

m^{poly} dominates P-samplable distributions.

Lemma 37. *For every P-samplable distribution $\mathcal{D}$, there is a polynomial p such that m^p dominates $\mathcal{D}$.*

Proof. Let $M_{\mathcal{D}}$ be a probabilistic algorithm and let q be the polynomial such that $M_{\mathcal{D}}$ outputs x with probability $\mathcal{D}(x)$ within $q(|x|)$ steps. Consider any $n \in \mathbb{N}$. Let M be a sampler that, on input 1^n, runs $M_{\mathcal{D}}$ for $q(n)$ steps and outputs whatever is on its output tape. It is easy to see that M runs in time $\mathrm{poly}(q(n))$. Also, for every $x \in \{0,1\}^n$, $M(1^n)$ outputs x with probability at least $\mathcal{D}(x)$. By the coding theorem for pK^t (Theorem 5), we have, for some polynomial p (which depends on the running time of M),

$$\mathsf{pK}^p_*(x) \leq \log(1/\mathcal{D}(x)) + O(\log n),$$

which implies

$$m^p(x) = 2^{-\mathsf{pK}^p_*(x)} \geq \frac{\mathcal{D}(x)}{|x|^{O(1)}}.$$

This completes the proof. □

5.2.2 Putting It All Together

Theorem 38 (Reminder of Theorem 6)**.** *The following are equivalent for every language L.*

1. *For every P-samplable distributions $\mathcal{D}$, L can be solved in polynomial time on average with respect to $\mathcal{D}$.*
2. *For every polynomial p, there exist a constant $b > 0$ and an algorithm computing L whose running time is bounded by $2^{O(\mathsf{pK}^p(x)-\mathsf{K}(x)+b\cdot\log(|x|))}$ for every input x.*

Proof.
$(1 \implies 2)$. Let p be any polynomial. By Lemma 36, there exists a P-samplable distribution $\mathcal{D}$ that dominates m^p. By assumption, there is an algorithm A that computes L and runs in polynomial time on average with respect to $\mathcal{D}$. Then by Fact 10, A also runs in average polynomial time with respect to m^p. Finally, by Lemma 35, we have that the running time of A is bounded by $2^{O\left(\mathsf{pK}^p_*(x)-\mathsf{K}(x)+\log(|x|)\right)}$ for every input x, as desired.

$(2 \implies 1)$. Let $\mathcal{D}$ be any P-samplable distribution. By Lemma 37, there is a polynomial p such that m^p dominates $\mathcal{D}$. By assumption, there is an algorithm A that computes L such that on input x, A runs in time at most

$$2^{O(\mathsf{pK}^p(x)-\mathsf{K}(x)+b\cdot\log(|x|))} \leq 2^{O\left(\mathsf{pK}^p_*(x)-\mathsf{K}(x)+\log(|x|)\right)}.$$

Then by Lemma 35, A runs in polynomial time on average with respect to m^p, which by Fact 10 implies that A also runs in average polynomial time with respect to $\mathcal{D}$, as desired. □

6 Concluding Remarks and Open Problems

Our results indicate that Theorem 1 might be optimal among *efficient* coding theorems for rKt, i.e., those that efficiently produce representations matching the existential bounds. In the case of pK^t, the corresponding coding theorem (Theorem 5) is optimal. We have described a concrete application of Theorem 5 (Theorem 6). A second application appears in [GKLO22]. In both cases, achieving an optimal dependence on the probability parameter δ is critical, and for this reason, the result from [LO21] is not sufficient.

Naturally, we would like to understand the possibility of establishing an *unconditional* coding theorem for rKt with an optimal dependence on the probability parameter δ. While the validity of $\mathsf{Crypto\text{-}ETH}$ implies that no *efficient* coding theorem with this property exist, we have an *existential* coding theorem of this form under a derandomization assumption (Proposition 2). In the case of K^t complexity, it is known that an unconditional coding theorem with optimal dependence on δ implies that $\mathsf{EXP} \neq \mathsf{BPP}$ (see [Lee06, Theorem 5.3.4]). However, the techniques behind this connection do not seem to lead to an interesting consequence in the case of rKt and rK^t. Consequently, an optimal coding theorem for rKt might be within the reach of existing techniques.

It would also be interesting to establish Theorem 3 under a weaker assumption, or to refute $\mathsf{Crypto\text{-}SETH}$. A related question is the possibility of basing $\mathsf{Crypto\text{-}ETH}$ on the existence of one-way functions of exponential hardness. Existing reductions are not strong enough to provide an equivalence between one-way functions and cryptographic pseudorandomness in the exponential regime (see [VZ13, HRV13]).

Finally, are there more applications of pK^t complexity and of Theorem 5? Since this coding theorem is both optimal and unconditional, we expect more applications to follow.

Acknowledgements. We are grateful to Bruno Bauwens for discussions and useful insights. M. Zimand was supported in part by the National Science Foundation through grant CCF 1811729. Z. Lu and I.C. Oliveira received support from the Royal Society University Research Fellowship URF\R1\191059 and from the EPSRC New Horizons Grant EP/V048201/1.

References

[Aar14] Scott Aaronson. The equivalence of sampling and searching. *Theory Comput. Syst.*, 55(2):281–298, 2014.

[AF09] Luis Filipe Coelho Antunes and Lance Fortnow. Worst-case running times for average-case algorithms. In *Conference on Computational Complexity* (CCC), pages 298–303, 2009.

[AFV03] Luis Antunes, Lance Fortnow, and N. V. Vinodchandran. Using depth to capture average-case complexity. In *Fundamentals of Computation Theory* (FCT), pages 303–310, 2003.

[All92] Eric Allender. Applications of time-bounded Kolmogorov complexity in complexity theory. In *Kolmogorov complexity and computational complexity*, pages 4–22. Springer, 1992.

[All01] Eric Allender. When worlds collide: Derandomization, lower bounds, and Kolmogorov complexity. In *International Conference on Foundations of Software Technology and Theoretical Computer Science* (FSTTCS), pages 1–15. Springer, 2001.

[All17] Eric Allender. The complexity of complexity. In *Computability and Complexity*, pages 79–94. Springer, 2017.

[BCGL92] Shai Ben-David, Benny Chor, Oded Goldreich, and Michael Luby. On the theory of average case complexity. *J. Comput. Syst. Sci.*, 44(2):193–219, 1992.

[BFL01] Harry Buhrman, Lance Fortnow, and Sophie Laplante. Resource-bounded Kolmogorov complexity revisited. *SIAM J. Comput.*, 31(3):887–905, 2001.

[BLvM05] Harry Buhrman, Troy Lee, and Dieter van Melkebeek. Language compression and pseudorandom generators. *Comput. Complex.*, 14(3):228–255, 2005.

[BT06] Andrej Bogdanov and Luca Trevisan. Average-case complexity. *Found. Trends Theor. Comput. Sci.*, 2(1), 2006.

[BZ19] Bruno Bauwens and Marius Zimand. Universal almost optimal compression and Slepian-Wolf coding in probabilistic polynomial time. *CoRR*, abs/1911.04268, 2019.

[CGLQ20] Kai-Min Chung, Siyao Guo, Qipeng Liu, and Luowen Qian. Tight quantum time-space tradeoffs for function inversion. In *Symposium on Foundations of Computer Science* (FOCS), pages 673–684, 2020.

[CKL+22] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. *CoRR*, abs/2203.00671, 2022.

[CRVW02] M. R. Capalbo, O. Reingold, S. P. Vadhan, and A. Wigderson. Randomness conductors and constant-degree lossless expanders. In *STOC*, pages 659–668, 2002.

[DTT10] Anindya De, Luca Trevisan, and Madhur Tulsiani. Time space tradeoffs for attacks against one-way functions and PRGs. In *Annual International Cryptology Conference* (CRYPTO), pages 649–665, 2010.

[FN99] Amos Fiat and Moni Naor. Rigorous time/space trade-offs for inverting functions. *SIAM J. Comput.*, 29(3):790–803, 1999.

[For04] Lance Fortnow. Kolmogorov complexity and computational complexity. *Complexity of Computations and Proofs. Quaderni di Matematica*, 13, 2004.

[GKLO22] Halley Goldberg, Valentine Kabanets, Zhenjian Lu, and Igor C. Oliveira. Probabilistic Kolmogorov complexity with applications to average-case complexity. Preprint, 2022.

[GUV09] Venkatesan Guruswami, Christopher Umans, and Salil P. Vadhan. Unbalanced expanders and randomness extractors from Parvaresh-Vardy codes. *J. ACM*, 56(4):20:1–20:34, 2009.

[Hir18] Shuichi Hirahara. Non-black-box worst-case to average-case reductions within NP. In *Symposium on Foundations of Computer Science* (FOCS), pages 247–258, 2018.

[Hir21] Shuichi Hirahara. Average-case hardness of NP from exponential worst-case hardness assumptions. In *Symposium on Theory of Computing* (STOC), pages 292–302, 2021.

[HLW06] S. Hoory, N. Linial, and A. Wigderson. Expander graphs and their applications. *Bull. Amer. Math. Soc.*, 43:439–561, 2006.

[HRV13] Iftach Haitner, Omer Reingold, and Salil P. Vadhan. Efficiency improvements in constructing pseudorandom generators from one-way functions. *SIAM J. Comput.*, 42(3):1405–1430, 2013.

[IP01] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

[Kra21] Jan Krajíček. Information in propositional proofs and algorithmic proof search. *The Journal of Symbolic Logic*, page 1–22, 2021.

[Lee06] Troy Lee. *Kolmogorov complexity and formula lower bounds*. PhD thesis, University of Amsterdam, 2006.

[Lev84] Leonid A. Levin. Randomness conservation inequalities; information and independence in mathematical theories. *Information and Control*, 61(1):15–37, 1984.

[Lev86] Leonid A. Levin. Average case complete problems. *SIAM J. Comput.*, 15(1):285–286, 1986.

[LO21] Zhenjian Lu and Igor C. Oliveira. An efficient coding theorem via probabilistic representations and its applications. In *International Colloquium on Automata, Languages, and Programming* (ICALP), pages 94:1–94:20, 2021.

[LOS21] Zhenjian Lu, Igor C. Oliveira, and Rahul Santhanam. Pseudodeterministic algorithms and the structure of probabilistic time. In *Symposium on Theory of Computing* (STOC), pages 303–316, 2021.

[LP20] Yanyi Liu and Rafael Pass. On one-way functions and Kolmogorov complexity. In *Symposium on Foundations of Computer Science* (FOCS), pages 1243–1254, 2020.

[LV92] Ming Li and Paul M. B. Vitányi. Average case complexity under the universal distribution equals worst-case complexity. *Inf. Process. Lett.*, 42(3):145–149, 1992.

[LV19] Ming Li and Paul M. B. Vitányi. *An introduction to Kolmogorov complexity and its applications*. Springer-Verlag, 2019. 4th edition (1st edition in 1993).

[Mad13] Aleksander Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *Symposium on Foundations of Computer Science* (FOCS), pages 253–262, 2013.

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

[Oli19] Igor C. Oliveira. Randomness and intractability in Kolmogorov complexity. In *International Colloquium on Automata, Languages, and Programming* (ICALP), pages 32:1–32:14, 2019.

[RS21] Hanlin Ren and Rahul Santhanam. Hardness of KT characterizes parallel cryptography. In *Computational Complexity Conference* (CCC), pages 35:1–35:58, 2021.

[RST15] Benjamin Rossman, Rocco A. Servedio, and Li-Yang Tan. Complexity theory column 89: The polynomial hierarchy, random oracles, and boolean circuits. *SIGACT News*, 46(4):50–68, 2015.

[Sip83] Michael Sipser. A complexity theoretic approach to randomness. In *Symposium on Theory of Computing* (STOC), pages 330–335, 1983.

[ST19] Rocco A. Servedio and Li-Yang Tan. Improved pseudorandom generators from pseudorandom multi-switching lemmas. In *International Workshop on Randomization and Approximation Techniques* (RANDOM), pages 45:1–45:23, 2019.

[SU05] Ronen Shaltiel and Christopher Umans. Simple extractors for all min-entropies and a new pseudorandom generator. *J. ACM*, 52(2):172–216, 2005.

[Tal17] Avishay Tal. Tight bounds on the fourier spectrum of AC^0. In *Conference on Computational Complexity* (CCC), pages 15:1–15:31, 2017.

[TUZ07] Amnon Ta-Shma, Christopher Umans, and David Zuckerman. Lossless condensers, unbalanced expanders, and extractors. *Combinatorica*, 27(2):213–240, 2007.

[TX13] Luca Trevisan and Tongke Xue. A derandomized switching lemma and an improved derandomization of AC^0. In *Conference on Computational Complexity* (CCC), pages 242–247, 2013.

[Vad12] Salil P. Vadhan. Pseudorandomness. *Found. Trends Theor. Comput. Sci.*, 7(1-3):1–336, 2012.

[VZ13] Salil P. Vadhan and Colin Jia Zheng. A uniform min-max theorem with applications in cryptography. In *Annual Cryptology Conference* (CRYPTO), pages 93–110, 2013.

A Estimating the probability of sampling a given string

In Theorem 1, we are assuming that the compressor has both the code of the sampler A and δ which estimates from below the probability p_x with which the string x is sampled.

This seems redundant, because with the sampler A and x in her hands, the compressor can run A and find a good estimation δ of p_x. While this is true, there is a cost: assuming black-box access to the sampler, we need to run it $\Omega(1/p_x)$ times to get an estimation of p_x within a constant multiplicative factor. This follows from the following fact, proved by Bruno Bauwens (private communication). (Note: We have chosen the multiplicative factor of 2 for simplicity, it can be replaced with any positive constant).

Proposition 39. *Consider the following task: the input is a binary string u of length N. By doing random probes in u, we want to find with probability $1-\epsilon$ a number $\tilde{p} \in (\frac{1}{2}p, 2p)$, where p is the fraction of 1's in p.*

Then the number of probes has to be larger than $m_\varepsilon(p) := (1/p)\cdot((1/2)\ln(1/\varepsilon)))$ (provided $p > 0$).

Proof. Suppose there is an algorithm that does at most $m_\epsilon(p_u)$ probes for all N-bit strings u and finds the estimation of p_u (the fraction of 1's in u) with the required precision, and with probability $1-\epsilon$. Let u_1 be a string $\in \{0,1\}^N$ that has sN 1's, and u_2 be a string in $\{0,1\}^N$ that has $4sN$ 1's, where s is some value in $(0, 1/8)$. If we read $m := m_\varepsilon(s)$ probes from u_1, the probability that all the probes turn out to be 0's is at least $(1-s)^m$, which is greater than ϵ. The same happens if we read $m_\varepsilon(4s)$ probes from u_2. If we only probe 0's, the algorithm will perform in the same way for both u_1 and u_2. Since the intervals $(\frac{1}{2}s, 2s)$ and $(\frac{1}{2}4s, 2(4s))$ are disjoint, the algorithm will make a mistake in one of the two situations, contradicting that the error probability for all strings is at most ϵ. □

We next show that the lower bound in Proposition 39 is tight: there exists an algorithm that runs the sampler $(1/p)\cdot 8\log(1/\varepsilon)$ times and estimates p within the multiplicative factor of 2. We start with the following lemma.

Lemma 40. *Let x be an n-bit string and let p_x be the probability that a sampler A produces x. We assume $p_x > 0$. Let $s = 4\log(1/\varepsilon)$ for some parameter $\varepsilon > 0$. A* success *is a run of the sampler A that produces x. Let T be the number of times we run the sampler till there are s successes. Let $\mathcal{E}$ be the event*

$$(1/2) \cdot s \cdot (1/p_x) \leq T \leq 2 \cdot s \cdot (1/p_x).$$

Then $\mathcal{E}$ has probability $1 - 2\varepsilon$.

Proof. Let T be the number of samplings till there are s successes. The expected value of T is $\mu_T = s(1/p_x)$, because the expected number of samplings till each success is $1/p_x$ and T is the sum of s random variables with this expectation.

We use a known technique to obtain concentration bounds for the geometric distribution. We first estimate the probability that the second inequality in event $\mathcal{E}$ fails. This is the probability that $T > 2\mu_T$, which is equal to the probability of the event $\mathcal{A}$ = "In $2\mu_T$ samplings the number of successes is $< s$." Let Z be the number of successes in $2\mu_T$ samplings. The expected value of Z is

$$\mu_Z = 2\mu_T \cdot p_x = 2s(1/p_x) \cdot p_x = 2s.$$

Then the second inequality in $\mathcal{E}$ fails with probability

$$\mathbf{Pr}[\mathcal{A}] = \mathbf{Pr}[Z < s] = \mathbf{Pr}[Z < (1/2)\mu_Z] < e^{-(1/4)\cdot(2s/2)} = \varepsilon.$$

(We have used the Chernoff bound $\mathbf{Pr}[Z < (1-\delta)\mu_Z] \leq e^{-(\delta^2\mu_Z)/2}$, for $\delta = 1/2$.)

We now estimate in the same way the probability that the first inequality in event $\mathcal{E}$ fails, which is the probability that $T < (1/2)\mu_T$, which is equal to the probability of the event $\mathcal{B}$ = "In $(1/2)\mu_T$ samplings the number of successes is $> s$." Let W be the number of successes in $(1/2)\mu_T$ samplings. The expected value of W is

$$\mu_W = (1/2)\mu_T \cdot p_x = (1/2)s(1/p_x) \cdot p_x = s/2.$$

Then the first inequality in $\mathcal{E}$ fails with probability

$$\mathbf{Pr}[\mathcal{B}] = \mathbf{Pr}[W > s] = \mathbf{Pr}[W > 2\mu_W] < (e/4)^{s/2} < \varepsilon.$$

(We have used the Chernoff bound $\mathbf{Pr}[W > (1+\delta)\mu_W] \leq \left(\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right)^{\mu_W}$, for $\delta = 1$.) □

Proposition 41 (Algorithm for estimating the probability with which a string is sampled)**.** *Let A be a sampler that produces strings of length n. There is an algorithm that on input an n-bit string x that is sampled by A, and $\varepsilon > 0$, has the following behaviour with probability at least $1 - 2\varepsilon$:*

- *If $p_x \geq 2^{-n}$, then it calls the sampler at most $(1/p_x) \cdot 8\log(1/\epsilon)$ times and returns a value $\tilde{p} \in \left[\frac{1}{2}p_x, 2p_x\right]$,*
- *If $p_x < 2^{-n}$, then it calls the sampler at most $2^n \cdot 8\log(1/\varepsilon)$ times and returns a value $\tilde{p} \leq 2^{-(n-1)}$.*

Proof. The algorithm runs as follows:

We run the sampler multiple rounds and we halt when either

(a) the sampler has obtained x (the "success" event) $s := 4\log(1/\varepsilon)$ times, or

(b) in $2s2^n$ sampling rounds, the number of successes is less than s.

In other words, we stop sampling immediately when we obtain the s-th success, or if $2s2^n$ samplings did not manage to do this.
Let T denote the number of samplings. In case (a), the algorithm returns $\tilde{p} = s/T$, and, in case (b) it returns $\tilde{p} = 2^{-n}$.

The conclusion follows with an analysis of the following cases, in which we condition on the event $\mathcal{E}$ from Lemma 40 (which holds with probability $1 - 2\varepsilon$).

- Suppose $p_x \geq 2^{-n}$. Then $T \leq 2s \cdot (1/p_x) \leq 2s \cdot 2^n$, and therefore case (a) holds. Then the algorithm returns $s/T \in \left(\frac{1}{2}p_x, 2p_x\right)$ (recall that we are conditioning on $\mathcal{E}$).
- Now, suppose $p_x < 2^{-n}$. Then the algorithm returns either $\tilde{p} = s/T \leq 2p_x < 2 \cdot 2^{-n}$ (if case (a) holds), or $\tilde{p} = 2^{-n}$ (if case (b) holds). The bound on the number of calls follows because the algorithm never does more than $2s2^n$ calls.

□

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