Verifiable Capacity-bound Functions: A New Primitive from Kolmogorov Complexity

Giuseppe Ateniese[1], Long Chen[2], Danilo Francati[1⋆], Dimitrios Papadopoulos[3], and Qiang Tang[4]

[1] Stevens Institute of Technology, New Jersey, USA
gatenies@stevens.edu, dfrancat@stevens.edu
[2] New Jersey Institute of Technology, New Jersey, USA
longchen@njit.edu
[3] Hong Kong University of Science and Technology, Hong Kong
dipapado@cse.ust.hk
[4] The University of Sydney, Sydney, Australia
qiang.tang@sydney.edu.au

Abstract. We initiate the study of *verifiable capacity-bound function* (VCBF). The main VCBF property imposes a lower bound on the number of bits read from memory during evaluation (referred to as minimum capacity). No adversary, even with unbounded resources, should produce an output without spending this minimum memory capacity. Moreover, a VCBF allows for an efficient public verification process: Given a proof-of-correctness, checking the validity of the output takes significantly fewer memory resources, sublinear in the target minimum capacity. Finally, it achieves soundness, i.e., no computationally bounded adversary can produce a proof that passes verification for a false output. With these properties, we believe a VCBF can be viewed as a "space" analog of a verifiable delay function. We then propose the first VCBF construction relying on evaluating a degree-d polynomial f from $\mathbb{F}_p[x]$ at a random point. We leverage ideas from *Kolmogorov complexity* to prove that sampling f from a large set (i.e., for high-enough d) ensures that evaluation must entail reading a number of bits proportional to the size of its coefficients. Moreover, our construction benefits from existing verifiable polynomial evaluation schemes to realize our efficient verification requirements. In practice, for a field of order $O(2^\lambda)$ our VCBF achieves $O((d+1)\lambda)$ minimum capacity, whereas verification requires just $O(\lambda)$.

Keywords: Kolmogorov complexity · Polynomial evaluation · Verifiable computation · Verifiable delay function

1 Introduction

Time and space complexity are functions that measure the efficiency of algorithms. These two functions are related (sometimes appear in the same setting) but distinct. For instance, "time" may refer to the number of memory

⋆ This work is based on the author's Ph.D dissertation.

accesses performed by an algorithm, while "space" refers to the amount of memory needed. In general, we try to minimize these functions, i.e., an ideal algorithm is one that is fast and tight. However, in cryptography, we are also interested in algorithms that are deliberately slow or capacious with the idea that, if the adversary must run them, the attack will be slow and costly. This has found numerous applications, e.g., in the context of proof-of-work for distributed consensus [46], and anti-spam mechanisms [28,8]; and password hashing or key derivations to be used against offline brute-force [52,40].

Existing Notions for "space-demanding" functions. The most prominent definitions for "space-demanding" functions proposed in the literature are memory-hardness [50,5,2,19,4,3,21], and bandwidth-hardness [55,15]. While they share the same initial motivation, these notions vary in their formalization and achieved security guarantees. Memory-hardness, as originally defined [50], guarantees a lower bound in the memory/time product required to compute the function. Informally, a function is memory-hard if the product of the evaluation memory cost m and time t for any adversary cannot be less than $mt \in O(n^2)$, where $O(n)$ is the time for an honest part. This has been widely proposed as a countermeasure against attackers that aim to gain an unfair advantage by using customized hardware, such as an ASIC, as it forces one to dedicate a significant area of memory to avoid being too slow. Thus, the cost of ASIC manufacturing would grow proportionally. Bandwidth-hardness guarantees that the *energy cost* for evaluating the function does not differ much across different platforms with variable computing energy costs (e.g., CPU vs. ASIC). In practice, this is based on the observation that although ASICs may have superior energy consumption for specific tasks, off-chip memory accesses incur comparable energy costs on ASICs and CPUs. Thus, energy consumption is enforced by ensuring a substantial amount of off-chip memory accesses.

None of these provides a strict bound on the amount of accessed memory: The former allows for a trade-off between memory accesses and computing, whereas the latter bounds the ratio of energy consumption benefits for ASIC adversaries. A different notion, predating memory and bandwidth-hardness, is that of memory-bound functions [27,29,1] that do impose a lower bound on the number of memory accesses, expressed as cache-misses.

All these notions have "symmetric" hardness in the following sense. Given a candidate input-output pair (x, y) for function f, verifying whether $f(x) = y$ is, at best, achieved by evaluating f. In that sense, evaluation and checking require the same amount of resources. In many applications, it would be desirable to have an *efficient public verification* algorithm that can check the correctness of an evaluation using significantly fewer resources, after the party that evaluates f provides a proof of correctness π for y. In practice, considering a cryptographic puzzle application [28,38,44], a challenger receiving multiple candidate puzzle solutions from different parties should be able to verify their correctness with much less effort than it took to compute them. Even considering egalitarian proofs of work [11], checking the validity of a proposed evaluation with con-

siderably smaller memory requirements allows for easy validation by numerous lightweight clients.

In the context of time-demanding functions, verifiable delay functions (VDFs) introduced by Boneh et al. [18] achieve such a property; any observer can verify that the computation of the function was performed correctly and can do so efficiently. The scope of this paper is to introduce an analogous function but for capacious/space-hungry algorithms. However, "space" or memory functions appear to be more intricate. Indeed, space-hardness does cover the memory needed by an algorithm for instructions, data, and inputs. Still, as discussed above, hardness often involves a trade-off between space and time, i.e., an algorithm is allowed to use more time to make up for a smaller memory footprint.

This work: Verifiable capacity-bound functions. In this work, we initiate the study of *verifiable capacity-bound functions (VCBF)*. At a high level, a VCBF guarantees: (a) a lower bound m in the necessary number of bits read from memory in order to evaluate the function (referred to as *minimum capacity* complexity), (b) a public verification process that given a proof π can check the correctness of an evaluation by reading only $o(m)$ bits, and (c) soundness, i.e., no computationally-bounded adversary should be able to produce a convincing proof for an incorrect evaluation.

Unlike the notion of asymmetric hardness [12], which allows parties with access to a secret trapdoor to evaluate f quickly, we aim for public verifiability. In that sense, a VCBF can be viewed as a space-analog of a VDF. Next, we provide a more detailed discussion of the relation between VCBFs and other primitives that attempt to bound the resources used when evaluating a function.

Minimum number of computation steps. Such primitives provide a lower bound on the minimum number of *sequential steps* necessary. Notable examples include classic time-locked puzzles [56], key-derivation function PBKDF2 [40], and the recently proposed verifiable delay functions mentioned above [18,60,51]. Another related notion is proof-of-sequential-work (PoSW) [43,23,26], which is similar to VDF except PoSW is not a function. Typically, these enforce a repeated operation (hashing or squaring in the group with an unknown order). As discussed, our VCBF shares the same spirit as VDF but for space/energy consumption.

Minimum number of memory access. As explained above, memory-bound functions provide an amortized lower bound on the number of memory accesses for any polynomial-time bounded adversary. However, they do not support (space-) efficient verification. In [1,27] a big random table (thus incompressible) is accessed during evaluation. For verification, this would need to be transferred over the network, and the verifier must mimic evaluation. Follow-up work [29] suggests a time/space trade-off for the process of constructing the table from a representation, but this allows an adversary to easily trade memory accesses for computation workload. One could consider building a VCBF by combining the constructions of [1,27] with a succinct non-interactive argument of knowledge [13] to verify the matrix traversal given only its short representation. Intuitively, it seems this would achieve very similar properties to a VCBF. That said, un-

like our result, which is in the standard model, such a construction would *de facto* have to employ non-standard assumptions, due to both the random oracle of [1,27] and SNARK impossibility results [35].

Code-hard functions. Code-hard algorithms require that a minimum amount of *space* is used in order to store the code. This has found different applications, e.g., white box encryption [10,16,17,34] or big-key encryption [9]. The key difference between a code-hard function and VCBF is that while a large amount of memory space must be dedicated for storing f, it is possible that only a small fraction of those stored bits must be retrieved during evaluation. A VCBF imposes a strict lower bound on bits read from memory during each evaluation.

Memory and Bandwidth-hard functions. As discussed above, both these definitions may allow adversaries to trade additional computation for reduced space or memory accesses; thus, they do not meet our strict lower bound guarantee. Moreover, many existing formalizations are highly reliant on the random oracle model, e.g., [55] for bandwidth hardness and [5] for memory hardness (in the parallel random oracle model). This comes naturally, as many of these works use variations of a graph-pebbling game to model their computation, heuristically estimating the energy cost for each unit computation and memory access operations. On the other hand, our VCBF definition does not rely on the random oracle model (clearly, this does not preclude the possibility of specific VCBF constructions operating in this model). Another impact of relying on the random oracle model is that it makes it harder to design an efficient verification algorithm as it "destroys" any algebraic structures between inputs and outputs.

We stress that a VCBF's lower bound in memory bits accessed can be used to infer a lower bound in energy consumption, analogous to the motivation behind bandwidth-hard functions. E.g., considering an ASIC-based adversary with on-chip memory of size s bits (such as a hardware cache) a VCBF that guarantees to access m bits from main memory imposes a $u(m-s)$ lower energy consumption, where u is the atomic cost for reading one bit from memory.

In a recent work [32], the first memory-hard VDF construction was proposed by combining a SNARK with parallelizable prover with a memory-hard "sequential" function. Although this result is close in spirit with what a VCBF tries to achieve, we do not aim for an explicit time lower bound, whereas the memory bound we achieve is strict without leaving room for space/time trade-offs, as explained above.

Proof of space (PoSpace). PoSpace is a puzzle system analogous to proof of work [6,30,54]. It extends memory-hard functions with efficient verification and adopts the graph pebbling framework and the random oracle model. The prover convinces a verifier that it consumed its space capacity to store data while allowing for efficient verification in both space and time. Like memory-hard functions, the PoSpace constructions can only guarantee a space-time tradeoff, thus cannot enforce a space lower bound. Also, the security analysis is based on the heuristic (parallel) random oracle model.

Symmetric key primitives against memory-bounded adversaries. A relatively recent series of works studies symmetric key primitives against memory-bounded

adversaries [59,38,37,25,36]. These results mainly focus on asymptotic adversaries with time-space trade-off without enforcing a resource lower bound. Moreover, VCBF (and the previously mentioned primitives) only require constructions with moderate hardness.

Overview of Techniques. The main challenge in building a VCBF is finding a function that has a natural lower bound on the space necessary for evaluation while still allowing for efficient verification. Previous works [55,15,5,2,19,4,3,21] achieve the first property by relying on assumptions such as the random oracle or ideal cipher. However, this approach makes it harder to achieve the second property as it dismantles structured relations between the function's inputs and outputs that could be used for efficient verification.

In this work, we deviate from previous techniques significantly. To model the inability of an adversary to compute an output without reading enough data from memory, we turn our attention to *Kolmogorov complexity* [41], which measures the complexity of an object in terms of the minimum number of bits necessary to represent it. Kolmogorov complexity is viewed as a fundamental theory of computer science and has been shown connected with multiple areas in cryptography [42,45,57]. (The most recent work of Liu and Pass [42] proves the equivalence of a computational bounded version of the Kolmogorov complexity and the existence of one-way functions.) Somewhat more formally, the Kolmogorov complexity of object x is the minimum number of bits needed to represent any *description* (T, α) where T is a Turing machine and α is a string such that $\mathsf{T}(\alpha)$ outputs x. One can view T as a decompressing algorithm and α as a "compression" of x. Based on this, our first observation is that if an algorithm depends on an object x (e.g., x could be the description of the algorithm itself or the algorithm's input), then its execution *cannot require reading fewer bits than the Kolmogorov complexity of* x. In that sense, Kolmogorov complexity is the right tool for us; choosing a function with high Kolmogorov complexity readily provides an arguably loose bound for the minimum capacity requirement of a VCBF.

On the other hand, when building our VCBF we need to identify a function that is amenable to verification; ideally, it should preserve an efficiently checkable (algebraic) relation between inputs and outputs. One candidate function is *polynomial evaluation* for single-variable polynomial $f(X) \in \mathbb{F}_p[x]$ of degree d of the form $f(X) = \sum_{i=0}^{d} a_i \cdot x^i$. The good news is that there exist numerous works in the literature for verifiable polynomial evaluation (e.g., [33,48,62,31]). In order to use such a scheme for a VCBF we need to ensure it is *publicly verifiable* (anyone can verify it using public parameters) and *publicly delegatable* (anyone can query it on an evaluation point). In our construction, we use the lightweight scheme of Elkhiyaoui et al. [31]. Its verification process requires a constant number of operations among a constant number of elliptic curve elements. This is important for us since we want VCBF to have verification capacity complexity *sublinear* in its evaluation's minimum capacity. Using [31], the latter is $O((d+1)\lambda)$ whereas the former is $O(\lambda)$ (where λ is the security parameter), i.e., the gap is linear in the degree of the polynomial.

The "honest" way of evaluating polynomial $f(X)$ is by reading its coefficients a_i, so by fixing $|(a_0, \ldots, a_d)| \geq m$ one would hope to get a VCBF with minimum capacity m. However, this is not the case as every polynomial has multiple alternative representations that an adversary may try to exploit in order to bypass the memory capacity bound. For example, all lists of the form $(x_0, \ldots, x_d)$, $(f(x_0), \ldots, f(x_d))$, for any choice of $d+1$ distinct x_i, completely determine the coefficients $(a_0, \ldots, a_d)$ of $f(X)$ (by interpolating the points). Here is where Kolmogorov complexity comes in handy: The above list of evaluations and points together with a Turing machine that performs polynomial interpolation are a valid description, in terms of Kolmogorov complexity, of the coefficients $(a_0, \ldots, a_d)$. As a consequence, it *cannot be significantly shorter* than the Kolmogorov complexity $C(a_0, \ldots, a_d)$ of the coefficients of the polynomial $f(X)$ (Theorem 6).

What remains is to find a way to sample a polynomial $f(X)$ with high Kolmogorov complexity (see Section 2.3). For any large-enough set, most of its elements have sufficiently high Kolmogorov complexity (Theorem 3). Since this holds for arbitrary sets, sampling at random from a large-enough set of polynomials guarantees that the chosen polynomial is of high Kolmogorov complexity with high-enough probability.

As discussed above, many previous works inherently adopt non-standard models in their definitions to capture the fact that a function is memory-heavy (e.g., random-oracle, ideal cipher, or heuristic assumptions about graph pebbling). Instead, we want to base our security definition in the standard setting, and we regard our paper on VCBF as a foundational one. Our approach is to model adversaries as Turing machines that read (at most) a fixed number of distinct bits m (whose value is estimated using the Kolmogorov complexity) from a precomputed memory τ of size $n \geq m$ (Section 4). We stress that it is crucial to consider the memory of size n larger than m since an adversary can leverage a large memory to increase its advantage ϵ while, at the same time, minimizing the number m of distinct bits it must read to answer a particular challenge (for example, it can store a large dictionary containing several evaluations of the polynomial $f(X) \in \mathbb{F}_p[x]$). However, this introduces the new challenge of estimating the adversary's advantage ϵ with respect to the memory size n: A particularly challenging task when working in the standard model with black-box access to the adversary. In more detail, it is hard to provide a bound on the number of (partial) information that can be stored in a memory of size n since their space requirement highly depends on the precomputation strategy (e.g., the entropy of the precomputed values) and the encoding (e.g., memory organization, memory access patterns) used by the adversary. Still, we show that it is possible to give a positive, meaningful estimation of ϵ and n when considering adversaries that perform a restricted number of v random accesses (e.g., conditional jumps) in order to read discontinuous bits from memory. We discuss the details of the formulation of our definition and our results in Section 4 and Section 5.1, respectively.

Summary of Our Contributions. Our contributions in this work can be summarized as follows:

1. We build a cryptographic framework that combines the notion of Kolmogorov complexity and randomized Turing machines and use it to bound the minimum amount of bits required in order to evaluate a polynomial $f(X) = \sum_{i=0}^{d} a_i \cdot x^i$ (Section 3).
2. We propose a formal definition of verifiable capacity-bound functions VCBFs that captures (a) a lower bound m on the number of bits read from memory (of bounded size) for evaluation (minimum capacity), (b) efficient verification of outputs with minimum capacity that is sublinear in m with respect to any malicious evaluator, and (c) soundness, i.e., no computationally bounded adversary can produce an incorrect evaluation output that passes verification (Section 4).
3. We propose the first VCBF construction that satisfies our definition, based on single-variable polynomial evaluation for polynomial $f(X) \in \mathbb{F}_p[x]$ of degree d. To achieve efficient verification, we employ the publicly verifiable and publicly delegatable verifiable computation scheme of [31]. For a target minimum capacity $m \in O((d+1)\lambda)$, it suffices to set the size of the polynomial to $(d+1)\lambda$, where λ is the security parameter. Hence, to achieve large capacity bounds, we need to set $d \gg \lambda$, e.g., $d \in O(\lambda^c)$ for $c > 1$ constant. On the other hand the capacity complexity of the verification is $O(\lambda)$, i.e., independent of d hence verification remains efficient (Section 5).
4. We provide an estimation of the concrete parameters for our construction in Table 1. For an elliptic curve group of order p of size 1024 bits, a polynomial of size 1GB ($d = 78.20 \cdot 10^5 \approx \lambda^{2.29}$) guarantees a minimum capacity m of 0.82GB, even with respect to an unbounded adversary that can spend an exponential amount of computational resources.

We stress that a target minimum capacity m of a VCBF is guaranteed only in the presence of adversaries with a limited memory size n. As explained, the estimation of n is a major challenge when working in the standard setting (this work). Along this line, we initiate a fine-grained study on the memory size n estimation according to the number v of (adaptive) random accesses performed by the adversary (denoted by the set $\mathcal{A}^{v\text{-access}}$). In particular, we prove (in the concrete setting) that the evaluation of a polynomial $f(X) \in \mathbb{F}_p[x]$ guarantees a target capacity $m \in O((d+1)\lambda)$ even if an adversary $\mathsf{A} \in \mathcal{A}^{1\text{-access}}$ has access to a memory whose size n is proportional to the cardinality of the input space of the polynomial $f(X)$, i.e., exponential (Theorem 9). Our results can be extended to the asymptotic setting for the class $\mathcal{A}^{O(1)\text{-access}}$. However, such a result should be interpreted only as a purely theoretical result about the feasibility of VCBF. This is because the constants hidden by the asymptotic notation are not negligible. In Section 4 and Section 5.1 we discuss the results included in this work in more detail.

1.1 Applications of VCBF

Since VCBF can be seen as a space-analog of VDF, replacing minimum sequential steps with a minimum number of bits retrieved from memory, we believe they can find applications in a variety of settings where memory-usage needs to be enforced. In this direction, we describe how VCBF can be used for building, *energy-consumption client puzzles* that achieve fairness among ASIC and CPU participants. We then briefly discuss other potential VCBF applications.

Energy-consumption client puzzles. The concept of cryptographic puzzles can be traced back to Merkle's key exchange [44] and Dwork and Naor's pricing function [28]. The formal notion of a client-puzzle was proposed by Juels and Brainard [39] to mitigate denial of service attacks. The general idea of such puzzles is to associate a cost to each resource allocation request by requiring the client to complete a task before the server performs any expensive operation, thus making large-scale attacks infeasible. Classic client puzzles [39,7,20,22,58] will force adversaries to consume certain CPU cycles as the cost for attacks. However, state-of-the-art hash engines [14,55] could be $200,000\times$ faster and $40,000\times$ more energy-efficient than a state-of-art multi-core CPU. Hence, denial-of-service attacks may still be feasible for ASIC-equipped adversaries, even when such client puzzles are deployed as counter-mechanisms.

Motivated by this, we propose client puzzles that replace CPU cycles with alternative resources, i.e., energy consumption using a VCBF . Classic ASIC-resistant methods follow the memory-hard function approach, i.e., ensuring that solving the puzzle "costs" much memory. In this manner, the cost of manufacturing an ASIC for puzzle solving would increase proportionally to the chip area devoted to memory. However, as argued in [55], memory hardness only partially solves the problem since it does not address the energy aspect of ASIC advantage. Indeed, energy consumption can be more important than the one-shot ASIC manufacturing cost since the corresponding cost (due to electricity consumption) keeps accumulating with time. Hence, a puzzle with a lower bound on energy consumption, due to off-chip memory accesses enforced via VCBF, could fill in a critical, but often overlooked, gap in ASIC-resistance.

Our energy consumption puzzle could be a protocol between a server S and a client C using VCBF f as follows:

- C contacts server S, requesting permission to use some service such as establishing TLS connection [24] or accepting an email [28].
- S returns a fresh challenge x to the client C.
- C evaluates VCBF f on x, and returns the output and proof π to the server S.
- The server S verifies the correctness of $f(x)$. If the verification succeeds, it allows C to use the service.

Jumping ahead, from Theorem 8 (in Section 5), we could easily find a set of parameters so that an adversary needs to invest a sufficiently large amount of energy to solve a puzzle.

Other possible applications. Beyond energy puzzles, we believe VCBF may find a plethora of other applications. One such application could be a proof of network bandwidth that allows paying customers to verify their real-time upload bandwidth by requiring their cloud storage service provider to retrieve an uploaded test file. Unlike existing bandwidth puzzles (e.g., [53,63]), a VCBF-based solution would not rely on random oracles. Other applications may include digital rights management (by requesting the symmetric-key decryption algorithm used to retrieve copyrighted data to be a VCBF; hence illegal sharing it on a large scale would be less feasible), or securing mobile payments (by requiring that the signing process to authorize a transaction is a VCBF, making it hard for attackers to "extract" a signing key).

2 Preliminaries

2.1 Notation

We use the notation $[n] \stackrel{\text{def}}{=} \{1, \ldots, n\}$. Capital boldface letters (such as $\mathbf{X}$) are used to denote random variables, small letters (such as x) to denote concrete values, calligraphic letters (such as $\mathcal{X}$) to denote sets, and sans-serif letters (such as A) to denote algorithms. All of our algorithms are modeled as (possibly interactive) Turing machines. For a string $x \in \{0,1\}^*$, we let $|x|$ be its length; if $\mathcal{X}$ is a set, $|\mathcal{X}|$ represents the cardinality of $\mathcal{X}$. When x is chosen randomly in $\mathcal{X}$, we write $x \leftarrow\!\!\$\ \mathcal{X}$.

Turing machines. If A is an algorithm, we write $y \leftarrow\!\!\$\ \mathsf{A}(x)$ to denote a run of A on input x and output y. If A is randomized, y is a random variable and $\mathsf{A}(x;r)$ denotes a run of A on input x and randomness $r \leftarrow\!\!\$\ \{0,1\}^{\ell_{rnd}}$ where $\{0,1\}^{\ell_{rnd}}$ is the randomness space of A. An algorithm A is *probabilistic polynomial-time* (PPT) if A is randomized and for any input $x \in \{0,1\}^*, r \in \{0,1\}^{\ell_{rnd}}$ the computation of $\mathsf{A}(x;r)$ terminates in a polynomial number of steps (in the input size). We say that an algorithm A is *constant-size* if the code of the Turing machine that implements A is independent of the input.

Self-delimiting codes and two-part codes. A self-delimiting code allows us to encode binary strings in a prefix-free fashion. A set $\mathcal{A}$ is prefix-free if there do not exist two strings $x, y \in \mathcal{A}$ such that $x||y = z$ and $z \in \mathcal{A}$. A prefix-free code allows Turing machines to read any input composed by multiple concatenated values, each of arbitrary length. In this work, we use the Elias delta coding (δ-code in short) that is asymptotically optimal in the length of the original string. Also, it is very efficient in the concrete setting. The δ-encoding E of a string x of size n is defined as $E(x) = 1^{|n|}||0||n||x$ where $|n| = \lceil \log(n) \rceil$ is the number of bits needed to represent n in binary. The δ-code produces relatively shorts codes of size $|E(x)| = \log(x) + 2\log(\log(x)) + 1$ that are asymptotically optimal: The bit length of $|E(x)|$ is equal to the length of x in the asymptotic setting, i.e., $|E(x)| \in O(|x|)$.

Self-delimiting codes are necessary whenever a Turing machine receives multiple input values that it must unambiguously read. Suppose a Turing machine T needs to perform some computation on two values (x, y). Such an input (x, y) can be encoded in a prefix-free fashion by leveraging two-part codes. In more detail, a two-part code $\langle \cdot, \cdot \rangle$ w.r.t. the δ-code $E(\cdot)$ is defined as $\langle x, y \rangle = E(x)||y$. Two-part codes can be used recursively. For example, three values (x, y, z) can be encoded as $\langle x, \langle y, z \rangle \rangle = E(x)||E(y)||z$. For the sake of clarity we write $\langle x, y, z \rangle = \langle x, \langle y, z \rangle \rangle$.

2.2 Publicly Verifiable Computation for Polynomial Evaluation

A publicly verifiable computation scheme (VC) for polynomial evaluation allows a client to outsource the computation of a polynomial f to an untrusted server. We are interested in VC schemes that are both *publicly delegatable* and *publicly verifiable*. The former allows any querier to submit input to the server, while the latter allows any verifier to check the computation's correctness.

Formally, a VC scheme for a family of polynomials $\mathcal{F}$ with input space $\mathcal{X}$ is composed of the following algorithms:

$\mathsf{Setup}(1^\lambda, f)$**:** Upon input the security parameter 1^λ and a polynomial $f \in \mathcal{F}$, the randomized setup algorithm returns the evaluation key ek_f and the verification key vk_f for the polynomial f.

$\mathsf{ProbGen}(\mathsf{vk}_f, x)$**:** Upon input the verification key vk_f for a polynomial $f \in \mathcal{F}$ and an input $x \in \mathcal{X}$, the deterministic problem generation algorithm outputs an encoding σ_x and the verification key vk_x for the input x.

$\mathsf{Compute}(\mathsf{ek}_f, \sigma_x)$**:** Upon input the evaluation key ek_f for a polynomial $f \in \mathcal{F}$ and an encoding σ_x for input $x \in \mathcal{X}$, the deterministic computation algorithm returns a value y and a proof π_y.[5]

$\mathsf{Verify}(\mathsf{vk}_x, y, \pi_y)$**:** Upon input the verification key vk_f for a polynomial $f \in \mathcal{F}$, the verification key vk_x for an input $x \in \mathcal{X}$, a value $y \in \mathcal{Y}$, and a proof π_y, the deterministic verification algorithm returns a decisional bit b.

Correctness of a publicly VC scheme captures the fact that an honest execution of the computation to evaluate a polynomial $f \in \mathcal{F}$ on input $x \in \mathcal{X}$ produces the correct output $y = f(x)$ along with a proof π_y that correctly verifies.

Definition 1 (Correctness of VC). *A publicly VC scheme Π for a family of polynomials $\mathcal{F}$ with input space $\mathcal{X}$ is correct if $\forall \lambda \in \mathbb{N}$, $\forall f \in \mathcal{F}$, $\forall x \in \mathcal{X}$ the following holds:*

$$\Pr\left[\mathsf{Verify}(\mathsf{vk}_x, y, \pi_y) = 1 \;\middle|\; \begin{array}{l}(\mathsf{ek}_f, \mathsf{vk}_f) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, f) \\ (\sigma_x, \mathsf{vk}_x) = \mathsf{ProbGen}(\mathsf{vk}_f, x) \\ (y, \pi_y) = \mathsf{Compute}(\mathsf{ek}_f, \sigma_x)\end{array}\right] = 1.$$

[5] We explicitly detached y from its proof π_y. Several works define the output of the computation algorithm $\mathsf{Compute}$ as a singleton σ_y (the encoding of the output y) defined as $\sigma_y = (y, \pi_y)$.

As for security, a malicious evaluator cannot convince an honest verifier that $y^* \neq f(x^*)$ is the correct evaluation of $f(x^*)$ on an arbitrary input $x^* \in \mathcal{X}$. This security property is known as *soundeness*.

Definition 2 (Soundness of VC [49,31]). *A publicly VC scheme Π for a family of functions $\mathcal{F}$ with input space $\mathcal{X}$ is (ϵ)-sound if $\forall f \in \mathcal{F}$ and every PPT adversary $\mathsf{A} = (\mathsf{A}_0, \mathsf{A}_1)$ we have:*

$$\Pr\left[\mathbf{G}^{\mathsf{Sound}}_{\Pi,\mathsf{A}}(1^\lambda, f) = 1\right] \leq \epsilon,$$

where $\mathbf{G}^{\mathsf{Sound}}_{\Pi,\mathsf{A}}(1^\lambda, f)$ is defined as follows:

- $(\mathsf{ek}_f, \mathsf{vk}_f) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, f)$.
- $(x^*, \mathsf{aux}) \leftarrow\!\!\$\ \mathsf{A}_0(1^\lambda, \mathsf{ek}_f, \mathsf{vk}_f, f)$.
- $(\sigma_{x^*}, \mathsf{vk}_{x^*}) = \mathsf{ProbGen}(\mathsf{vk}_f, x^*)$.
- $(y^*, \pi_{y^*}) \leftarrow\!\!\$\ \mathsf{A}_1(\sigma_{x^*}, \mathsf{vk}_{x^*}, \mathsf{aux})$.
- *If* $\mathsf{Verify}(\mathsf{vk}_{x^*}, y^*, \pi_{y^*}) = 1$ *and* $y^* \neq f(x^*)$ *return* 1. *Otherwise, return* 0.

In this work, we are interested in single-variable polynomials $f(X) \in \mathbb{F}_p[x]$ of degree d of the form $f(X) = \sum_{i=0}^{d} a_i \cdot x^i$. An example of such a VC scheme has been proposed by Elkhiyaoui et al. [31]. It uses an asymmetric bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ where $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$ are groups of prime order p, and its security follows from the $(d/2)$-SDH assumption.

Time Efficient Verification. VC schemes allow verifiers to check the computation's correctness more efficiently than the work required for the polynomial evaluation; otherwise, the verifier would perform the computation by itself without outsourcing the computation to an external untrusted server. In more detail, if an honest evaluation of the polynomial $f(X)$ requires t steps, then the verification time complexity (i.e., the execution of $\mathsf{ProbGen}$ and Verify) must be sublinear in t (i.e., $o(t)$).[6]

The publicly VC scheme proposed in [31] yields a constant time $O(1)$ verification: The execution of $\mathsf{ProbGen}$ requires two multiplications and two exponentiations in $\mathbb{G}_1$ and $\mathbb{G}_2$ while Verify requires one multiplication in $\mathbb{G}_T$ and two pairing operations, where $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an asymmetric bilinear pairing.

2.3 Kolmogorov Complexity

The Kolmogorov complexity [41] aims to measure the complexity of objects in terms of the minimum amount of bits required to represent them. We define a description of a string $x \in \{0,1\}^*$ in terms of algorithmic complexity as follows.

Definition 3 (Description of a string). *Let T be a deterministic Turing machine and $x \in \{0,1\}^*$ a bit string. We say that (T, α) is a (possibly inefficient) description of x if $\mathsf{T}(\alpha) = x$.*

[6] We include the algorithm $\mathsf{ProbGen}$ as part of the verification phase since we consider VC schemes that are both *publicly delegatable* and *publicly verifiable*.

Kolmogorov complexity with respect to a Turing machine. We can look at T as a decoding algorithm and $\alpha \in \{0,1\}^*$ as an encoding of x. The minimum amount of bits needed to represent a fixed bit string x is measured by the Kolmogorov complexity $C_{\mathsf{T}}(x)$. In more detail, the Kolmogorov complexity $C_{\mathsf{T}}(x)$ of a bit string $x \in \{0,1\}^*$ with respect to a deterministic Turing machine T (called reference Turing machine) is defined as:

$$C_{\mathsf{T}}(x) = \min_{\alpha \in \{0,1\}^*} \{|\alpha| : \mathsf{T}(\alpha) = x\}.$$

Similarly, the conditional Kolmogorov complexity measures the complexity of x given some auxiliary information $y \in \{0,1\}^*$, i.e.,

$$C_{\mathsf{T}}(x|y) = \min_{\alpha \in \{0,1\}^*} \{|\alpha| : \mathsf{T}(\langle \alpha, y \rangle) = x\}.$$

The two above definitions of Kolmogorov complexity are known as *plain* Kolmogorov complexity. The name comes from the fact that no constraints are put on the input α of the Turing machine T. Another type of complexity, called *prefix-free* Kolmogorov complexity [41, Section 3], focuses only on prefix-free programs, i.e., Turing machines that only take in input strings encoded in a prefix-free fashion. In this work, we focus on the plain version, and we refer the reader to [41, Section 3] for a more detailed discussion about the prefix-free version.

Kolmogorov complexity independent of the Turing machine. It is possible to make the definition of plain Kolmogorov complexity independent from the reference Turing machine. Indeed, Turing machines are enumerable. The code of any Turing machine T can be interpreted as a binary string i.[7] Therefore, we can define a *universal* Turing machine U as $\mathsf{U}(i, \alpha) = \mathsf{T}_i(\alpha)$. In other words, U simulates all possible computations that Turing machines perform by taking in input $\alpha \in \{0,1\}^*$ and the code i of the i-th Turing machine T_i and executes the computation $\mathsf{T}_i(\alpha)$. Based on this observation, it has been proved that the Kolmogorov complexity with respect to different Turing machines is invariant only up to a constant that depends on the reference Turing machine.

Theorem 1 (Invariance Theorem [41, Theorem 2.1.1]). *There is a universal deterministic Turing machine U such that for any deterministic Turing machine T, there is a constant c_{T} that only depends on T, such that for any string $x, y \in \{0,1\}^*$:*

$$C_{\mathsf{U}}(x) \leq C_{\mathsf{T}}(x) + c_{\mathsf{T}} \text{ and } C_{\mathsf{U}}(x|y) \leq C_{\mathsf{T}}(x|y) + c_{\mathsf{T}},$$

or equivalently

$$|C(x) - C_{\mathsf{T}}(x)| \leq c_{\mathsf{T}} \text{ and } |C(x|y) - C_{\mathsf{T}}(x|y)| \leq c_{\mathsf{T}}.$$[8]

[7] Note that not all binary strings are valid Turing machines.

[8] The constant c_{T} corresponds to the self-delimiting description of the Turing machine T.

Since the choice of the reference Turing machine does not significantly change the Kolmogorov complexity of any string, we express the Kolmogorov complexity using the universal Turing machine U as a reference machine.

Definition 4. *The Kolmogorov complexity of a string x is defined as $C(x) \stackrel{def}{=} C_{\mathsf{U}}(x)$ and $C(x|y) \stackrel{def}{=} C_{\mathsf{U}}(x|y)$ for the universal Turing machine U.*

The Kolmogorov complexity is invariant up to a constant (Theorem 1) since it assumes Turing machines of constant-size, i.e., Turing machines whose description size does not depend on the input. Note that restricting the size of the Turing machine is necessary. As mentioned in [41, Section 2.1.4], by removing the size constraint of T, it is possible to assign low complexity to any string by simply selecting a reference Turing machine with large complexity. For example, consider the reference Turing machine T' that has hardcoded x into its code. In this case, T' is able to output x with no input, i.e., $\mathsf{T}'(\perp) = x$. Hence, we have $C_{\mathsf{T}'}(x) = 0$. However, this example does not violate the intuition of the Kolmogorov complexity since the large size of the Turing machine T' is captured by the large constant c_{T} in the invariance theorem. Such an example shows that it is impossible to reduce a string's complexity significantly. It may only be split among the reference Turing machine and its corresponding input.

The constant-size reference Turing machine assumption makes the definition of complexity precise while covering all the possible descriptions (T, α) of any string $x \in \{0,1\}^*$. As we will see later, this will be very useful in building VCBFs since it allows us to precisely measure the minimum number of bits an algorithm (whose computation is described by a Turing machine) needs to read to perform specific computations. Observe that the size constraint does not reduce the number of languages recognizable by a Turing machine. For example, it was shown the existence of a universal Turing machine with 15 states, 2 symbols, and 30 state-symbol product (transition function) [61,47], with a polynomial slowdown of $O(t^6)$.

Kolmogorov complexity upper bound. The Invariance Theorem (Theorem 1) allows us to prove upper bounds of $C(x)$. Intuitively, the Kolmogorov complexity of a string can not exceed the length of the string except for a constant that depends on the choice of the reference Turing machine.

Theorem 2 (Kolmogorov complexity upper bound [41, Theorem 2.1.2]). *There is a constant c such that, for all strings $x, y \in \{0,1\}^*$, the following holds:*

$$C(x) \leq |x| + c \text{ and } C(x|y) \leq C(x) + c.$$

String incompressibility. A crucial notion derived from the Kolmogorov complexity is the incompressibility of a string [41, Definition 2.2.1] with respect to unbounded deterministic Turing machines.

Definition 5 (Deterministic c-incompressibility [41, Definition 2.2.1]). *A string $x \in \{0,1\}^*$ is c-DET-incompressible if $C(x) \geq |x| - c$.*

We will refer to the above definition as *deterministic* c-incompressibility (c-DET-incompressibility in short) since it covers deterministic Turing machines, i.e., the universal reference Turing machine of the Kolmogorov complexity is deterministic.

The following theorem provides a lower-bound on the number of c-DET-incompressible elements in a given set $\mathcal{X}$.

Theorem 3 ([41, Theorem 2.2.1]). *Let $c \geq 0$ be a positive constant. For each $y \in \{0,1\}^*$, every finite set $\mathcal{X}$ of cardinality m has at least $m(1-2^{-c})+1$ elements $x \in \mathcal{X}$ such that $C(x|y) \geq \log(m) - c$.*

By leveraging Theorem 3, we can calculate the probability of sampling a c-DET-incompressible string from $\mathcal{X}$. The formal proof is included in Appendix A.1

Theorem 4. *Let $\mathcal{X}$ be a finite set of cardinality m, then the following probability holds:*

$$\Pr[x \text{ is } c\text{-DET-incompressible} \mid x \leftarrow\!\!\$\ \mathcal{X}] \geq 1 - 2^{-c} + \frac{1}{m}.$$

String incompressibility in the randomized setting. In cryptography, we deal with randomized adversaries represented by randomized Turing machines. However, the c-DET-incompressibility only covers deterministic Turing machines since the reference Turing machine (used to measure the Kolmogorov complexity) is deterministic. Accordingly, we extend the notion of c-DET-incompressibility to randomized Turing machines.

Definition 6 (Randomized (c, ℓ_{rnd})-incompressibility). *A string $x \in \{0,1\}^*$ is (c, ℓ_{rnd})-RND-incompressible if for all constant-size unbounded randomized Turing machine T with randomness space $\{0,1\}^{\ell_{rnd}}$, for all $r \in \{0,1\}^{\ell_{rnd}}$, and for all $\alpha \in \{0,1\}^{|x|-c}$, we have $\Pr[\mathsf{T}(\alpha; r) = x] = 0$.*

Recall that a string x is c-DET-incompressible (Definition 5) if $C_{\mathsf{T}}(x) \geq |x| - c$ where T is deterministic. The (c, ℓ_{rnd})-incompressibility follows the very same idea of its deterministic version, i.e., it measures the length of the smallest input (in addition to the randomness) that allows T to output x.

Naturally, there is an obvious connection between the two definitions of incompressibility. Indeed, the randomness of a randomized Turing machine can be seen as part of the input of a deterministic one. Hence, if a string $x \in \{0,1\}^*$ is c-DET-incompressible (Definition 5), then x is $(c + \ell_{rnd} + 2\log(\ell_{rnd}) + 1 + O(1), \ell_{rnd})$-RND-incompressible (Definition 6). The factor $\ell_{rnd} + 2\log(\ell_{rnd}) + 1$ is due to the self-delimiting δ-encoding (Section 2.1) to encode the randomness $r \in \{0,1\}^{\ell_{rnd}}$.[9] Observe that the relation between the two incompressibility definitions is up to a constant $O(1)$. This is because the DET-incompressibility leverages the Kolmogorov complexity defined over universal Turing machines U. In this setting, as implied by the invariance theorem (Theorem 1), any equality holds up to a constant factor. The following Theorem 5 reports the formal result, whose proof appears in Appendix A.2.

[9] We only consider constant-size Turing machines. This requires the use of a self-delimiting coding strategy.

Theorem 5. *Let $x \in \{0,1\}^*$ be a string. If x is c-DET-incompressible (Definition 5) then x is (c', ℓ_{rnd})-RND-incompressibile (Definition 6) where $c' = c + \ell_{rnd} + 2\log(\ell_{rnd}) + 1 + O(1)$.*

3 Kolmogorov-bound for Polynomial Evaluation

At each evaluation, a VCBF scheme forces the evaluator to read at least m distinct bits from its main memory. To achieve this functionality, our construction leverages a single variable polynomial $f(X) = \sum_{i=0}^{d} a_i \cdot x^i \in \mathbb{F}_p[x]$ of degree d. Intuitively, on receiving a challenge $x \in \{0,1\}^{\ell_{in}}$, an honest evaluator needs to read the coefficients $(a_0, \ldots, a_d) \in \mathbb{F}_p^{d+1}$ that determine the polynomial $f(X)$ in order to compute $y = f(x)$. In this case, we obtain the desired functionality by setting $|(a_0, \ldots, a_d)| \geq m$. However, a malicious evaluator may find an alternative strategy to compute $y = f(x)$ and read fewer than m bits. In this section, we prove the lower bound on the number of bits read during the polynomial evaluation by leveraging the Kolmogorov complexity.

Next, we provide some examples of strategies a malicious evaluator could adopt:

1. Compress the coefficients $(a_0, \ldots, a_d)$ into a smaller string α. In this way, the evaluator just needs to read α, decompress it into $(a_0, \ldots, a_d)$, and evaluate $f(X)$ on the desired point x.
2. Precompute a dictionary $T \stackrel{\text{def}}{=} (f(x_0), \ldots, f(x_n))$ composed of the evaluation of $f(X)$ on points $(x_0, \ldots, x_n)$. By accessing T, the malicious evaluator can simply read and return $y_i = f(x_i)$ if the challenge x_i is one of the precomputed points. In this case, the malicious evaluator reads only $|y_i| \leq |p| < m$.
3. Instead of storing $(a_0, \ldots, a_d)$, the evaluator may choose to store $d+1$ arbitrary points $(x_0, \ldots, x_d)$, the corresponding evaluations $(f(x_0), \ldots, f(x_d))$, and the prime p. These pieces of information are enough to recover a via polynomial interpolation. As a result, if the expression of $(f(x_0), \ldots, f(x_d))$, the points $(x_0, \ldots, x_d)$ and the prime p could be effectively compressed, the evaluator will read fewer bits than expected when evaluating the polynomial.

To estimate the bits that an adversary/algorithm needs to read to evaluate $f(X)$ correctly, we built a bridge between the Kolmogorov complexity and polynomial evaluation. Our approach is based on two main observations.

- First, any string a can be encoded into $f(X) = \sum_{i=0}^{d} a_i \cdot x^i$ by setting its coefficients to different sub-portions of a. Let p be a prime of size $\lambda + 1$ bits. We can interpret a string $a \in \{0,1\}^{(d+1)\lambda}$ as $a = a_0 || \ldots || a_d$ where $a_i \in \mathbb{F}_p$ (i.e., $|a_i| \leq \lambda < |p|$) and use $(a_0, \ldots, a_d)$ as the coefficients of $f(X)$.
- Second, if algorithm T is able to compute $(f(x_0), \ldots, f(x_d))$ taking in input a string α and the challenge d points $(x_0, \ldots, x_d)$, then $(\mathsf{T}, \langle \alpha, x_0, \ldots, x_d \rangle)$ is a valid description of $(f(x_0), \ldots, f(x_d))$ according to Definition 3. As explained in Item 3, the tuples $(f(x_0), \ldots, f(x_d))$, $(x_0, \ldots, x_d)$, and the prime p, are enough to reconstruct $(a_0, \ldots, a_d)$ via polynomial interpolation (i.e., the prime's size $\lambda + 1$ guarantees the encoding is injective).

By combining the above two observations, we can easily bound the size of α with the Kolmogorov complexity $C(a)$ of a. In more detail, consider a Turing machine T' that first executes $\mathsf{T}(\alpha, x_0, \ldots, x_d)$ to compute $f(x_0), \ldots, f(x_d)$, and then retrieves and return a via polynomial interpolation. This implies that $(\mathsf{T}', \langle p, \alpha, x_0, \ldots, x_d\rangle)$ is a description of a (according to Definition 3). As a consequence, the size of α (the string that T would read to compute $(f(x_0), \ldots, f(x_d))$) cannot be too small and must be related to the complexity $C(a)$ of a. Below, we provide the formal result whose proof appears in Appendix A.3.

Theorem 6 (Kolmogorov-bound for Polynomial Evaluation). *For any $\lambda \in \mathbb{N}$, let $a \in \{0,1\}^{(d+1)\lambda}$ and p be a binary string and a prime p of size $\lambda + 1$, respectively. Fix the polynomial $f(X) = \sum_{i=0}^{d} a_i \cdot x^i \in \mathbb{F}_p[x]$ of degree d with input space $\{0,1\}^{\ell_{in}}$ where $a = a_0||\ldots||a_d$ and $a_i \in \mathbb{F}_p$ for $i \in [d]$. If a is (c', ℓ_{rnd})-RND-incompressible (Definition 6), then for every constant-size randomized unbounded Turing machine T with randomness space $\{0,1\}^{\ell_{rnd}}$, every $\alpha \in \{0,1\}^m$, every $r \in \{0,1\}^{\ell_{rnd}}$, and every tuple $(x_0, \ldots, x_d)$ such that $\underset{i \neq j}{\forall} i, j \in \{0, \ldots, d\}$, $x_i \neq x_j$ and $x_i \in \{0,1\}^{\ell_{in}}$, the following probability holds:*

$$\Pr[(f(x_0), \ldots, f(x_d)) = \mathsf{T}(\alpha, x_0, \ldots, x_d; r)] = 0$$

where $m = (d+1)(\lambda - \ell_{in} - 2\log(\ell_{in}) - 1) - c' - \lambda - 2\log((d+1)\lambda - c') - 2\log(\lambda) - 2$.

Remark 1. An alternative way to interpret Theorem 6 is that any possible description (T, α) of $f(X)$ is bigger than the parameter m (defined in Theorem 6). Indeed, if there exists a string α that encodes $f(X)$ such that $|\alpha| < m$, then we contradict Theorem 6 since α would allow the prover to answer to any point $x \in \{0,1\}^{\ell_{in}}$. In addition, the parameter m depends on the size of both $a \in \{0,1\}^{(d+1)\lambda}$ and $x \in \{0,1\}^{\ell_{in}}$. In particular, we have a loss factor that is proportional to $(d+1)\ell_{in}$. Hence, for some values of ℓ_{in} (e.g., $\ell_{in} = \lambda$), we derive that $m < 0$, which makes the theorem meaningless. The reason behind this non-negligible loss is due to the fact that $(x_0, \ldots, x_d)$ may contain several bits of information about $f(X)$. Suppose that $\ell_{in} = \lambda$. In this case, it is possible that $(x_0, \ldots, x_d) = (a_0, \ldots, a_d)$ since Theorem 6 holds for every tuple of points $(x_0, \ldots, x_d)$ such that $x_i \in \{0,1\}^{\ell_{in}}$ for $i \in \{0, \ldots, d\}$. In this scenario, we can set $\alpha = \perp$ since the tuple $(x_0, \ldots, x_d)$ leaks the entire polynomial $f(X)$.[10] For this reason, to ensure that $m \geq 0$, we must set $\ell_{in} \leq \frac{(d+1)\lambda - c' - 2\log((d+1)\lambda - c') - 2\log(\lambda) - 2}{d+1} - 2\log(\ell_{in}) - 1$ (note that this implies $\ell_{in} < \lambda$). Hence, α must necessarily contain the bits of information about $f(X)$ that cannot be reclaimed from the points $(x_0, \ldots, x_d)$.

4 Definition of Verifiable Capacity-bound Functions

A VCBF forces an evaluator to read at least m distinct bits from its main memory. As explained in [54], the number of off-chip memory accesses impacts

[10] The example can be expressed in term of Kolmogorov complexity, i.e., $C(a_0, \ldots, a_d | x_0, \ldots, x_d) = 0$ where $x_i = a_i$ for $i \in \{0, \ldots, d\}$.

the energy consumption of the machine. If the cache's size is significantly smaller than m, evaluating the function requires significant resources. However, on the (honest) receiver's side, the validity of the computation can be verified efficiently in terms of capacity.

Formally, a VCBF scheme Π with input space $\{0,1\}^{\ell_{in}}$ is composed of the following polynomial-time algorithms:

$\mathsf{Setup}(1^\lambda, 1^k)$**:** Upon input the security parameter 1^λ and the capacity parameter 1^k, the randomized setup algorithm returns the evaluation key ek and the verification key vk.

$\mathsf{Eval}(\mathsf{ek}, x)$**:** Upon input the evaluation key ek and an input $x \in \{0,1\}^{\ell_{in}}$, the deterministic evaluation algorithm returns the output y and a proof π.

$\mathsf{Verify}(\mathsf{vk}, x, y, \pi)$**:** Upon input the verification key vk, an input $x \in \{0,1\}^{\ell_{in}}$, an output y, and a proof π, the deterministic verification algorithm returns a decisional bit b.

Here, the *capacity parameter* k regulates the actual capacity cost (the number of bits read by the evaluator).

Generally speaking, a VCBF scheme should satisfy four basic properties: *correctness*, *minimum capacity*, *soundness* and *capacity efficient verification*.

Correctness. Intuitively, a VCBF scheme is correct if the output of an honest execution of the evaluation algorithm is accepted by the verification algorithm.

Definition 7 (Correctness of VCBF). *A VCBF scheme Π with input space $\{0,1\}^{\ell_{in}}$ is correct if $\forall \lambda \in \mathbb{N}, \forall k \in \mathbb{N}$, $\forall x \in \{0,1\}^{\ell_{in}}$, we have:*

$$\Pr\left[\mathsf{Verify}(\mathsf{vk}, x, y, \pi) = 1 \,\middle|\, \begin{array}{l}(\mathsf{ek}, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, 1^k) \\ (y, \pi) = \mathsf{Eval}(\mathsf{ek}, x)\end{array}\right] = 1.$$

Minimum Capacity. The name captures the scheme's lower-bound on the number of distinct bits m that must be fetched from the main memory to evaluate the function. In more detail, on input a random challenge $x \leftarrow\!\!\$\ \{0,1\}^{\ell_{in}}$, the adversary A is asked to return the correct output $y = \mathsf{Eval}(\mathsf{ek}, x)$ while reading at most m bits from its main memory. We assume the main memory of A is bounded since there is a strict relationship between the memory available and A's advantage ϵ. Indeed, as discussed in Section 3, a viable adversarial strategy is to precompute a relatively large dictionary $\tau = (\mathsf{Eval}(\mathsf{ek}, x_1), \ldots, \mathsf{Eval}(\mathsf{ek}, x_n))$ (stored in the main memory) and return $\mathsf{Eval}(\mathsf{ek}, x)$, if x has been precomputed and included into τ. A larger memory would allow the adversary to store more precomputed values $\mathsf{Eval}(\mathsf{ek}, x_i)$, thus increasing the probability of success.

More formally, let $\tau \in \{0,1\}^n$ and $x \in \{0,1\}^{\ell_{in}}$ be the binary string representing the memory of the adversary A and a challenge, respectively. We denote with $\mathcal{I}_{\mathsf{A}(\tau,x;r)} = \{i_1, i_2, \ldots, i_{n'}\}_{n' \leq n}$ the set of n' distinct indexes read by A during the computation of the output $y = \mathsf{A}(\tau, x; r)$ for the corresponding challenge x while having access to memory τ and randomness $r \in \{0,1\}^{\ell_{rnd}}$. Intuitively,

on input the challenge x and randomness r, the adversary A fetches the binary string $\tau_{x,r} = b_{i_1}||\ldots||b_{i_{n'}}$ from τ (where b_i represents the i-th bit of τ and $\mathcal{I}_{\mathsf{A}(\tau,x;r)} = \{i_1, i_2, \ldots, i_{n'}\}$) and then compute the output y using the knowledge of $\tau_{x,r}$, x, and r.[11] A VCBF scheme is secure if the adversary does not compute the correct output $\mathsf{Eval}(\mathsf{ek}, x) \neq y = \mathsf{A}(\tau, x; r)$ when reading $|\mathcal{I}_{\mathsf{A}(\tau,x;r)}| = m$ bits from τ.[12] The formal definition is provided below.

Definition 8 (Minimum Capacity of VCBF). *Fix the keys* $(\mathsf{ek}, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, 1^k)$. *A VCBF scheme* Π *with input space* $\{0,1\}^{\ell_{in}}$ *satisfies* $(\epsilon, m, \ell_{rnd}, n)$*-min-capacity with respect to keys* $(\mathsf{ek}, \mathsf{vk})$ *if for all constant-size unbounded randomized adversaries* A *with randomness space* $\{0,1\}^{\ell_{rnd}}$ *and for all* $\tau \in \{0,1\}^n$*, we have:*

$$\Pr\left[\mathsf{Eval}(\mathsf{ek}, x) = y \wedge |\mathcal{I}_{\mathsf{A}(\tau,x;r)}| = m \,\middle|\, \begin{array}{l} x \leftarrow\!\!\$\ \{0,1\}^{\ell_{in}}, \\ y = \mathsf{A}(\tau, x; r) \end{array}\right] \leq \epsilon, \tag{1}$$

where $r \leftarrow\!\!\$\ \{0,1\}^{\ell_{rnd}}$.

Informally, Definition 8 states that if a VCBF scheme Π satisfies $(\epsilon, m, \ell_{rnd}, n)$-min-capacity then the only way for an adversary A to increase its advantage ϵ is to either read more than m distinct bits or have access to a memory larger than n bits (e.g., by storing in the memory $\tau \in \{0,1\}^n$ more precomputed values).

Relation between the memory size n and the advantage ϵ. Definition 8 is optimal in the sense that it does not put any constraint on the indexes $\mathcal{I}_{\mathsf{A}(\tau,x;r)}$ read by the adversary A. This means that A can arbitrarily access its memory. For example, it may perform multiple random accesses to the memory τ, i.e., perform one or more conditional jumps into specific memory indexes to read different portions of the memory). Hence, one (or more) couple of progressive indexes $\{i_j, i_{j+i}\} \subset \mathcal{I}_{\mathsf{A}(\tau,x;r)}$ may be not consecutive (i.e., $|i_j - i_{j+1}| > 1$).

The optimality of Definition 8 appears to be the primary (apparently insurmountable) obstacle when trying to relate the memory size n and the advantage ϵ. To retain an advantage ϵ, an adversary A may choose to store in the memory a precomputed data structure in which are stored (possibly partial) precomputed values about some evaluations $y = \mathsf{Eval}(\mathsf{ek}, x_i)$ of a subset of inputs $\mathcal{X} \subset \{0,1\}^{\ell_{in}}$ (e.g., precomputed dictionary). However, the estimation of the memory size n (required to store the data structure) highly depends on what type of precomputation is performed (e.g., the entropy of the precomputed values, the algorithm used, etc.) and on the type of encoding and memory access strategy used by A when fetching the data from memory τ to answer to an incoming challenge x. Unfortunately, this turned out to be a primary challenge when having block-box

[11] Observe that $\tau_{x,r}$ can be fetched from τ in an adaptive fashion according to the challenge x and randomness r.

[12] Without loss of generality, we assume the adversary reads exactly m bits since the higher the number of bits read, the higher the probability to compute the correct $y = \mathsf{Eval}(\mathsf{ek}, x)$.

access to A and working in the standard setting (i.e., no oracles, no idealized functionalities, no ROM).

As a foundation paper of VCBF, we initiate a fine-grained study regarding the level of minimum capacity that can be achieved according to specific classes of adversaries. In particular, we provide a feasibility result showing (in the concrete setting) the meaningful relation between parameters ϵ and n (using an information-theoretic approach) when dealing with the smaller class of adversaries $\mathcal{A}^{\text{1-access}}$. Such a class is composed by all the adversaries that perform exactly one (adaptive) random access to the memory τ, i.e., on input the memory $\tau \in \{0,1\}^n$, the challenge $x \in \{0,1\}^{\ell_{in}}$, and randomness $r \in \{0,1\}^n$, an adversary $\mathsf{A} \in \mathcal{A}^{\text{1-access}}$ adaptively jumps to an index $i \in [n-m+1]$ (memory location) and reads m consecutive indexes. Formally, when dealing with $\mathsf{A} \in \mathcal{A}^{\text{1-access}}$, the indexes $\mathcal{I}_{\mathsf{A}(\tau,x;r)} = \{i_1, \ldots, i_m\}$ read by A are consecutive, i.e., $i_j + 1 = i_{j+1}$ for $j \in [m-1]$.[13] Observe that in $\mathcal{A}^{\text{1-access}}$ we can identify several adversarial strategies used mainly in practice, e.g., precomputed dictionary attacks or any rainbow table technique that leverages a single adaptive random access.

As we will see during the security analysis of our construction (Section 5.1), by restricting the adversaries to the ones of the class $\mathcal{A}^{\text{1-access}}$, we can use a counting argument to concretely estimate the memory size n that an adversary $\mathsf{A} \in \mathcal{A}^{\text{1-access}}$ requires in order to retain a fixed advantage ϵ (Theorem 9). For completeness, we also include the results regarding the class $\mathcal{A}^{v\text{-access}}$ for $1 \leq v \leq m$, i.e., adversaries that perform exactly v (adaptive) random access to the memory (observe that Definition 8 coincides with Definition 9 when $\mathcal{A} = \bigcup_{i=1}^{m} \mathcal{A}^{i\text{-access}}$). However, due to the limited power of counting arguments, the memory size estimation n presents an exponential loss proportional to the number v of random accesses that $\mathsf{A} \in \mathcal{A}^{v\text{-access}}$ performs. In any case, this is enough to show that there exists a VCBF that satisfies $(\mathsf{negl}(\lambda), O((d+1)\lambda), o((d+1)\lambda), \mathsf{exp}(\lambda))$-min-capacity (in the asymptotic setting) with respect to the class of adversaries $\mathcal{A}^{O(1)\text{-access}}$. We now provide the formal security definition of minimum capacity with respect to a specific class of adversaries $\mathcal{A}$.

Definition 9 ($\mathcal{A}$-class minimum capacity of VCBF). *A VCBF scheme Π with input space $\{0,1\}^{\ell_{in}}$ satisfies $(\epsilon, m, \ell_{rnd}, n)$-min-capacity with respect to the class of adversaries $\mathcal{A}$ if Π satisfies $(\epsilon, m, \ell_{rnd}, n)$-min-capacity of Definition 8 where A is sampled from $\mathcal{A}$.*

To conclude, we stress that any scheme secure under Definitions 8-9 gives robust guarantees in terms of capacity (according to the corresponding class of adversaries) as we clarify below:

1. We cover adversaries with unbounded computational power as considered in information-theoretic cryptography. Hence, once the random challenge $x \in \{0,1\}^{\ell_{in}}$ is received, the adversary can spend an exponential amount of time to compute $y = \mathsf{Eval}(\mathsf{ek}, x)$ after reading $\tau_i \in \{0,1\}^m$ from the main memory.

[13] Without loss of generality, we assume that reading the first m bits of τ requires the adversary to perform a random access to the first index of τ.

2. A VCBF scheme guarantees a minimum capacity (for some parameters $(\epsilon, m, \ell_{rnd}, n)$) only if Equation (1) holds for every possible adversary A and memory τ after the instantiation of the scheme, i.e., the execution of $(\mathsf{ek}, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, 1^k)$. Hence, Definitions 8-9 provide a very strong level of security, since the adversary is allowed to select its strategy according to the setup result.[14]

Since we allow the adversary to select a strategy adaptively after the setup, it is critical to require the adversary to be of constant size. If we remove this constraint, Definitions 8-9 becomes meaningless since there always exists an adversary A_{ek} that can answer all challenges $x \in \{0,1\}^{\ell_{in}}$ by using ek hardcoded into its code. In a practical scenario, this means that a malicious manufacturer may defeat VCBFs by merely building a machine with no memory and ek hardcoded in its hardware.

Enforcing this constraint is reasonable, particularly in the context of ASIC resistance, where the cost of manufacturing a chip is proportional to its area [50,55]. Thus, embedding ek may be rendered uneconomic. Furthermore, one could periodically refresh $(\mathsf{ek}, \mathsf{vk})$ to force a manufacturer to build new machines or rely on external storage.

Standard vs. Ad-hoc Models. Our definition focuses on the minimum number of bits read from the main memory, while the definition of bandwidth hardness [55,15] focuses on the energy consumption rate. However, there is a more fundamental aspect to consider regarding Definitions 8-9: They do not rely on any heuristic assumptions, such as the Random Oracle (RO) or the Ideal Cipher [21], to measure the number of read bits. In fact, previous definitions of bandwidth-hard or memory-hard functions [55,15,5,2,19,4,3,21] do not directly measure the bits read by the evaluator. Instead, those models only calculate the number of the random oracle queries for each step. Thus, they only work within the RO model and take for granted that the random oracle's outputs are incompressible. However, the Kolmogorov theorem shows (at least from a theoretical point of view) that even a purely random string may have a certain probability of being compressed (See Theorem 3 and [41]). Therefore, the gap between RO queries and the actual number of bits read by the evaluator is artificially ignored in previous models. Also, we stress that both RO and Ideal Cipher definitions neglect (and do not take into account) the adversary's strategy in organizing and accessing specific portions of the memory. This is a fundamental aspect that needs to be considered when proving specific concrete memory bounds for VCBFs. In fact, this is the main obstacle that forces us to consider Definition 9 for the weaker class of adversaries $\mathcal{A}^{\text{1-access}}$. By fixing the class of adversaries to $\mathcal{A}^{\text{1-access}}$, we are able to give a meaningful estimation to memory size n (see Section 5.1) and prove that our polynomial-based VCBF satisfies min-capacity for parameters $(\epsilon, m, \ell_{rnd}, n)$ defined in Theorem 9.

[14] Note that if a VCBF scheme Π satisfies min-capacity for parameters $(\epsilon, m, \ell_{rnd}, n)$-w.r.t. Definition 8, then the very same level of min-capacity holds even if the adversary is sampled before the instantiation of the scheme $(\mathsf{ek}, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, 1^k)$.

Soundness. Soundness captures the infeasibility of convincing the verifier that $y^* \neq \mathsf{Eval}(\mathsf{ek}, x)$ is the correct output of the computation. In more detail, it is infeasible for a malicious evaluator to compute a triple (x^*, y^*, π^*) that verifies successfully, but y^* is not the correct output of the computation. Soundness is also fundamental to enforce the $(\epsilon, m, \ell_{rnd}, n)$-min-capacity (Definitions 8-9) of a VCBF scheme. For example, if soundness does not hold, a malicious evaluator can deceive the verifier by returning a proof π^* and an output $y^* \neq \mathsf{Eval}(\mathsf{ek}, x)$ such that $\mathsf{Verify}(\mathsf{vk}, x, y^*, \pi^*) = 1$. In this case, the energy consumption is not guaranteed since the value y^* is incorrect and may have been computed without fetching any bit from the main memory.

Below, we provide the formal definition of soundness.

Definition 10 (Soundness of VCBF). *A VCBF scheme Π with input space $\{0,1\}^{\ell_{in}}$ is (ϵ)-sound if for all PPT adversary A we have:*

$$\Pr\left[\begin{array}{c}\mathsf{Verify}(\mathsf{vk}, x, y, \pi) = 1 \quad \textit{and} \\ \mathsf{Eval}(\mathsf{ek}, x) \neq y\end{array} \middle| \begin{array}{l}(\mathsf{ek}, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, 1^k) \\ (x, y, \pi) \leftarrow\!\!\$\ \mathsf{A}(1^\lambda, \mathsf{ek}, \mathsf{vk})\end{array}\right] \leq \epsilon.$$

Capacity Efficient Verification. The resource considered by VCBFs is the capacity since an evaluator is forced to read m distinct bits from its main memory. The verifier, on the other hand, should not have the same workload. For this reason, we require a VCBF scheme Π to be efficiently verifiable:

Definition 11 (Capacity Efficient Verification of VCBF). *If Π satisfies $(\epsilon, m, \ell_{rnd}, n)$-min-capacity (either Definition 8 or Definition 9) then an honest execution of the verification algorithm requires at most fetching $o(m)$ bits from the memory (i.e., sublinear in m).*

In particular, in this work, the capacity parameter is of the form $m \in O((d+1)\lambda)$ where d is the degree of a polynomial $f(X) = \sum_{i=0}^{d} a_i \cdot x^i \in \mathbb{F}_p[x]$ and $\lambda+1$ is the size of the prime p. As we will see, to reach high capacities (such as GB or even TB), for a fixed λ we will have to set $d \in O(\lambda^c)$ for a constant $c \geq 1$. Nevertheless, the verification will be independent of d (by leveraging the publicly VC scheme of Elkhiyaoui et al. [31]). Hence, we will obtain at least $O(\lambda^{c+1})$ of min-capacity for the evaluation, and at most $O(\lambda)$ of min-capacity for the verification.[15]

On Energy Consumption. A motivation for VCBFs is ASIC resistance. State-of-the-art hash engines [14,55] could be $200,000\times$ faster and $40,000\times$ more energy efficient than multi-core CPUs. However, the energy consumption for off-chip memory accesses is similar for CPUs and ASICs [55]. If we assume the ASIC can hardcode only s bits, min-capacity guarantees that the ASIC will transfer at least $m - s$ bits from the external memory during the evaluation. If the energy cost is u nJ per bit for external memory accesses, the evaluation of the VCBF costs at least $u(m - s)$ nJ.

[15] In the verification, $O(\lambda)$ is for reading a constant number of group elements of order p of size at most $\lambda+1$. In the evaluation, $O((d+1)\lambda) = O(\lambda^{c+1})$ is for the d coefficients $(a_0, \ldots, a_d) \in \mathbb{F}_p^{d+1}$ of the polynomial $f(X) \in \mathbb{F}_p[x]$.

5 VCBF from VC for Polynomial Evaluation

In this section we show how to build a VCBF from VC for a family of polynomials $\mathcal{F}_{\lambda,d,p}$ defined as follows

$$\mathcal{F}_{\lambda,d,p} \stackrel{\text{def}}{=} \left\{ f_a(X) = \sum_{i=0}^{d} a_i \cdot x^i \mod p \;\middle|\; a = a_0||\ldots||a_d \right\}_{a\in\{0,1\}^{(d+1)\lambda}}, \quad (2)$$

where $\lambda \in \mathbb{N}$, $d \in \mathbb{N}$, and p is a prime of $\lambda + 1$ bits.

Construction 1 *Let* $\mathsf{VC} = (\mathsf{Setup}_{\mathsf{VC}}, \mathsf{ProbGen}_{\mathsf{VC}}, \mathsf{Compute}_{\mathsf{VC}}, \mathsf{Verify}_{\mathsf{VC}})$ *be a publicly VC scheme for the family of polynomials* $\mathcal{F}_{\lambda,d,p}$ *defined in Equation (2). We build a VCBF scheme with input space* $\{0,1\}^{\ell_{in}}$ *in the following way:*

$\mathsf{Setup}(1^\lambda, 1^k)$**:** *On input the security parameter* 1^λ *and the capacity parameter* 1^k*, the setup algorithm proceeds as follows:*
- *Without loss of generality, we assume* $k = (d+1)\lambda$ *for* $d \in \mathbb{N}$.
- *Samples* $a_0||\ldots||a_d = a \leftarrow\!\!\$\ \{0,1\}^{(d+1)\lambda}$ *where* $|a_i| = \lambda$ *for* $i \in \{0,\ldots,d\}$.

Then, it outputs the evaluation key $\mathsf{ek} = (\mathsf{ek}_{f_a}, \mathsf{vk}_{f_a})$ *and the verification key* $\mathsf{vk} = \mathsf{vk}_{f_a}$ *where* $(\mathsf{ek}_{f_a}, \mathsf{vk}_{f_a}) \leftarrow\!\!\$\ \mathsf{Setup}_{\mathsf{VC}}(1^\lambda, f_a)$ *and* $f_a \in \mathcal{F}_{\lambda,d,p}$ *as defined in Equation (2).*

$\mathsf{Eval}(\mathsf{ek}, x)$**:** *On input the evaluation key* $\mathsf{ek} = (\mathsf{ek}_{f_a}, \mathsf{vk}_{f_a})$ *and an input* $x \in \{0,1\}^{\ell_{in}}$*, the evaluation algorithm returns* $(y, \pi) = \mathsf{Compute}_{\mathsf{VC}}(\mathsf{ek}_{f_a}, \sigma_x)$ *where* $(\sigma_x, \mathsf{vk}_x) = \mathsf{ProbGen}_{\mathsf{VC}}(\mathsf{vk}_{f_a}, x)$.

$\mathsf{Verify}(\mathsf{vk}, x, y, \pi)$**:** *On input the verification key* $\mathsf{vk} = \mathsf{vk}_{f_a}$*, an input* $x \in \{0,1\}^{\ell_{in}}$*, an output* $y \in \mathcal{Y}$*, and a proof* π*, the verification algorithm returns* $b = \mathsf{Verify}_{\mathsf{VC}}(\mathsf{vk}_x, y, \pi)$ *where* $(\sigma_x, \mathsf{vk}_x) = \mathsf{ProbGen}_{\mathsf{VC}}(\mathsf{vk}_{f_a}, x)$.

In this scheme, the honest evaluator needs to read at least $k = (d+1)\lambda$ bits to load all the coefficients of the polynomial regardless of the cost of generating the proof π. The correctness follows directly from the correctness of the underlying schemes. For security and verification complexity, we establish the following results.

5.1 Security Analysis

The soundness is trivial. It follows from the (ϵ)-soundness of VC. The proof is standard, so we omit it.

Theorem 7 (Soundness). *If* VC *is* (ϵ)*-sound (Definition 2), then the VCBF scheme* Π *of Construction 1 with input space* $\{0,1\}^{\ell_{in}}$ *is* (ϵ)*-sound (Definition 10).*

Next, we show the level of minimum capacity that our VCBF scheme Π of Construction 1 satisfies with respect to the class of adversaries $\mathcal{A}^{v\text{-access}}$ (Definition 9) for $1 \leq v \leq m$. Our technique is divided into two parts: 1) We prove that Construction 1 satisfies an alternative definition of minimum capacity

dubbed *decomposed minimum capacity* and, 2) we demonstrate that any VCBF scheme satisfying the decomposed minimum capacity also satisfies the minimum capacity with respect to the class of adversaries $\mathcal{A}^{v\text{-access}}$ (Definition 9) for some paramters $(\epsilon, m, \ell_{rnd}, n)$. The second part is only required to give an upper bound to the memory size parameter n.

(Part one) Decomposed minimum capacity. The definition of decomposed minimum capacity is identical to Definition 8 except that the memory τ is decomposed into n independent strings $(\tau_1, \ldots, \tau_n)$ such that $\tau_i \in \{0,1\}^m$ for $i \in [n]$ (intuitively, each τ_i represents one possible string of length m that the adversary can read and interpret from its main memory, i.e., $(\tau_1, \ldots, \tau_n)$ is the *decomposition* of the main memory).[16] Then, the adversary succeeds if there exists $i \in [n]$ such that $y = \mathsf{A}(\tau_i, x; r_i)$ where $r_i \leftarrow\!\!\$\ \{0,1\}^{\ell_{rnd}}$ and $x \leftarrow\!\!\$\ \{0,1\}^{\ell_{in}}$.

Definition 12 (Decomposed minimum capacity of VCBF). *Fix the keys* $(\mathsf{ek}, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, 1^k)$. *A VCBF scheme Π with input space $\{0,1\}^{\ell_{in}}$ satisfies $(\epsilon, m, \ell_{rnd}, n)$-decomposed-min-capacity with respect to keys $(\mathsf{ek}, \mathsf{vk})$ if for all constant-size unbounded randomized adversaries A with randomness space $\{0,1\}^{\ell_{rnd}}$ and for all $(\tau_1, \ldots, \tau_n)$ such that $\tau_i \in \{0,1\}^m$ for $i \in [n]$, we have:*

$$\Pr\left[\exists i \in [n], \mathsf{Eval}(\mathsf{ek}, x) = y_i \;\middle|\; \begin{array}{c} x \leftarrow\!\!\$\ \{0,1\}^{\ell_{in}} \\ \{y_i \leftarrow\!\!\$\ \mathsf{A}(\tau_i, x)\}_{i\in[n]} \end{array}\right] \leq \epsilon. \tag{3}$$

Intuitively, Construction 1 satisfies the decomposed minimum capacity (Definition 12). For each string $\tau_i \in \{0,1\}^m$, the adversary can compute at most d distinct points $x \in \{0,1\}^{\ell_{in}}$ under the condition that the coefficients $(a_0, \ldots, a_d)$ of the polynomial $f_a(X) \in \mathcal{F}_{\lambda,d,p}$ are RND-incompressible (see Lemma 1). If this is not the case, it would violate Theorem 6 which states that a constant-size Turing machine needs more than m bits to compute $d+1$ points if the coefficients $(a_0, \ldots, a_d)$ are (c', ℓ_{rnd})-RND-incompressible. Since the adversary is only allowed to access n strings τ_i, the maximum number of the points the adversary can evaluate for $f_a(X)$ is nd. Based on our parameters, nd points are only a small fraction of the input space $\{0,1\}^{\ell_{in}}$. Then, we conclude the first part of the proof by showing that the string $a = a_0||\ldots||a_d$ (i.e., the coefficients of the polynomial $f_a(x)$ described by ek) is (c', ℓ_{rnd})-RND-incompressible with high probability. We report the formal result whose proof appears in Appendix A.4.

Theorem 8 (Decomposed minimum capacity). *Fix the keys* $(\mathsf{ek}, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, 1^k)$. *The VCBF scheme Π of Construction 1 with input space $\{0,1\}^{\ell_{in}}$ satisfies $(\epsilon, m, \ell_{rnd}, n)$-decomposed-min-capacity (Definition 12) with respect to*

[16] The decomposed minimum capacity definition can also be interpreted as a selective security version of minimum capacity in which each tape $\tau_i \in \{0,1\}^m$ corresponds to a fetched string according to a particular access pattern fixed before seeing the challenge $x \in \{0,1\}^{\ell_{in}}$.

keys $(\mathsf{ek}, \mathsf{vk})$ *where* $\lambda \in \mathbb{N}, d \in \mathbb{N}, c \in \mathbb{N}, \epsilon_1 \in [0,1]$,

$$\begin{aligned} m &= (d+1)(\lambda - \ell_{in} - 2\log(\ell_{in}) - 1) - c' \\ &\quad - \lambda - 2\log((d+1)\lambda - c') - 2\log(\lambda) - 2, \\ c' &= c + \ell_{rnd} + 2\log(\ell_{rnd}) + 1 + O(1), \\ \epsilon &= \epsilon_1 + \frac{d+1}{2^{\ell_{in}}} + \frac{1}{2^c} - \frac{1}{2^{(d+1)\lambda}}, \\ n &= \frac{\epsilon_1 \cdot 2^{\ell_{in}}}{d} + 1. \end{aligned}$$

(Part two) From Definition 12 to Definition 9. We now show that any VCBF that satisfies $(\epsilon, m, \ell_{rnd}, n-m+1)$-decomposed-min-capacity (Definition 12) also satisfies $(\epsilon, m, \ell_{rnd}, n)$-min-capacity with respect to the class of adversaries $\mathcal{A}^{\text{1-access}}$ (Definition 9). The result follows by using a counting argument: An adversary $\mathsf{A} \in \mathcal{A}^{\text{1-access}}$ with access to memory τ of length n can read at most $n-m+1$ different strings $(\tau_1, \ldots, \tau_{n-m+1})$ each of length m (as defined in Theorem 8). For completeness, we generalize the technique to every class of adversaries $\mathcal{A}^{v\text{-access}}$ for $1 \leq v \leq m$. However, due to the limited power of counting arguments, we stress that memory size n presents an exponential loss proportional to v. The proof is included in Appendix A.5.

Theorem 9 ($\mathcal{A}^{v\text{-access}}$-class minimum capacity). *Let $v \in \mathbb{N}$ and Π be a VCBF scheme with input space $\{0,1\}^{\ell_{in}}$. Fix the keys $(\mathsf{ek}, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, 1^k)$. If Π satisfies $(\epsilon, m, \ell_{rnd}, n_1)$-decomposed-min-capacity (Definition 12) with respect to keys $(\mathsf{ek}, \mathsf{vk})$ then Π satisfies $(\epsilon, m, \ell_{rnd}, n_2)$-min-capacity with respect to the class of adversaries $\mathcal{A}^{v\text{-access}}$ and keys $(\mathsf{ek}, \mathsf{vk})$ (Definition 9) where*

$$n_1 = \begin{cases} n_2 - m + 1 & \text{if } v = 1 \\ \frac{n_2!}{(n_2-v)!}\binom{m-1}{v-1} & \text{if } 1 < v \leq m. \end{cases}$$

The following corollaries report the concrete and asymptotic min-capacity of Construction 1. The concrete Corollary 1 (whose proof appears in Appendix A.6) shows that we can obtain a reasonable concrete tradeoff between ϵ and n only for the class of adversaries $\mathcal{A}^{\text{1-access}}$ (this is due to the limited power of our counting argument). In Section 5.3 we provide some examples of concrete instantiations. On the other hand, the asymptotic Corollary 2 shows that a secure VCBF exists with respect to the class of adversary $\mathcal{A}^{O(1)\text{-access}}$. We stress that this must be interpreted as a purely theoretical result showing the feasibility of VCBF since the constants hidden by the asymptotic notation are significantly large.

Corollary 1. *For any $v \in \mathbb{N}$, the VCBF scheme Π of Construction 1 with input space $\{0,1\}^{\ell_{in}}$ satisfies $(\epsilon, m, \ell_{rnd}, n)$-min-capacity (Definition 9) with respect to*

the class of adversaries $\mathcal{A}^{v\text{-}access}$ *where* $\lambda \in \mathbb{N}, d \in \mathbb{N}, c \in \mathbb{N}, \epsilon_1 \in [0,1]$,

$$
\begin{aligned}
m &= (d+1)(\lambda - \ell_{in} - 2\log(\ell_{in}) - 1) - c' \\
&\quad - \lambda - 2\log((d+1)\lambda - c') - 2\log(\lambda) - 2, \\
c' &= c + \ell_{rnd} + 2\log(\ell_{rnd}) + 1 + O(1), \\
\epsilon &= \epsilon_1 + \frac{d+1}{2^{\ell_{in}}} + \frac{1}{2^c} - \frac{1}{2^{(d+1)\lambda}}, \\
n &= \begin{cases} m + \frac{\epsilon_1 \cdot 2^{\ell_{in}}}{d} & \text{if } v = 1 \\ \sqrt[v]{\left(\frac{\epsilon_1 \cdot 2^{\ell_{in}}}{d} + 1\right) / \left(v! \left(\frac{m-1}{v-1}\right)^{v-1}\right)} \cdot v & \text{if } 1 < v \leq m. \end{cases}
\end{aligned}
$$

Corollary 2. *For any* $\lambda \in \mathbb{N}$ *and* $k = (d+1)\lambda \in \mathbb{N}$ *such that* $d \in \mathbb{N}$*, there exists a VCBF that satisfies* $(\mathsf{negl}(\lambda), O((d+1)\lambda), o((d+1)\lambda), \mathsf{exp}(\lambda))$*-min-capacity with respect to the class of adversaries* $\mathcal{A}^{O(1)\text{-}access}$ *.*

Proof. The corollary follows directly by setting $v \in O(1)$, $\ell_{in} \in [\omega(\log(\lambda)), o(\lambda)]$, $\ell_{rnd} \in o((d+1)\lambda)$, and $c \in [\omega(\log(\lambda)), O((d+1)\lambda)]$ in Corollary 1. □

Remark 2. We stress that, independently from the class of adversaries $\mathcal{A}^{v\text{-access}}$ considered, the evaluation of a random incompressible polynomial $f(X) \in \mathbb{F}[x]_p$ (as for Construction 1) requires reading m distinct bits (see Theorem 6 and Remark 1). For this reason, it is reasonable to believe that Construction 1 satisfies $(\epsilon, m, \ell_{rnd}, n)$-min-capacity with respect to any class of adversaries $\mathcal{A}^{v\text{-access}}$ (for $v > 1$) where the concrete memory size parameter n is significantly larger than the one of Corollary 1 (possibly reducing the exponential loss proportional to v). To support the aforementioned observation, consider an adversary $\mathsf{A} \in \mathcal{A}^{v\text{-access}}$ with access to a precomputed memory $\tau \in \{0,1\}^n$ whose size n is enough only to encode in a meaningful way the (possibly partial) information regarding the evaluations of a relatively small (not exponential) subset of points $\mathcal{X} \subset \{0,1\}^{\ell_{in}}$, i.e., $|\mathcal{X}| \ll 2^{\ell_{in}}$. It is clear that A has no choice other than evaluating the random polynomial $f(x) = y \in \mathbb{F}_p$ (by reading m bits as defined by Theorem 6) if $x \notin \mathcal{X}$ (i.e., the challenge does not belong to the small subset $\mathcal{X}$ considered during the precomputation of $\tau \in \{0,1\}^n$). However, the estimation of the concrete memory size n according to the number of points $|\mathcal{X}|$ (considered during the precomputation) appears to be a primary challenge when having only black-box access to adversaries that perform multiple random access (e.g., $\mathcal{A}^{v\text{-access}}$ for $v > 1$) while using a purely information-theoretical approach. Indeed, counting all the possible combinations of bits that an adversary $\mathcal{A}^{v\text{-access}}$ can read from its memory (see proof of Theorem 9) only allows to prove a loose bound: Not all combinations of bits (of the memory) are valid and allow the adversary to extract meaningful information.

5.2 Verification Complexity

Corollary 2 shows that an evaluator needs to read at least $O((d+1)\lambda)$ distinct bits from its main memory. We now analyze the verifier capacity complexity. By inspecting Construction 1, we observe that the complexity of Verify

Table 1: Example of concrete instantiations of Corollary 1 with respect to the class of adversaries $\mathcal{A}^{\text{1-access}}$ for $k = 1\text{GB}$ and $k = 100\text{GB}$ where $\ell_{in} = \ell_{rnd} = c = 128$ and $k = (d+1)\lambda$.

Capacity Param. k	**Prime Size** $\lambda+1$	**Advantage Param.** ϵ_1	
		2^{-90}	2^{-50}
1GB	512 ($d \approx 15.65 \cdot 10^6 \approx \lambda^{2.66}$)	$\epsilon \approx 2^{-90}$, $n \approx 0.69\text{GB}$, $m \approx 0.69\text{GB}$	$\epsilon \approx 2^{-50}$, $n \approx 2.14\text{PB}$, $m \approx 0.69\text{GB}$
	1024 ($d \approx 78.20 \cdot 10^5 \approx \lambda^{2.29}$)	$\epsilon \approx 2^{-90}$, $n \approx 0.82\text{GB}$, $m \approx 0.82\text{GB}$	$\epsilon \approx 2^{-50}$, $n \approx 4.29\text{PB}$, $m \approx 0.82\text{GB}$
100GB	512 ($d \approx 15.65 \cdot 10^8 \approx \lambda^{3.39}$)	$\epsilon \approx 2^{-90}$, $n \approx 67.25\text{GB}$, $m \approx 67.25\text{GB}$	$\epsilon \approx 2^{-50}$, $n \approx 22.06\text{TB}$, $m \approx 67.25\text{GB}$
	1024 ($d \approx 78.20 \cdot 10^7 \approx \lambda^{2.95}$)	$\epsilon \approx 2^{-90}$, $n \approx 80.20\text{GB}$, $m \approx 80.20\text{GB}$	$\epsilon \approx 2^{-50}$, $n \approx 44.02\text{TB}$, $m \approx 80.20\text{GB}$

coincides with the ones of algorithms $\mathsf{ProbGen}_{\mathsf{VC}}$ and $\mathsf{Verify}_{\mathsf{VC}}$ of the underlying VC scheme. Therefore, we must consider a concrete instantiation of the VC scheme. For this reason, we measured the efficiency of our VCBF w.r.t. the VC scheme of Elkhiyaoui et al. [31], which uses an asymmetric bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. The execution of $\mathsf{ProbGen}_{\mathsf{VC}}(\mathsf{vk}_f, x)$ computes and returns $\mathsf{vk}_x = (\mathsf{vk}_x^0, \mathsf{vk}_x^1) = (b_0 \cdot g^{x^2}, r_1^x \cdot r_0)$ and $\sigma_x = x$ where $\mathsf{vk}_f = (b_0, r_1, f_0) \in \mathbb{G}_1 \times \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_2$, $x \in \mathbb{F}_p$, and g the generator of $\mathbb{G}_1$. Moreover, $\mathsf{Verify}_{\mathsf{VC}}(\mathsf{vk}_x, y, \pi_y)$ checks the correctness of the computation by computing:

$$e(g, h^y) \stackrel{?}{=} e(\mathsf{vk}_x^0, \pi_y) \cdot e(g, \mathsf{vk}_x^1),$$

where $\mathsf{vk}_x = (\mathsf{vk}_x^0, \mathsf{vk}_x^1)$ and h is a generator of $\mathbb{G}_2$. Hence, in the worst case, the verification capacity complexity of our VCBF is $O(\lambda) \in o((d+1)\lambda)$, while the verification time is $O(1)$ in the number of group operations.

This because the executions of $\mathsf{ProbGen}_{\mathsf{VC}}$ and $\mathsf{Verify}_{\mathsf{VC}}$ are independent of the polynomial degree d in terms of both capacity and time.

5.3 Concrete Instantiation

Table 1 shows two candidate instantiations satisfying the requirements of Corollary 1 for the capacity parameter $k = 1\text{GB}$ and $k = 100\text{GB}$ when considering the class of adversaries $\mathcal{A}^{\text{1-access}}$. To calculate them, we first set the intermediate parameter ϵ_1 as 2^{-90} and 2^{-50}, respectively. Then, we compute the adversary's advantage ϵ, the bound m of minimum capacity, and the memory size n according to Corollary 1. Observe that the bound m of Corollary 1 contains a constant $O(1)$ which represents the tolerance that depends on the constant-size Turing machine implementing $\mathsf{A} \in \mathcal{A}^{\text{1-access}}$.

The efficient verification algorithm allows our VCBF scheme to handle very large capacities (i.e., $d \gg \lambda$) since its complexity is independent of the degree of the polynomial d. Note that d is the dominating factor of the minimum capacity

bound m. For example, for $\lambda+1 = 512$ and $k = (d+1)\lambda = 1\text{GB}$, we need to set the polynomial degree to $d = 15.65 \cdot 10^6 \approx \lambda^{2.66}$ (Table 1). Hence, the verifier complexity is $d \approx \lambda^{2.66}$ times smaller than the evaluator complexity, i.e., at most $O(\lambda)$ verification capacity versus at least $O(\lambda^{3.66})$ minimum capacity.

There is always a loss between the capacity parameter k (the number of bits read by an honest evaluator) and the provable minimum capacity m. Such a loss is inevitable due to Theorem 6 (see Section 3 for additional details). To reduce this gap, one could reduce the input size ℓ_{in}, but this increases the advantage ϵ of the adversary. Alternatively, we recommend to increase the security parameter λ and get a larger prime p. For example, by increasing $\lambda + 1$ from 512 to 1024 bits, the lower-bound m goes from 0.69GB to 0.82GB (resp. from 67.25GB to 80.20GB) for $k = 1\text{GB}$ (resp. for $k = 100\text{GB}$).

Lastly, as we discussed in Section 4, the advantage ϵ of the adversary is highly correlated with the memory size n (see Corollary 1). If the adversary involves more memory (e.g., increase n from $\approx 80.20\text{GB}$ to $\approx 22.06\text{TB}$ for $k = 100\text{GB}$ and $\lambda + 1 = 512$), the advantage ϵ could be significantly improved from $\approx 2^{-90}$ to $\approx 2^{-50}$. This trade-off is inevitable. For example, an adversary may involve its memory only to store precomputed dictionary, i.e., it encodes in the memory τ of size n different precomputed polynomial evaluations $f(x_j)$ for arbitrary points $x_j \in \{0,1\}^{\ell_{in}}$. Intuitively, the larger the memory n, the more the precomputed polynomial evaluations $f(x_j)$, the higher the advantage ϵ. Analogously, for very small advantages ϵ, the memory size n decreases close to m (see column $\epsilon_1 = 2^{-90}$ of Table 1 and Corollary 1): An adversary needs a small memory in order to retain very small advantages. This also relates to Theorem 6: A binary string of size m may allow an adversary to answer to at most d points. Hence, a memory size $n = m$ is enough for an adversary to retain a very small advantage $\epsilon \approx d/2^{\ell_{in}}$. Concretely, for $k = 100\text{GB}$ and $d = 15.65 \cdot 10^8$ ($\lambda + 1 = 512$) and $\ell_{in} = 128$ (as in Table 1), a memory of size $n = m$ may permit to retain an advantage $\epsilon \approx d/2^{\ell_{in}} \approx 2^{-99}$.

6 Future Work

This work lays the foundation for verifiable capacity-bound functions (VCBFs). We have demonstrated the level of minimum capacity a VCBF can achieve when only leveraging an information-theoretical approach based on Kolmogorov complexity.

We believe this work opens up several new directions that could be investigated further. For example, a natural next step is to build new VCBFs (or extend our polynomial based construction) in stronger security models (e.g., to provide better concrete bounds between the adversary's advantage ϵ and memory size n for other classes of adversaries $\mathcal{A}^{v\text{-access}}$ for $v > 1$. See Remark 2) by combining our results together with additional (computational) assumptions. Also, it would be relevant to analyze the minimum capacity of VCBF schemes in a bounded computational model that may provide better bounds in terms of minimum capacity.

We have also shown that some classes of polynomials have high Kolmogorov complexity and require a large amount of space to be evaluated. Identifying (or even building) other high Kolmogorov complexity functions may yield new VCBFs.

Finally, our approach based on Kolmogorov complexity could result in new analysis frameworks that give prominence to space lower bounds for various cryptographic schemes – an aspect that is often overlooked in cryptography.

References

1. Abadi, M., Burrows, M., Manasse, M., Wobber, T.: Moderately hard, memory-bound functions. ACM Transactions on Internet Technology (TOIT) **5**(2), 299–327 (2005)
2. Alwen, J., Blocki, J.: Efficiently computing data-independent memory-hard functions. In: Annual International Cryptology Conference. pp. 241–271. Springer (2016)
3. Alwen, J., Blocki, J., Harsha, B.: Practical graphs for optimal side-channel resistant memory-hard functions. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1001–1017 (2017)
4. Alwen, J., Chen, B., Pietrzak, K., Reyzin, L., Tessaro, S.: Scrypt is maximally memory-hard. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 33–62. Springer (2017)
5. Alwen, J., Serbinenko, V.: High parallel complexity graphs and memory-hard functions. In: Proceedings of the forty-seventh annual ACM symposium on Theory of computing. pp. 595–603 (2015)
6. Ateniese, G., Bonacina, I., Faonio, A., Galesi, N.: Proofs of space: When space is of the essence. In: International Conference on Security and Cryptography for Networks. pp. 538–557. Springer (2014)
7. Aura, T., Nikander, P., Leiwo, J.: Dos-resistant authentication with client puzzles. In: International workshop on security protocols. pp. 170–177. Springer (2000)
8. Back, A.: Hashcash-a denial of service counter-measure (2002)
9. Bellare, M., Kane, D., Rogaway, P.: Big-key symmetric encryption: Resisting key exfiltration. In: Annual International Cryptology Conference. pp. 373–402. Springer (2016)
10. Biryukov, A., Bouillaguet, C., Khovratovich, D.: Cryptographic schemes based on the asasa structure: Black-box, white-box, and public-key. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 63–84. Springer (2014)
11. Biryukov, A., Khovratovich, D.: Egalitarian computing. In: Holz, T., Savage, S. (eds.) 25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016. pp. 315–326. USENIX Association (2016), https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/biryukov
12. Biryukov, A., Perrin, L.: Symmetrically and asymmetrically hard cryptography. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 417–445. Springer (2017)
13. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: From extractable collision resistance to succinct non-interactive arguments of knowledge, and

back again. In: Goldwasser, S. (ed.) Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012. pp. 326–349. ACM (2012). https://doi.org/10.1145/2090236.2090263, https://doi.org/10.1145/2090236.2090263
14. Bitmain: Antminer s9 (2020), https://shop.bitmain.com/product/detail?pid=00020200306153650096S2W5mY1i0661
15. Blocki, J., Ren, L., Zhou, S.: Bandwidth-hard functions: Reductions and lower bounds. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 1820–1836 (2018)
16. Bogdanov, A., Isobe, T.: White-box cryptography revisited: Space-hard ciphers. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. pp. 1058–1069 (2015)
17. Bogdanov, A., Isobe, T., Tischhauser, E.: Towards practical whitebox cryptography: optimizing efficiency and space hardness. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 126–158. Springer (2016)
18. Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: Annual International Cryptology Conference. pp. 757–788. Springer (2018)
19. Boneh, D., Corrigan-Gibbs, H., Schechter, S.: Balloon hashing: A memory-hard function providing provable protection against sequential attacks. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 220–248. Springer (2016)
20. Canetti, R., Halevi, S., Steiner, M.: Hardness amplification of weakly verifiable puzzles. In: Theory of Cryptography Conference. pp. 17–33. Springer (2005)
21. Chen, B., Tessaro, S.: Memory-hard functions from cryptographic primitives. In: Annual International Cryptology Conference. pp. 543–572. Springer (2019)
22. Chen, L., Morrissey, P., Smart, N.P., Warinschi, B.: Security notions and generic constructions for client puzzles. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 505–523. Springer (2009)
23. Cohen, B., Pietrzak, K.: Simple proofs of sequential work. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 451–467. Springer (2018)
24. Dean, D., Stubblefield, A.: Using client puzzles to protect tls. In: USENIX Security Symposium. vol. 42 (2001)
25. Dinur, I.: On the streaming indistinguishability of a random permutation and a random function. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 433–460. Springer (2020)
26. Döttling, N., Lai, R.W., Malavolta, G.: Incremental proofs of sequential work. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 292–323. Springer (2019)
27. Dwork, C., Goldberg, A., Naor, M.: On memory-bound functions for fighting spam. In: Annual International Cryptology Conference. pp. 426–444. Springer (2003)
28. Dwork, C., Naor, M.: Pricing via processing or combatting junk mail. In: Annual International Cryptology Conference. pp. 139–147. Springer (1992)
29. Dwork, C., Naor, M., Wee, H.: Pebbling and proofs of work. In: Annual International Cryptology Conference. pp. 37–54. Springer (2005)
30. Dziembowski, S., Faust, S., Kolmogorov, V., Pietrzak, K.: Proofs of space. In: Annual Cryptology Conference. pp. 585–605. Springer (2015)
31. Elkhiyaoui, K., Önen, M., Azraoui, M., Molva, R.: Efficient techniques for publicly verifiable delegation of computation. In: Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security. pp. 119–128. ACM (2016)

32. Ephraim, N., Freitag, C., Komargodski, I., Pass, R.: Sparks: Succinct parallelizable arguments of knowledge. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12105, pp. 707–737. Springer (2020). https://doi.org/10.1007/978-3-030-45721-1_25, https://doi.org/10.1007/978-3-030-45721-1_25
33. Fiore, D., Gennaro, R.: Publicly verifiable delegation of large polynomials and matrix computations, with applications. In: Yu, T., Danezis, G., Gligor, V.D. (eds.) the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012. pp. 501–512. ACM (2012). https://doi.org/10.1145/2382196.2382250, https://doi.org/10.1145/2382196.2382250
34. Fouque, P.A., Karpman, P., Kirchner, P., Minaud, B.: Efficient and provable white-box primitives. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 159–188. Springer (2016)
35. Gentry, C., Wichs, D.: Separating succinct non-interactive arguments from all falsifiable assumptions. In: Fortnow, L., Vadhan, S.P. (eds.) Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011. pp. 99–108. ACM (2011). https://doi.org/10.1145/1993636.1993651, https://doi.org/10.1145/1993636.1993651
36. Ghoshal, A., Jaeger, J., Tessaro, S.: The memory-tightness of authenticated encryption. In: Annual International Cryptology Conference. pp. 127–156. Springer (2020)
37. Ghoshal, A., Tessaro, S.: On the memory-tightness of hashed elgamal. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 33–62. Springer (2020)
38. Jaeger, J., Tessaro, S.: Tight time-memory trade-offs for symmetric encryption. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 467–497. Springer (2019)
39. Juels, A.: Client puzzles: A cryptographic countermeasure against connection depletion attacks. In: Proc. Networks and Distributed System Security Symposium (NDSS), 1999 (1999)
40. Kaliski, B.: Password-based cryptography specification. RFC 2898 (2000)
41. Li, M., Vitányi, P., et al.: An introduction to Kolmogorov complexity and its applications, vol. 3. Springer (2008)
42. Liu, Y., Pass, R.: On one-way functions and kolmogorov complexity. In: FOCS 2020, 61st Annual IEEE Symposium on Foundations of Computer Science
43. Mahmoody, M., Moran, T., Vadhan, S.: Publicly verifiable proofs of sequential work. In: Proceedings of the 4th conference on Innovations in Theoretical Computer Science. pp. 373–388 (2013)
44. Merkle, R.C.: Secure communications over insecure channels. Communications of the ACM **21**(4), 294–299 (1978)
45. Muchnik, A.A.: Kolmogorov complexity and cryptography. Proceedings of the Steklov Institute of Mathematics **274**(1), 193 (2011)
46. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
47. Neary, T., Woods, D.: Four small universal turing machines. Fundamenta Informaticae **91**(1), 123–144 (2009)
48. Papamanthou, C., Shi, E., Tamassia, R.: Signatures of correct computation. In: Sahai, A. (ed.) Theory of Cryptography - 10th Theory of Cryp-

tography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7785, pp. 222–242. Springer (2013). https://doi.org/10.1007/978-3-642-36594-2_13, https://doi.org/10.1007/978-3-642-36594-2_13
49. Parno, B., Raykova, M., Vaikuntanathan, V.: How to delegate and verify in public: Verifiable computation from attribute-based encryption. In: Theory of Cryptography Conference. pp. 422–439. Springer (2012)
50. Percival, C.: Stronger key derivation via sequential memory-hard functions (2009)
51. Pietrzak, K.: Simple verifiable delay functions. In: 10th innovations in theoretical computer science conference (itcs 2019). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2018)
52. Provos, N., Mazières, D.: A future-adaptable password scheme. In: Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference, June 6-11, 1999, Monterey, California, USA. pp. 81–91. USENIX (1999), http://www.usenix.org/events/usenix99/provos.html
53. Reiter, M.K., Sekar, V., Spensky, C., Zhang, Z.: Making peer-assisted content distribution robust to collusion using bandwidth puzzles. In: International Conference on Information Systems Security. pp. 132–147. Springer (2009)
54. Ren, L., Devadas, S.: Proof of space from stacked expanders. In: Theory of Cryptography Conference. pp. 262–285. Springer (2016)
55. Ren, L., Devadas, S.: Bandwidth hard functions for asic resistance. In: Theory of Cryptography Conference. pp. 466–492. Springer (2017)
56. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto (1996)
57. Souto, A., Teixeira, A., Pinto, A.: One-way functions using kolmogorov complexity. Proceedings of the Computability in Europe pp. 346–356 (2010)
58. Stebila, D., Kuppusamy, L., Rangasamy, J., Boyd, C., Nieto, J.G.: Stronger difficulty notions for client puzzles and denial-of-service-resistant protocols. In: Cryptographers' Track at the RSA Conference. pp. 284–301. Springer (2011)
59. Tessaro, S., Thiruvengadam, A.: Provable time-memory trade-offs: symmetric cryptography against memory-bounded adversaries. In: Theory of Cryptography Conference. pp. 3–32. Springer (2018)
60. Wesolowski, B.: Efficient verifiable delay functions. Journal of Cryptology pp. 1–35 (2020)
61. Woods, D., Neary, T.: The complexity of small universal turing machines: A survey. Theoretical Computer Science **410**(4-5), 443–450 (2009)
62. Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., Papamanthou, C.: vsql: Verifying arbitrary SQL queries over dynamic outsourced databases. In: 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017. pp. 863–880. IEEE Computer Society (2017). https://doi.org/10.1109/SP.2017.43, https://doi.org/10.1109/SP.2017.43
63. Zhang, Z.: A new bound on the performance of the bandwidth puzzle. IEEE Transactions on Information Forensics and Security **7**(2), 731–742 (2012)

A Supporting Proofs

A.1 Proof of Theorem 4

Let $\mathcal{Y} \subseteq \mathcal{X}$ be the set composed of all strings $x \in \mathcal{X}$ that are c-DET-incompressible. By using Theorem 3, we conclude that $\mathcal{Y}$ is of size at least $m(1-2^{-c})+1$. Hence,

the probability that $x \leftarrow\!\!\$\ \mathcal{X}$ is c-DET-incompressible is defined as follows:

$$\Pr[x \text{ is } c\text{-DET-incompressible} \mid x \leftarrow\!\!\$\ \mathcal{X}] = \frac{|\mathcal{Y}|}{|\mathcal{X}|} \geq \frac{m(1-2^{-c})+1}{m} = 1-2^{-c}+\frac{1}{m}.$$

This concludes the proof.

A.2 Proof of Theorem 5

The proof follows the same strategy of [41, Lemma 2.1.1]. Let $e = \ell_{rnd} + 2\log(\ell_{rnd})+1$. By contradiction assume that, for any constant $v \in O(1)$, x is not $(c+e+v, \ell_{rnd})$-RND-incompressible, i.e., there exists a constant-size unbounded randomized Turing machine T, a string $\alpha \in \{0,1\}^{|x|-c-e-v}$, and $r \in \{0,1\}^{\ell_{rnd}}$ such that $\Pr[\mathsf{T}(\alpha; r) = x] = 1$. Let T' be the deterministic Turing machine defined as $\mathsf{T}'(\alpha, r) \stackrel{\text{def}}{=} \mathsf{T}(\alpha; r)$. Without loss of generality, assume that T' is the i-th deterministic machine with respect to the enumeration defined by a universal deterministic Turing machine U, i.e.,

$$\mathsf{U}(\langle i, r, \alpha\rangle) = \mathsf{T}'_i(\alpha, r) = \mathsf{T}'(\alpha, r) = \mathsf{T}(\alpha; r).$$

The Kolmogorov complexity of x with respect to the deterministic reference universal Turing machine U is

$$\begin{aligned} C_{\mathsf{U}}(x) &= |\langle i, r, \alpha\rangle| \leq \log(i) + 2\log\log(i) + 1 + \ell_{rnd} + 2\log(\ell_{rnd}) + 1 + |\alpha| \\ &= \log(i) + 2\log\log(i) + 1 + e + |\alpha| < |x| - c - v + \log(i) + 2\log\log(i) + 1. \end{aligned}$$

If we set $v = \log(i)+2\log\log(i)+1 \in O(1)$ we obtain that $C_{\mathsf{U}}(x) < |x|-c$. (Note that v is a constant that only depends on the enumeration U.) This contradicts the fact that x is c-DET-incompressible.

A.3 Proof of Theorem 6

Suppose there exists a constant-size randomized unbounded Turing machine T with randomness space $\{0,1\}^{\ell_{rnd}}$, a string $\alpha \in \{0,1\}^m$, a randomness $r \in \{0,1\}^{\ell_{rnd}}$, and a tuple $(x_0, \ldots, x_d)$ such that $\underset{i \neq j}{\forall} i, j \in \{0, \ldots, d\}$, $x_i \neq x_j$ and $x_i \in \{0,1\}^{\ell_{in}}$ for which the following probability holds

$$\Pr[(f(x_0), \ldots, f(x_d)) = \mathsf{T}(\alpha, x_0, \ldots, x_d; r)] = 1.^{17}$$

Then, we show that a is not (c', ℓ_{rnd})-RND-incompressible (Definition 6). Let $\alpha' = \langle p, \alpha, x_0, \ldots, x_d\rangle$ and consider the following Turing machine T':

$\mathsf{T}'(\alpha'; r')$**:** On input $\alpha' = \langle p, \alpha, x_0, \ldots, x_d\rangle$ and a randomness $r' \in \{0,1\}^{\ell_{rnd}}$, T' proceeds as follows:

[17] Note that the randomness is made explicit. Hence, the probability is equal to either 0 or 1.

1. Interpret the input $\alpha' = \langle p, \alpha, x_0, \ldots, x_d \rangle$ and separate the inputs p, α and $x_0, \ldots, x_d$.
2. Compute $(f(x_0), \ldots, f(x_d)) = \mathsf{T}(\alpha, x_0, \ldots, x_d; r')$.
3. Reconstruct the polynomial via interpolation.
4. Return $a = a_0 || \ldots || a_d$.

The steps executed by T' can be implemented by a constant number of operations if T receives in input all the pieces of information on which the computation depends on. That is, $(p, \alpha, x_0, \ldots, x_d, r, \mathsf{T}, d)$ and the structure of the polynomial. Note that the values $(p, \alpha, x_0, \ldots, x_d)$ are δ-encoded (Section 2.1) into α', r' is taken as input, and the degree d can be computed by counting the points $(x_0, \ldots, x_d)$ and decrease that amount by 1 (i.e., $|\{x_0, \ldots, x_d\}| - 1 = d$). Moreover, T and the structure of $f(X)$ are of constant size and can be hardcoded into the code of T'. Hence, we can conclude that T' is of constant size.

In addition, it is easy to see that T', on input α', correctly computes a, if $r' = r$. We can bound the size of α' as follows:

$$\begin{aligned}
|\alpha'| &= |\langle p, \alpha, x_0, \ldots, x_d \rangle| \\
&\leq \lambda + 2\log(\lambda) + m + 2\log(m) + (d+1)(\ell_{in} + 2\log(\ell_{in}) + 1) + 2 \\
&\leq (d+1)\lambda - c' + 2\log(m) - 2\log((d+1)\lambda - c') \\
&< \log(a) - c' \leq (d+1)\lambda - c',
\end{aligned}$$

where we used the fact that $|\alpha| \leq m = (d+1)(\lambda - \ell_{in} - 2\log(\ell_{in}) - 1) - c' - \lambda - 2\log((d+1)\lambda - c') - 2\log(\lambda) - 2$ and $\log(m) < \log((d+1)\lambda - c')$. This contradicts the fact that a is c'-RND-incompressible with respect to randomized Turing machines with randomness space $\{0,1\}^{\ell_{rnd}}$.

A.4 Proof of Theorem 8

Let $f_a(x) \in \mathcal{F}_{\lambda,d,p}$ be the polynomial sampled by Setup (as defined in Construction 1). Let $E^{\mathrm{RND}}_{c',\ell_{rnd}}$ be the event that the string $a = a_0 || \ldots || a_d$ (the coefficients of the polynomial $f_a(x)$ described by ek) is (c', ℓ_{rnd})-RND-incompressible (Definition 6). We first prove the following lemma.

Lemma 1. *If $E^{RND}_{c',\ell_{rnd}}$ holds, then the VCBF scheme Π of Construction 1 with input space $\{0,1\}^{\ell_{in}}$ satisfies $(\epsilon', m, \ell_{rnd}, n)$-decomposed-min-capacity (Definition 12) with respect to keys $(\mathsf{ek}, \mathsf{vk})$ where $\lambda \in \mathbb{N}, d \in \mathbb{N}, c \in \mathbb{N}, \epsilon_1 \in [0,1]$,*

$$\begin{aligned}
m &= (d+1)(\lambda - \ell_{in} - 2\log(\ell_{in}) - 1) - c' \\
&\qquad - \lambda - 2\log((d+1)\lambda - c') - 2\log(\lambda) - 2, \\
c' &= c + \ell_{rnd} + 2\log(\ell_{rnd}) + 1 + O(1), \\
\epsilon' &= \epsilon_1 + \frac{d+1}{2^{\ell_{in}}}, \\
n &= \frac{\epsilon_1 \cdot 2^{\ell_{in}}}{d} + 1.
\end{aligned}$$

Proof. Assume there exists a constant-size randomized unbounded adversary A with randomness space $\{0,1\}^{\ell_{rnd}}$ and a tuple $(\tau_1, \ldots, \tau_n)$ such that $\tau_i \in \{0,1\}^m$ for $i \in [n]$ for which the following probability holds:

$$\Pr\left[\exists i \in [n], \mathsf{Eval}(\mathsf{ek}, x) = y_i \middle| \begin{array}{c} E^{\mathrm{RND}}_{c',\ell_{rnd}}, \\ x \leftarrow\!\!\$\ \{0,1\}^{\ell_{in}}, \\ \{y_i \leftarrow\!\!\$\ \mathsf{A}(\tau_i, x)\}_{i\in[n]} \end{array}\right] =$$

$$\Pr\left[\exists i \in [n], \mathsf{Eval}(\mathsf{ek}, x) = y_i \middle| \begin{array}{c} E^{\mathrm{RND}}_{c',\ell_{rnd}}, \\ x \leftarrow\!\!\$\ \{0,1\}^{\ell_{in}}, \\ \{y_i = \mathsf{A}(\tau_i, x; r_i)\}_{i\in[n]} \end{array}\right] > \epsilon', \tag{4}$$

where $\mathsf{Eval}(\mathsf{ek}, x) = f_a(x)$ and $r_i \leftarrow\!\!\$\ \{0,1\}^{\ell_{rnd}}$ is the i-th fresh randomness taken in input by A, together with τ_i, at the i-th independent execution.

Next, we count the number of points x that A can successfully compute. We fix the code and the random tape (randomness r_i) of A, and run A on different x^*. In this case, let $\mathcal{X}$ be the set of inputs $x^* \in \{0,1\}^{\ell_{in}}$ for which A is able to correctly compute $y^* = f_a(x^*)$ in at least one of n independent executions. Respectively, let $\mathcal{X}_i$ (such that $\mathcal{X}_i \subseteq \mathcal{X}$) be the set of inputs $x^* \in \{0,1\}^{\ell_{in}}$ for which A can correctly compute $y^* = f_a(x^*)$ during the i-th execution with string τ_i and randomness r_i. Formally,

$$\mathcal{X}_i \stackrel{\text{def}}{=} \{x^* \in \{0,1\}^{\ell_{in}} \quad | \quad f_a(x^*) = \mathsf{A}(\tau_i, x^*; r_i)\} \text{ and } \mathcal{X} \stackrel{\text{def}}{=} \bigcup_{i\in[n]} \mathcal{X}_i.$$

By leveraging Equation (4), we can conclude that the following inequality holds $\Pr\left[x \in \mathcal{X} | x \leftarrow\!\!\$\ \{0,1\}^{\ell_{in}}\right] = \frac{|\mathcal{X}|}{2^{\ell_{in}}} > \epsilon'$ (otherwise A's advantage cannot be greater than ϵ'). This implies that the set $\mathcal{X}$ has cardinality $|\mathcal{X}| > \epsilon' \cdot 2^{\ell_{in}} = \epsilon_1 \cdot 2^{\ell_{in}} + d + 1$.

We now claim that conditioned on $E^{\mathrm{RND}}_{c',\ell_{rnd}}$, we have $|\mathcal{X}_i| \leq d$, for all $i \in [n]$. Assume the contrary, i.e., $|\mathcal{X}_i| \geq d+1$. Hence, we can build a Turing machine T by running A on $d+1$ points in $\mathcal{X}_i$ with $\tau_i \in \{0,1\}^m$ and $r_i \in \{0,1\}^{\ell_{rnd}}$, since $f_a(x^*) = \mathsf{A}(\tau_i, x^*; r_i)$ for all $x^* \in \mathcal{X}_i$ and $|\mathcal{X}_i| \geq d+1$. This would contradict Theorem 6 and accordingly $|\mathcal{X}_i| \leq d$. Since each i-th execution of A on input τ_i and r_i allows us to correctly compute at most $|\mathcal{X}_i| \leq d$, then we conclude that the only way to cover all points $x^* \in \mathcal{X}$ is to set n to be at least

$$n \geq \frac{|\mathcal{X}|}{d} > \frac{\epsilon_1 \cdot 2^{\ell_{in}}}{d} + \frac{d+1}{d} > \frac{\epsilon_1 \cdot 2^{\ell_{in}}}{d} + 1.$$

This concludes the proof of Lemma 1. □

To conclude the proof of Theorem 8, we need to bound the probability of $E^{\mathrm{RND}}_{c',\ell_{rnd}}$. Let E^{DET}_c be the following event

$$E^{\mathrm{DET}}_c \stackrel{\text{def}}{=} \left(a \text{ is } c\text{-DET-incompressibile}\right),$$

where $a = a_0 || \ldots || a_d$ are the coefficients of the polynomial $f_a(x)$ sampled by Setup. By using Theorem 5, we know that if a is c-DET-incompressible then

a is (c', ℓ_{rnd})-RND-incompressible for $c' = c + \ell_{rnd} + 2\log(\ell_{rnd}) + 1 + O(1)$ as defined in Theorem 8. Since we only consider randomized adversaries with randomness space $\{0,1\}^{\ell_{rnd}}$, we have that E_c^{DET} implies $E_{c',\ell_{rnd}}^{\text{RND}}$, and $\neg E_{c',\ell_{rnd}}^{\text{RND}}$ implies $\neg E_c^{\text{DET}}$. Thus, $\Pr\left[\neg E_c^{\text{DET}}\right] \geq \Pr\left[\neg E_{c',\ell_{rnd}}^{\text{RND}}\right]$. Hence, we conclude that, given $r_i \leftarrow\!\!\$\ \{0,1\}^{\ell_{rnd}}$ and for $i \in [n]$, the following inequality holds:

$$\Pr\left[\exists i \in [n], \mathsf{Eval}(\mathsf{ek}, x) = y_i \,\middle|\, \begin{array}{c} x \leftarrow\!\!\$\ \{0,1\}^{\ell_{in}}, \\ \{y_i \leftarrow\!\!\$\ \mathsf{A}(\tau_i, x)\}_{i\in[n]} \end{array}\right] =$$

$$\Pr\left[\exists i \in [n], \mathsf{Eval}(\mathsf{ek}, x) = y_i \,\middle|\, \begin{array}{c} x \leftarrow\!\!\$\ \{0,1\}^{\ell_{in}}, \\ \{y_i = \mathsf{A}(\tau_i, x; r_i)\}_{i\in[n]} \end{array}\right] =$$

$$\Pr\left[\exists i \in [n], \mathsf{Eval}(\mathsf{ek}, x) = y_i \,\middle|\, \begin{array}{c} E_{c',\ell_{rnd}}^{\text{RND}}, \\ x \leftarrow\!\!\$\ \{0,1\}^{\ell_{in}}, \\ \{y_i = \mathsf{A}(\tau_i, x; r_i)\}_{i\in[n]} \end{array}\right] \cdot \Pr\left[E_{c',\ell_{rnd}}^{\text{RND}}\right] \tag{5}$$

$$+ \Pr\left[\exists i \in [n], \mathsf{Eval}(\mathsf{ek}, x) = y_i \,\middle|\, \begin{array}{c} \neg E_{c',\ell_{rnd}}^{\text{RND}}, \\ x \leftarrow\!\!\$\ \{0,1\}^{\ell_{in}}, \\ \{y_i = \mathsf{A}(\tau_i, x; r_i)\}_{i\in[n]} \end{array}\right] \cdot \Pr\left[\neg E_{c',\ell_{rnd}}^{\text{RND}}\right] \tag{6}$$

$$< \epsilon' + \frac{1}{2^c} - \frac{1}{2^{(d+1)\lambda}} = \epsilon_1 + \frac{d+1}{2^{\ell_{in}}} + \frac{1}{2^c} - \frac{1}{2^{(d+1)\lambda}} = \epsilon \tag{7}$$

where in Equation (5) we used the fact that Construction 1 satisfies $(\epsilon', m, \ell_{rnd}, n)$-decomposed-min-capacity when $E_{c',\ell_{rnd}}^{\text{RND}}$ occurs (Lemma 1) and $\Pr\left[E_{c',\ell_{rnd}}^{\text{RND}}\right] \leq 1$. Conversely, in Equation (6) we used the fact that the advantage of A is at most 1 (even conditioned to $\neg E_{c',\ell_{rnd}}^{\text{RND}}$) and $\Pr\left[\neg E_{c',\ell_{rnd}}^{\text{RND}}\right] \leq \Pr\left[\neg E_c^{\text{DET}}\right] < \frac{1}{2^c} - \frac{1}{2^{(d+1)\lambda}}$ when $a \leftarrow\!\!\$\ \{0,1\}^{(d+1)\lambda}$ (Theorem 4). This concludes the proof.

A.5 Proof of Theorem 9

Case $v = 1$. Recall that, on input $x \in \{0,1\}^{\ell_{in}}$, $r \in \{0,1\}^{\ell_{rnd}}$, and $\tau \in \{0,1\}^{n_2}$, a constant-size randomized unbounded adversary $\mathsf{A} \in \mathcal{A}^{\text{1-access}}$ (with randomness space $\{0,1\}^{\ell_{rnd}}$) performs exactly one (adaptive) random access to τ and, then, read m consecutive bits (i.e., $\mathcal{I}_{\mathsf{A}(\tau,x;r)} = \{i_1, \ldots, i_m\}$ such that $i_j + 1 = i_{j+1}$ for $i \in [m-1]$). Independently from the challenge $x \in \{0,1\}^{\ell_{in}}$, the randomness $r \in \{0,1\}^{\ell_{rnd}}$, the memory $\tau \in \{0,1\}^{n_2}$, an adversary $\mathsf{A} \in \mathcal{A}^{\text{1-access}}$ is able to read at most $n_2 - m + 1$ strings $\{\tau_1, \ldots, \tau_{n_2-m+1}\}$ each of length m. This because there are at most $n_2 - m + 1$ different valid indexes of the memory $\tau \in \{0,1\}^{n_2}$ that allow A to read m (consecutive) bits. Since Π satisfies $(\epsilon, m, \ell_{rnd}, n_1)$-decomposed-min-capacity for $n_1 = n_2 - m + 1$, then a constant-size randomized unbounded adversary $\mathsf{A} \in \mathcal{A}^{\text{1-access}}$ with randomness space $\{0,1\}^{\ell_{rnd}}$ retains at most the same advantage ϵ when having access to $\tau \in \{0,1\}^{n_2}$.

Case $1 < v \leq m$. We now give an upper bound on the number of strings n_1 of length m that an adversary $\mathsf{A} \in \mathcal{A}^{v\text{-access}}$ can read from a memory of size n_2. Recall, an adversary $\mathsf{A} \in \mathcal{A}^{v\text{-access}}$ performs exactly v random accesses to

the memory. Hence, the number of valid ordered random access combinations that A can perform with a memory $\tau \in \{0,1\}^{n_2}$ are at most $P(n_2, v) = \frac{n_2!}{(n_2-v)!}$ where $P(n,k) = \frac{n!}{(n-k)!}$ is the number of permutation of k elements (without replacement) out of n distinct elements. Fix the random access composed by v different indexes $\mathcal{V} = \{i_1, \ldots, i_v\}$. Let t_j be the number of consecutive bits read by A starting from the index i_j. The number of different strings q of length m that A can read using the fixed random access $\mathcal{V}$ are at most $q \leq |\mathcal{S}|$ where

$$\mathcal{S} \stackrel{\text{def}}{=} \{(t_1, \ldots, t_v) \in \mathbb{N}^v | t_j \geq 1 \text{ for } j \in [v] \text{ and } t_1 + \ldots + t_v = m\}.$$

Observe that the cardinality of $|\mathcal{S}| = \binom{m-1}{v-1}$ can be estimated by leveraging the stars and bars combinatorial theorem where the number of stars is m and the number of bars is v. To conclude, the number n_1 of distinct strings of length m that an adversary $\mathsf{A} \in \mathcal{A}^{v\text{-access}}$ could potentially read are at most

$$n_1 \leq \frac{n_2!}{(n_2-v)!} \cdot \binom{m-1}{v-1}.$$

This concludes the proof.

A.6 Proof of Corollary 1

Proof. For $v = 1$, we easily obtain $n = m + \frac{\epsilon_1 \cdot 2^{\ell_{in}}}{d}$ by combining Theorems 8-9. We now proceed with the case $1 < v \leq m$. Let n_1 as defined in Theorem 9. By leveraging the fact that $\binom{\ell}{k} \geq (\ell/k)^k$ and $\frac{n_2!}{(n_2-v)!} = \binom{n_2}{v} v!$ we conclude that

$$n_1 = \frac{n_2!}{(n_2-v)!}\binom{m-1}{v-1} = v!\binom{n_2}{v}\binom{m-1}{v-1} \geq v!\left(\frac{n_2}{v}\right)^v \left(\frac{m-1}{v-1}\right)^{v-1}.$$

This means that Theorem 9 still holds if we consider $n_1 = v!\left(\frac{n_2}{v}\right)^v \left(\frac{m-1}{v-1}\right)^{v-1}$. The latter equality allows us to obtain the following bound

$$\sqrt[v]{\frac{n_1}{v!\left(\frac{m-1}{v-1}\right)^{v-1}}} \cdot v = n_2. \tag{8}$$

By substituing $n_1 = \frac{\epsilon \cdot 2^{\ell_{in}}}{d} + 1$ as defined in Theorem 8 and setting $n = n_2$ we obtain the bound of n as defined in the Corollary 1. □