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

(Revisiting space-based security in the adaptive setting)

Giuseppe Ateniese[1†], Long Chen[2‡], Danilo Francati[3*§], Dimitrios Papadopoulos[4], and Qiang Tang[5]

[1] George Mason University, Virginia, USA
ateniese@gmu.edu
[2] Institute of Software, Chinese Academy of Sciences, Beijing, China
chenlong@iscas.ac.cn
[3] Aarhus University, Aarhus, Denmark
dfrancati@cs.au.dk
[4] Hong Kong University of Science and Technology, Hong Kong
dipapado@cse.ust.hk
[5] 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 strict lower bound on the number of bits read from memory during evaluation (referred to as minimum capacity). No adversary, even with unbounded computational 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 $O(\lambda)$. The minimum capacity of our VCBF construction holds against adversaries that perform a constant number of random memory accesses during evaluation. This poses the natural question of whether a VCBF with high minimum capacity guarantees exists when dealing with adversaries that perform non-constant (e.g., polynomial) number of random accesses.

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

[*] An abridged version of this paper appears in the Proceedings of the 26th IACR International Conference on Practice and Theory of Public-Key Cryptography (PKC 2023).

[†] Work done in part while at Stevens Institute of Technology, Hoboken, USA.

[‡] Work done in part while at New Jersey Institute of Technology, New Jersey, USA.

[§] This work is based on the author's Ph.D. dissertation.

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 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 [49], and anti-spam mechanisms [28,9]; and password hashing or key derivations to be used against offline brute-force [55,41].

The most prominent definitions for "space-demanding" functions proposed in the literature are memory-hardness [52,6,2,19,5,3,21,4], and bandwidth-hardness [58,15]. While they share the same initial motivation, these notions vary in their formalization and achieved security guarantees. Memory-hardness, as originally defined [52], 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 $mt \in \Omega(n^2)$, where $O(n)$ is the time for an honest party. 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 distinct bits read: The former allows for a trade-off between memory block 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 an expected 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 [28,39,46], 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 [12], checking the validity of a proposed evaluation with considerably 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

	Space-unit per execution	Security analysis	Publicly verifiable
Code-hard functions [13]	Memory	Ideal cipher	✗
Memory-hard functions [52,6,4]	Time/Memory trade-off	ROM & Pebbling	✗
Bandwidth-hard functions [58]	Time/Cache-miss trade-off	ROM & Pebbling	✗
VCBF (this work)	Bits read	Standard	✓

Table 1: Comparison summary between VCBF and existing space-demanding functions. We exclude from the comparison any primitive that deviates from our objectives: (i) primitives based on heuristics or enforce memory/space usage on expectation, i.e., no strict lower bound on the memory/space usage (e.g., [1] and puzzle-based constructions [27,29]) or, (ii) primitives that require interaction (e.g., [7,30,57]). Publicly verifiable means that the correctness of the function's output can be publicly verified with significantly fewer space-units than evaluating the function. We use the term "memory" to denote the total space required to evaluate the function (this does not guarantee a lower bound on the number of bits read).

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 strict lower bound m in the necessary number of *distinct bits read* from memory in order to evaluate the function each time (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. The space notion of VCBF differs significantly from other space-related functions: It provides a strict lower bound on the number of distinct bits read at each evaluation of the function (minimum capacity) even if an adversary adaptively chooses its strategy after the function is instatiated. In addition, it does not present any time/space trade-off on evaluation, i.e., the only way to compute the VCBF's output is to satisfy its minimum capacity complexity unless the VCBF is heavily precomputed. Note that every function inevitably presents a time/space trade-off under heavy precomputation, e.g., evaluate the function on all inputs and store the outputs into an ordered dictionary. This differs from other space functions [29,1,58,6,4] in which an evaluator can tune the memory usage at the price of computing the function in more time, even if the function has not been preprocessed.

Also, unlike the notion of asymmetric hardness [13] which allows parties with access to a secret trapdoor to evaluate f quickly, we aim for public verifiability. Hence, a VCBF is a publicly verifiable function that does not present a time/space trade-off on evaluation. In that sense, it can be viewed as a space-analog of a VDF.

Comparison between VCBF and other "space-demanding" functions. 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. See Table 1 for a comparison summary between VCBF and the most prominent functions and the corresponding space flavors.

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 [59], key-derivation function PBKDF2 [41], and the recently proposed verifiable delay functions mentioned above [18,64,53]. Another related notion is proof-of-sequential-work (PoSW) [45,23,26], 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 expectation of the lower bound on the number of cache misses for any polynomial-time bounded adversary. In [1,27] a subset of a large random table (thus incompressible) is accessed during evaluation. However, they do not meet our requirement of the strict capacity lower bound on the number of bits read for each evaluation (like VDF for the time setting) since their lower bound is only a statistical expectation.

Follow-up work [29] suggests a construction with a time/space trade-off for the process of constructing the table from a representation, but this permits us to easily trade memory accesses for computation workload. We stress that [27,29] leverage a puzzle-based approach: They reach the desired number of cache misses by evaluating the function multiple times. Hence, they cannot be considered functions due to their puzzle-based nature (similarly to the analogy between VDF and PoW in the time setting). Lastly, [1] leverages an inner function f whose inverse f^{-1} cannot be evaluated in less time than accessing the memory. Hence, their construction presents a time/space trade-off: A malicious adversary may choose to involve more time to reduce the number of memory accesses.

Code-hard functions. Code-hard algorithms [13] require that a minimum amount of memory is used in order to store the code (generated using block ciphers). This has found different applications, e.g., white box encryption [11,16,17,35] or big-key encryption [10]. The key difference between a code-hard function and VCBF is that while a large amount of memory space must be dedicated for storing the code-hard function, it is possible that only a small fraction of those stored bits must be retrieved during evaluation (i.e., using memory does not imply reading bits). A VCBF imposes a non-trivial strict lower bound on bits read from memory during each evaluation.

Memory and Bandwidth-hard functions. These functions adaptively read/write from/in the memory to achieve two different objectives: Memory-hard functions require evaluators to use a large amount of memory while bandwidth-hard functions produce a high number of cache misses.[1] These functions [52,6,58,4] allow adversaries to dynamically trade additional computation for reduced memory usage on evaluation (even without precomputation); thus, they do not meet our strict lower bound guarantee.[2] Moreover, the existing formalizations are highly reliant on the random oracle model, e.g., [58] for bandwidth hardness and [52,6,4] 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 (this does not preclude the possibility of specific VCBFs operating in

[1] We stress that, in the setting of memory-hard functions, the term "memory" is used to denote the number of memory blocks required to correctly evaluate (in a given time) the function. This differs from the VCBF objective of forcing the evaluator to read a fixed number of distinct bits (requiring n memory blocks of size w on evaluation does not imply reading nw distinct bits since multiple memory blocks may present a redundant pattern that may be compressed).

[2] We stress that memory-hard functions present a time/space trade-off on evaluation that varies according to the notion of memory hardness considered (e.g., time-space complexity [52], cumulative space complexity [6], sustained space complexity [4]).

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 a 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 time/space trade-offs, as explained above.

Proof of space (PoSpace). PoSpace [7,30,57] 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 time/space trade-off, 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 [62,39,37,25,36]. 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 strict lower bound on the space necessary for evaluation while still allowing for efficient verification. Past works [58,15,6,2,19,5,3,21] achieve the first property only on expectation (i.e., expected lower bound) by relying on assumptions such as the random oracle or ideal cipher. Hence, this approach fails to achieve a strict lower bound and 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 adaptive space-based adversary to compute an output without reading enough data from memory, we turn our attention to *Kolmogorov complexity* [43], which measures the complexity (in an absolute sense) 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 [44,48,60]. (The most recent work of Liu and Pass [44] 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 an adaptive decompressing algorithm and α as a "compression" computed adaptively from 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 of a VCBF even in the presence of an adaptive adversary that chooses its strategy (that determines how memory is read and organized) after the function is instatiated (see Section 1.1 for a discussion about adaptive security in the space setting).

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., [33,51,66,31]). 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 $|(a_0, \ldots, a_d)| \geq m$ (where $|x|$ denotes the bit length of x) 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 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 5).

What remains is to find a way to sample a polynomial $f(X)$ with high Kolmogorov complexity. For any large-enough set, most of its elements have sufficiently high Kolmogorov complexity. 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 $f(X)$). 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 strict 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) that can be adaptively chosen by an adversary after some parameters are revealed (e.g., the object to compress). Still, we show that it is possible to give a positive, meaningful estimation of ϵ and n when considering adversaries that perform a constant number $v \in O(1)$ of random accesses (e.g., conditional jumps) in order to read discontinuous bits from memory. We discuss 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 (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 output that passes verification (Section 4). We stress that the minimum capacity definition of VCBF (Section 4) significantly changes the perspective about how adversaries are usually modeled in cryptography. In our setting, the power of an adversary is solely dependent on the space it uses, i.e., the adversary has unbounded computational power, but it has limitations in the space it uses.[3] In a nutshell, an adversary is only limited to the size of available (precomputed) memory and the number of bits it reads from it. Considering space-only adversaries requires rethinking the meaning of adaptive security. As we will discuss next, adaptiveness refers to the ability of choosing the precomputation strategy (that sets the memory of the adversary) and the evaluation strategy (that sets the reading strategy during evaluation) after the VCBF's public parameters (i.e., the coefficients of the polynomial) are revealed. To work with such a space adaptive setting, Kolmogorov Complexity is essential and succeeds where any other standard entropy measure fails (see Section 1.1 for a detailed discussion).
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., $d \in \Omega(\lambda^c)$ 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 2 (Section 5.2). 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., super-polynomial (Theorem 8). Our results can be extended to the asymptotic setting for the class $\mathcal{A}^{O(1)\text{-access}}$ (Corollary 2). This result implies

[3] Considering unbounded adversaries is fundamental in order to capture the (concrete) strict lower bound on the number of distinct bits read that a VCBF must guarantee (i.e., a VCBF does not present any time/bits read trade-off). We provide a more detailed discussion in Section 4 and Remark 3.

the security of our construction against adversarial strategies primarily used in practice (e.g., precomputed dictionaries) or strategies executed on limited devices that have a bound on the number of random accesses (e.g., for energy efficiency) that they can perform. In Section 4 and Section 5.1, we discuss our results in more detail.

Regarding the larger class of adversaries $\mathcal{A}^{\omega(1)\text{-access}}$, the minimum capacity of our polynomial-based VCBF construction deteriorates when n gets closer to $d^{1+\delta}\lambda^{1+o(1)}$ for a constant $\delta > 0$. This is due to the work of Kedlaya and Umans [42]: They shows how to build a data structure D of size at most $d^{1+\delta}\lambda^{1+o(1)}$ (only from the coefficients of $f(X) \in \mathbb{F}_p[x]$) that allows them to evaluate $f(X)$ over any of the points. This evaluation requires reading a non-constant number of elements from D (using $\omega(1)$ random accesses) whose total size is at most $O(\log(d)^{s_1}\lambda^{s_2})$ for some positive $s_1, s_2 \in O(1)$. Hence, the plain evaluation of a polynomial of degree d can not guarantee a minimum capacity of $m \in \omega(\log(d)^{s_1}\lambda^{s_2})$ when an adversary $\mathsf{A} \in \mathcal{A}^{\omega(1)\text{-access}}$ has access to a memory of size n, close to or greater than $d^{1+\delta}\lambda^{1+o(1)}$ (see Section 5.1 for more details).

1.1 Adaptive security and Kolmogorov complexity (vs. selective security and other entropy measures)

Here, we provide an answer to two natural questions about the meaning of adaptive security (in the space setting) and Kolmogorov complexity. These points significantly differentiate the techniques used in this work from previous ones.

Adaptive security in the space setting. In the standard computational time cryptographic setting, adaptive security refers to the ability of an adversary of changing its behavior according to the scheme's parameters with the objective of increasing its advantage in breaking the scheme's security. An example is the adaptive CCA security of public encryption in which an adversary wants to increase its advantage in distinguishing between two encryptions by adaptively choosing both the two challenge messages and the next query for the decryption oracle after seeing the public key and the answers received from previous decryption queries. The natural question we pose is "What does adaptive security mean for the minimum capacity definition of VCBF?". To give a concrete answer to this question, it is necessary to rethink the meaning of adaptive security against adversaries whose power is measured by solely considering the memory used/read (as done in this work). Jumping ahead, the minimum capacity of VCBF (Section 4) guarantees that an adversary needs to read at least m bits from its memory τ (of size n) when asked to correctly evaluate the function on a random point. This must hold even if the adversary is computationally unbounded, and it is allowed to generate/organize its memory τ by precomputing the VCBF according to its parameters (i.e., polynomial). In such a setting, the objective of an adversary is to break the security of VCBF by minimizing the number of distinct bits m read from the precomputed memory. To achieve this, an adversary may think of changing its compression/precomputation strategy after the VCBF's parameters (i.e., the polynomial coefficients) are revealed.[4] This is analogous to the CCA public key encryption example in which an adversary changes its two challenge messages after seeing the public key and the answers of the decryption oracle. To formally define the intuitive security of VCBF (i.e., a strict lower bound on the number m of distinct bits read), it is fundamental to cover adaptive space-based adversaries (as described above). Indeed, if an adversary can change its precomputation/compression strategy after seeing the VCBF's parameters and reduce, for example, m by $\log(\lambda)$ bits then the strict lower bound is not strict anymore. For this reason, the natural definition of minimum capacity (Definition 8) requires that the function remains secure for any

[4] For example, a particular (hard to guess) compressible pattern may be revealed after the polynomial coefficients are chosen. Note that this may happen (with a certain probability) even if the polynomial is sampled at random.

possible space-based adversary sampled after the instantiation of VCBF, i.e., the precomputation and evaluation strategy (i.e., the memory and the bits read) of the adversary can depend on the VCBF itself. Such a model of adaptive security requires the usage of Kolmogorov complexity (see next).

Why Kolmogorov complexity? Conventional entropy measures (including Yao entropy that leverages the notion of compression and Shannon) consider the incompressibility of objects only on expectation. It implicitly means that the compression strategy does not depend on the object (i.e., our polynomial of our VCBF construction) sampled from a distribution. The typical example is on [38, Page 10] (quoting): “Consider the ensemble consisting of all binary strings of length 9999999999999999. By Shannon’s measure, we require 9999999999999999 bits on the average to encode a string in such an ensemble. However, the string consisting of 9999999999999999 1’s can be encoded in about 55 bits by expressing 9999999999999999 in binary and adding the repeated pattern 1”. Note that the above argument applies also to the Rényi family of entropies (e.g., min-entropy).[5] The Kolmogorov complexity overcomes these limitations by considering the worst-case scenario: It measures incompressibility in an absolute sense, i.e., the compression strategy can depend on the object. Hence, lower-bounds derived through Kolmogorov complexity are universal, and they do not hold only on expectation. Also, quoting [63]: “The Kolmogorov complexity of an object is a form of absolute information of the individual object. This is not possible to do by C.E. Shannon’s information theory. Unlike Kolmogorov complexity, information theory is only concerned with the average information of a random source”.

In the VCBF setting, these concepts translate into adaptive vs. selective security. Kolmogorov complexity allows us to bound the minimum capacity of VCBF in the adaptive setting in which the adversarial compression/precomputation strategy can depend on the VCBF’s parameters (this mimic the adversarial behavior of changing strategy after the parameters are revealed). As already discussed, this is fundamental in order to have strict (universal) lower bound on the number of bits m that an adversary needs to read to evaluate a VCBF. Adaptive security remains unachievable if we consider standard entropy measures: This is because Information Theory studies the average information in objects, i.e., compression/precomputation strategies are fixed before the object (i.e., polynomial) is revealed/sampled. Hence, Kolmogorov complexity remains a fundamental tool in order to deal with space-based adaptive security and, in turn, to prove the security of our polynomial-based VCBF.

1.2 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 various settings where memory usage needs to be enforced. In this direction, we describe how VCBF can be used as an *energy-consumption function* to achieve fairness among ASIC and CPU participants. We then briefly discuss other promising VCBF applications. We emphasize that the objective of this work is to lay the foundation for VCBFs, providing an initial study about publicly verifiable asymmetric memory/space hardness in the standard model. Naturally, depending on the application, ad-hoc properties and/or slightly different flavors of VCBF may be required, opening interesting directions for subsequent works (see Section 6).

[5] This can also be seen by observing that the Rényi family of entropies is equivalent to Shannon entropy when considering uniform distributions (as considered in this work, e.g., polynomial’s coefficients are sampled at random).

Energy-consumption function. Juels and Brainard [40] proposed client-puzzles as a solution to mitigate denial of service attacks (the concept of cryptographic puzzles can be traced back to Merkle's key exchange [46] and Dwork and Naor's pricing function [28]). 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 [40,8,20,22,61] will force adversaries to consume certain CPU cycles as the cost for attacks. However, state-of-the-art hash engines [14,58] 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 to 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 [58], 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 function with a strict 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 VCBF can be used as an energy-consumption function in the following protocol between a server S and a client C:

- 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 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 7 (in Section 5), we can easily find a set of parameters so that an adversary needs to invest a sufficiently large amount of energy in computing the function. Observe that the client is required to compute the VCBF f on a (honest) challenge x chosen by the server. Another option is to allow the client to choose multiple challenges on its own as is usually done in client-puzzles. In this case, it is fundamental that f can not be amortized, i.e., the puzzle's total energy-cost increases proportionally to the number of parallel evaluations (on different challenges) of f. We stress that this work does not study amortization but this is not an inherent limitation of VCBF as a primitive. Non-amortizable VCBFs can be studied in future works (Section 6).

The above protocol can be extended to blockchain systems that support smart contracts. For example, a client C may be required to evaluate the VCBF f on input $x = \mathsf{H}(s,t)$ in order to trigger the execution of a smart contract S. Here, inside the hash function H, we place s which is the current state of the smart contract and t a counter used to randomize the challenge $x = \mathsf{H}(s,t)$ (e.g., t is incremented after each invocation of the contract or after a block is mined).[6] In this way, an adversary is desisted from monopolizing the service offered by the smart contract S for specific malicious purposes. For instance, if the smart contract runs a decentralized auction system, the adversary will not be able to produce spamming bids to delay the acceptance of valid bids

[6] The challenge $x = \mathsf{H}(s,t)$ has this format since smart contracts cannot generate secret randomness to sample a random challenge.

from competitors. We stress that efficient public verification is essential in blockchain systems since verifiers, that check the correctness of executions, have limited resources.

Real-time Services (and VCBF vs. VDF). Although VCBF and VDF are both efficiently verifiable, there are applications in which VDFs can not be used, whereas VCBFs can. Consider a server S that offers a real-time service in which it is a requirement to receive requests within a precise time frame (e.g., within 1 minute). Clearly, using a VDF to block denial of service attacks is not an option since the time required to evaluate the VDF will delay all users' requests and affect the quality of the real-time service. A concrete example is an auction service: Bids must be received before the end of the auction or within a given time frame. Hence, VCBFs offer a unique solution in scenarios in which creating a delay is not acceptable.

The Filecoin network. Protocol Labs is working on Filecoin [54], a blockchain-based decentralized storage system that has gathered much visibility in the last few years (it raised over $250 million through an ICO in 2017). In Filecoin, miners earn coins by offering their storage to clients interested in storing and replicating files. The mining power in Filecoin is proportional to the active storage offered by a miner. Thanks to its public and capacity efficient verification, a VCBF can play an important role in improving *proof of useful space* (a fundamental primitive in the Filecoin protocol), i.e., a primitive that allows miners to prove that they are using a significant amount of space to store (multiple) files. In particular, Filecoin is interested in designing a proof of useful space in the cost model [47]. However, a common problem of proof of useful space constructions (e.g., [34]) is the possibility of trading space for computation: An evaluator may erase some data and reconstruct it on the fly when needed. A VCBF can tremendously improve proof of useful space by enforcing rationality during the computation when working in the cost model (e.g., by replacing the RO with a VCBF in graph-labeling based constructions). For example, the minimum capacity of VCBFs may increase the costs (e.g., energy consumption) of regeneration of the erased data. This encourages evaluators to store the data in its entirety. Also, the VCBF public verification does not introduce any additional cost to verifiers with minimal resources in terms of space and energy.

Other possible applications. 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., [56,67]), 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 $[n] \overset{\text{def}}{=} \{1, \ldots, n\}$. 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 $x \leftarrow_{\$} \mathcal{X}$.

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. A self-delimiting code permits 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 (constant size) Turing machines to read any input composed by multiple concatenated values, each of arbitrary length. We consider 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 (constant size) 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.[7]

$\mathsf{Verify}(\mathsf{vk}_x, y, \pi_y)$**:** Upon input 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.

[7] 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 $\sigma_y = (y, \pi_y)$.

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:*

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

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 $x^* \in \mathcal{X}$ (*soundness*).

Definition 2 (Soundness of VC [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 $(d/2)$-Strong Diffie-Hellman assumption ($(d/2)$-SDH).

Time Efficient Verification. VC schemes allow verifiers to check the computation's correctness more efficiently than the work required to evaluate the polynomial honestly; 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., $o(t)$).[8]

By leveraging the $(d/2)$-SDH assumption, 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. Furthermore, it presents succint verification keys vk_f of size $O(\lambda)$ that does not depend on the degree d of the polynomial $f(X) \in \mathbb{Z}_p[x]$. This will allow our VCBF to have an efficient verification in terms of capacity (see Section 5.1).

2.3 Kolmogorov Complexity

The Kolmogorov complexity [43] 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 $\mathsf{T}(\alpha) = x$.*

[8] 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.,

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

where $\langle a, b \rangle$ denotes the self-delimiting coding of strings a and b.[9] 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 [43, 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 [43, Section 3] for a more detailed discussion about the prefix-free version.

Kolmogorov complexity independent of the Turing machine. The definition of plain Kolmogorov complexity can be made 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.[10] 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 [43, 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 $x, y \in \{0,1\}^*$ we have $C_{\mathsf{U}}(x) \le C_{\mathsf{T}}(x) + c_{\mathsf{T}}$ and $C_{\mathsf{U}}(x|y) \le C_{\mathsf{T}}(x|y) + c_{\mathsf{T}}$.*[11]

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 $C(x) \overset{def}{=} C_{\mathsf{U}}(x)$ and $C(x|y) \overset{def}{=} C_{\mathsf{U}}(x|y)$ 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 [43, Section 2.1.4], by removing the size constraint of T, it is possible to assign low complexity to any string by simply

[9] As we will discuss later, Kolmogorov Complexity considers constant-size Turing machines. This requires the use of a self-delimiting code to encode multiple inputs.

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

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

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., $\mathsf{T}'(\bot) = x$. 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) [65,50], with a polynomial slowdown of $O(t^6)$.

String incompressibility. A crucial notion derived from the Kolmogorov complexity is the incompressibility of a string [43, Definition 2.2.1] with respect to unbounded deterministic Turing machines.

Definition 5 (Deterministic c-incompressibility [43, 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 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 2 ([43, 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 2, we can easily calculate the probability of sampling a c-DET-incompressible string from $\mathcal{X}$. The proof is included in Appendix A.1.

Theorem 3. *Let $\mathcal{X}$ be a finite set of cardinality m, then the following probability holds:*

$$\Pr[x \textit{ is c-DET-incompressible} \mid x \leftarrow_{\$} \mathcal{X}] \geq 1 - 2^{-c} + 1/m.$$

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 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 $\alpha \in \{0,1\}^{|x|-c-1}$, 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 $r \in \{0,1\}^{\ell_{rnd}}$. 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 4 reports the formal result, whose proof appears in Appendix A.2.

Theorem 4. *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 $T \overset{\text{def}}{=} (f(x_0), \ldots, f(x_n))$ 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 (of appropriate size) 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 lower 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. 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 5 (Kolmogorov-bound for (adaptive) Polynomial Evaluation). *For any $\lambda \in \mathbb{N}$, let $a \in \{0,1\}^{(d+1)\lambda}$ be a binary string and p a prime 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 $x_i \in \{0,1\}^{\ell_{in}}$, we have:*

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

where $m = (d+1)(\lambda - \ell_{in} - 2\log(\ell_{in}) - 1) - c' - \lambda - 2\log((d+1)\lambda - c') - 2\log(\lambda + 1) - 4$.

We stress that Kolmogorov complexity permits us to prove Theorem 5 under the universal quantification of any $d+1$ evaluation points and any adversarial strategy (i.e., any memory α and evaluation strategy T) selected after the polynomial. This is essential in the adaptive space-based setting (Section 1.1) in which we want to estimate the size of information generated/read w.r.t. an arbitrary precomputation of the polynomial. Indeed, the precomputation adopted by an adversary may depend on both the polynomial and an arbitrary distribution of the evaluation points (e.g., dictionary attack). This aspect is fundamental to prove the *adaptive* security of our VCBF (see Section 4 and Section 5.1).

Remark 1. An alternative way to interpret Theorem 5 is that any possible description (T, α) of $f(X)$ is bigger than the parameter m (defined in Theorem 5). Indeed, if there exists a string α that encodes $f(X)$ such that $|\alpha| < m$, then we contradict Theorem 5 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 5 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 $f(X)$.[12] For this reason, to ensure that $m \geq 0$, we must set $\ell_{in} \leq \frac{(d+1)\lambda - c' - 2\log((d+1)\lambda - c') - 2\log(\lambda+1) - 4}{d+1} - 2\log(\ell_{in}) - 1$ (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 $(x_0, \ldots, x_d)$.

[12] 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 $i \in \{0, \ldots, d\}$.

4 Definition of Verifiable Capacity-bound Functions

A VCBF forces an evaluator to read at least m distinct bits from its main memory. Moreover, a VCBF does not permit to trade time for capacity, i.e., an evaluator is forced to read m distinct bits independently from its computational capabilities. As explained in [57], the number of off-chip memory accesses impacts 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 $\mathit{\Pi}$ 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 *capacity parameter* 1^k regulates the actual capacity cost, i.e., the number of bits read by the evaluator), 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. In the paper, we use the notation $y = \mathsf{Eval}(\mathsf{ek}, x)$ (or simply $\mathsf{Eval}(\mathsf{ek}, x)$) to denote solely the output y.

$\mathsf{Verify}(\mathsf{vk}, x, y, \pi)$**:** 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.

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 $\mathit{\Pi}$ 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:*

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

Adaptive 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 ordered 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.[13] A VCBF scheme is secure

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

in the *adaptive setting* if for any unbounded adversary sampled after VCBF's instiantiation (i.e., execution of Setup) it is infeasible to 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.[14]

Definition 8 ((Adaptive) 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:*

$$\Pr\Big[\mathsf{Eval}(\mathsf{ek}, x) = y \wedge |\mathcal{I}_{\mathsf{A}(\tau,x;r)}| = m \Big| x \leftarrow\$ \{0,1\}^{\ell_{in}}, y = \mathsf{A}(\tau, x; r)\Big] \leq \epsilon, \tag{1}$$

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 (exponential time) 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). This guarantees the impossibility of trading time for capacity.

Note that the evaluator must return the correct output $y = \mathsf{Eval}(\mathsf{ek}, x)$ and not a verifying proof π. The infeasibility of computing a verifying proof for a false output is defined by the soundness property (see next Definition 10). The choice of defining these two properties independently allows us to define them with respect to different settings, i.e., unbounded vs. computational adversaries. As mentioned above, defining adaptive minimum capacity in the unbounded setting is necessary to properly capture the absence of time/bits read trade-offs. See Remark 3 for more details.

Moreover, the definition captures the adaptive space-based setting described in Section 1.1. This is because the quantifiers of the security definition states that the VCBF remains secure for any memory τ and adversary A both selected after the VCBF's instantiation (i.e., Setup algorithm). Intuitively, each τ (resp. A) represents an arbitrary precomputed memory (resp. arbitrary evaluation/reading strategy) that can depend on ek and vk (e.g., polynomial's coefficients).

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 which contains (possibly partial) precomputed values, e.g., some evaluations $y = \mathsf{Eval}(\mathsf{ek}, x_i)$ of a subset of inputs $\mathcal{X} \subset \{0,1\}^{\ell_{in}}$ (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 access to A and working in the standard setting (i.e., no oracles, no idealized functionalities, no ROM).

[14] 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 output $y = \mathsf{Eval}(\mathsf{ek}, x)$.

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 $j \in [m-1]$.[15] 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 8). For completeness, we also include the results regarding the class $\mathcal{A}^{v\text{-access}}$ for $1 \le v \le m$, 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 $(\mathsf{negl}(\lambda), O((d+1)\lambda), o((d+1)\lambda), \omega(\lambda^s))$-min-capacity (in the asymptotic setting) with respect to the class of adversaries $\mathcal{A}^{O(1)\text{-access}}$ for every positive constant s. Regarding $\mathcal{A}^{\omega(1)\text{-access}}$, the minimum capacity of our construction remains unclear. What we know is that the evaluation of a polynomial can not satisfy minimum capacity for $\epsilon \in \mathsf{negl}(\lambda)$ and $m \in \omega(\log(d)^{s_1}\lambda^{s_2})$ (for some positive $s_1, s_2 \in O(1)$) when n is close to or greater than $d^{1+\delta}\lambda^{1+o(1)}$ (for a constant $\delta > 0$) because of the efficient data structure for polynomial evaluation of Kedlaya and Umans [42] (see Section 5.1). 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 (adaptive) 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 $\mathcal{A}$.*

Remark 2. The Definitions 8-9 give robust guarantees in terms of capacity (according to the corresponding class of adversaries). For example, they consider unbounded adversaries and the minimum capacity must hold for every possible adversary A and memory τ after the instantiation of the scheme (execution of the setup algorithm). This corresponds to the adaptive space-based security setting described in Section 1.1. Also, 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 [58,15,6,2,19,5,3,21,4] 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. Therefore, the gap between RO queries and the actual number of bits read by the evaluator is artificially ignored in previous models. Finally, 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: A fundamental aspect that needs to be considered when proving specific concrete memory bounds (in the standard model) for VCBFs.

[15] 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.

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.

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:*

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

Remark 3 (On the combination of minimum capacity and soundness). Formalizing adaptive minimum capacity (Definitions 8 and 9) and soundness (Definition 10) separately allows us to define these notions with respect to two distinct settings, i.e., unbounded adversaries vs. computational bounded adversaries. In turn, minimum capacity with respect to unbounded adversaries is fundamental to capturing the (concrete) strict lower bound on the number of distinct bits guaranteed by a VCBF. This is because the unbounded setting guarantees that the lower bound must be satisfied independently of the running time of the adversary, i.e., trading time for bits read is infeasible. Observe that the computational bounded version of minimum capacity does not guarantee the absence of a time/bits read trade-off. For example, a VCBF presenting an exponential trade-off (e.g., a PPT adversary can choose not to read a few bits at the cost of doubling its running time) may satisfy, asymptotically speaking, computational minimum capacity. However, the concrete lower bound would not be strict since the adversary can play with the gap allowed by the trade-off (this is not allowed when considering minimum capacity w.r.t unbounded adversaries as in our results). This is problematic when the VCBF is instantiated in practice since it makes the capacity bound less clear. For this reason, we chose to formalize these notions separately instead of being combined into a single one with respect to computational bounded adversaries. Naturally, since we consider computational soundness, the final security of the VCBF holds only against computationally bounded adversaries (unless we drop the VCBF's efficient verification). Still, we emphasize once again that unbounded minimum capacity is fundamental since it guarantees the absence of a trade-off with which the (computationally bounded) adversary could play with. Lastly, it may seem that another natural approach is to combine minimum capacity and soundness into a single definition that considers unbounded adversaries. Unfortunately, this is not possible since a VCBF that has a capacity efficient verification (see next Definition 11) cannot satisfy, at the same time, both minimum capacity and soundness with respect to unbounded adversaries (soundness with respect to unbounded adversaries is also known as perfect soundness, i.e., it does not exist a valid proof for a false statement/output). This is because an exponential adversary always exists that brute-forces all pairs of proofs and outputs until it finds the one that verifies. By leveraging perfect soundness, the adversary is guaranteed that the corresponding output is the correct VCBF's evaluation. This attack only requires reading the VCBF's verification key vk, whose size must be sublinear in the VCBF's minimum capacity m. This is required to satisfy capacity efficient verification (see next Definition 11).

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 $\mathit{\Pi}$ to be efficiently verifiable:

Definition 11 (Capacity Efficient Verification of VCBF). *If $\mathit{\Pi}$ 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).*[16]

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 (see Section 5.1).[17]

On Energy Consumption. A motivation for VCBFs is ASIC resistance. State-of-the-art hash engines [14,58] 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 [58]. 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.

5 VCBF from VC for Polynomial Evaluation

In this section we show how to build a VCBF from VC for polynomial evaluation.

Construction 1 *Let $\mathcal{F}_{\lambda,d,p} = \{f_a(X) = \sum_{i=0}^{d} a_i \cdot x^i \mod p\}_{a \in \{0,1\}^{(d+1)\lambda}}$ be an ensemble of polynomials where $a = a_0 || \dots || a_d$, $\lambda \in \mathbb{N}$, $d \in \mathbb{N}$, and p is a prime of $\lambda + 1$ bits. 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 class $\mathcal{F}_{\lambda,d,p}$. We build a VCBF scheme with input space $\{0,1\}^{\ell_{in}}$ in the following way:*

$\mathsf{Setup}(1^\lambda, 1^k)$**:** *Without loss of generality, we assume $k = (d+1)\lambda$. On input the security parameter 1^λ and the capacity parameter 1^k, the setup algorithm samples $a_0 || \dots || a_d = a \leftarrow\$ \{0,1\}^{(d+1)\lambda}$ where $|a_i| = \lambda$ for $i \in \{0, \dots, d\}$. 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 $f_a \in \mathcal{F}_{\lambda,d,p}$.*

$\mathsf{Eval}(\mathsf{ek}, x)$**:** *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, honest evaluators need 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. Correctness follows directly from the correctness of the underlying schemes. For security and verification complexity, we establish the following results.

[16] Observe that $|\mathsf{vk}| + |x| + |y| + |\pi| \in o(m)$ (i.e., vk, π, y, x are "succinct") is necessary to obtain a capacity-efficient verification of $o(m)$. This is because vk, π, y, x are part of the verification algorithm Verify of VCBF.

[17] 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 $(a_0, \dots, a_d) \in \mathbb{F}_p^{d+1}$ of the polynomial $f(X) \in \mathbb{F}_p[x]$.

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 6 (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 parameters $(\epsilon, m, \ell_{rnd}, n)$. The second part is only required to give an upper bound to the memory size parameter n.

(Part one). Decomposed (adaptive) minimum capacity. The definition of decomposed minimum capacity is identical to Definition 8 except that the memory τ is decomposed into n distinct 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). 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 (adaptive) minimum capacity of VCBF). *Fix the keys* $(\mathsf{ek}, \mathsf{vk})$ *output by* $\mathsf{Setup}(1^\lambda, 1^k)$*. A VCBF scheme* Π *with input space* $\{0,1\}^{\ell_{in}}$ *satisfies* $(\epsilon, m, \ell_{rnd}, n)$*-decomp-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:*

$$\Pr\Big[\exists i \in [n], \mathsf{Eval}(\mathsf{ek}, x) = y_i \Big| x \leftarrow_\$ \{0,1\}^{\ell_{in}}, \{y_i \leftarrow_\$ \mathsf{A}(\tau_i, x)\}_{i \in [n]}\Big] \leq \epsilon.$$

Regarding decomposed minimum capacity (Definition 12) of Construction 1, 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 5 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 (for c' as defined in Theorem 4). 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 $\{0,1\}^{\ell_{in}}$. Lastly, we can leverage Theorems 3-4 to claim that the string $a = a_0 || \ldots || a_d$ (i.e., the coefficients of the polynomial $f_a(x)$ described by ek) is not (c', ℓ_{rnd})-RND-incompressible with probability at most $\frac{1}{2^c} - \frac{1}{2^{(d+1)\lambda}}$. We report the formal result whose proof appears in Appendix A.4.

Theorem 7 (Decomposed (adaptive) minimum capacity). *Fix the keys* $(\mathsf{ek}, \mathsf{vk})$ *output by* $\mathsf{Setup}(1^\lambda, 1^k)$*. The VCBF scheme* Π *of Construction 1 with input space* $\{0,1\}^{\ell_{in}}$ *satisfies* $(\epsilon, m, \ell_{rnd}, n)$*-decomp-min-capacity (Definition 12) with respect to keys* $(\mathsf{ek}, \mathsf{vk})$ *where* $\lambda \in \mathbb{N}, d \in \mathbb{N}, c \in \mathbb{N}, \epsilon_1 \in$

$[0, 1]$,

$$\begin{aligned} m &= (d+1)(\lambda - \ell_{in} - 2\log(\ell_{in}) - 1) - c' \\ &\qquad - \lambda - 2\log((d+1)\lambda - c') - 2\log(\lambda+1) - 4, \\ c' &= c + \ell_{rnd} + 2\log(\ell_{rnd}) + 1 + O(1), \\ \epsilon &= \epsilon_1 + \frac{d+1}{2^{\ell_{in}}} + \frac{1}{2^c} - \frac{1}{2^{(d+1)\lambda}}, \quad n = \frac{\epsilon_1 \cdot 2^{\ell_{in}}}{d} + 1. \end{aligned}$$

(Part two). From Definition 12 to Definition 9. We now show that any VCBF that satisfies $(\epsilon, m, \ell_{rnd}, n-m+1)$-decomp-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 $(\tau_1, \dots, \tau_{n-m+1})$ each of length m (as defined in Theorem 7). 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. Lastly, we want to stress that the counting argument is not required if we consider memories of size $n = m$. This because a precomputed memory $\tau \in \{0,1\}^m$ only allows an adversary to answer to at most d points (Theorem 5). This directly implies that our Construction 1 satisfies $(\epsilon, m, \ell_{rnd}, m)$-min-capacity with respect to the *optimal* Definition 8 (i.e., security against adversaries that arbitrarily access its memory) where $\epsilon = \frac{d+1}{2^{\ell_{in}}} + \frac{1}{2^c} - \frac{1}{2^{(d+1)\lambda}}$. We now report the formal result for the class of adversaries $\mathcal{A}^{v\text{-access}}$ whose proof is included in Appendix A.5.

Theorem 8 ($\mathcal{A}^{v\text{-access}}$-class (adaptive) 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 $(\epsilon, m, \ell_{rnd}, n_1)$-decomp-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*

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

The following corollaries report the concrete and asymptotic min-capacity of Construction 1 whose proofs appears in Appendix A.6 and Appendix A.7). The concrete Corollary 1 shows that we can obtain a reasonable concrete trade-off between ϵ and n only for the class of adversaries $\mathcal{A}^{\text{1-access}}$ (due to the limited power of our counting argument). Still, the result applies to adversarial strategies (primarily used in practice) such as precomputed dictionary attacks (e.g., ordered dictionary in which the x-th evaluation $f(x)$ is stored at the x-th offset) or limited devices that are hindered from performing non-constant random accesses (e.g., for energy efficiency). In Section 5.2 we provide some examples of concrete instantiations. On the other hand, the asymptotic Corollary 2 shows that a secure VCBF exists (in the standard model) 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 $\lambda \in \mathbb{N}, d \in \mathbb{N}, c \in \mathbb{N}, \epsilon_1 \in [0,1]$,

$$\begin{aligned}
m &= (d+1)(\lambda - \ell_{in} - 2\log(\ell_{in}) - 1) - c' \\
&\qquad - \lambda - 2\log((d+1)\lambda - c') - 2\log(\lambda+1) - 4, \\
c' &= c + \ell_{rnd} + 2\log(\ell_{rnd}) + 1 + O(1), \\
\epsilon &= \epsilon_1 + \frac{d+1}{2^{\ell_{in}}} + \frac{1}{2^c} - \frac{1}{2^{(d+1)\lambda}}, \\
n &= \begin{cases} m + \frac{\epsilon_1 \cdot 2^{\ell_{in}}}{d} & \textit{if } v = 1 \\ \sqrt[v]{\left(\frac{\epsilon_1 \cdot 2^{\ell_{in}}}{d} + 1\right) / \left(v! \left(\frac{m-1}{v-1}\right)^{v-1}\right)} \cdot v & \textit{if } 1 < v \le m. \end{cases}
\end{aligned}$$

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* $(\mathsf{negl}(\lambda), O((d+1)\lambda), o((d+1)\lambda), \omega(\lambda^s))$*-min-capacity with respect to the class of adversaries* $\mathcal{A}^{O(1)\text{-}access}$ *for every constant* $s \ge 1$.

We stress that the memory size n does not need to be super-polynomial (in the security parameter) in order to consider a VCBF secure. Indeed, in a scenario in which a machine has at most $n = \lambda^s \in \mathsf{poly}(\lambda)$ bits of free memory (for a positive constant s), it is enough to show that the VCBF satisfies $(\epsilon, m, \ell_{rnd}, \lambda^s)$-min-capacity where ϵ is the target advantage.

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 capacity complexity of Verify 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 with respect to the VC scheme of Elkhiyaoui et al. [31] that uses an asymmetric bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ in which the $(d/2)$-SDH assumption holds. The execution of $\mathsf{ProbGen}_{\mathsf{VC}}(\mathsf{vk}_f, x)$ computes and returns $\mathsf{vk}_x = (\mathsf{vk}_x^0, \mathsf{vk}_x^1) = (g_{b_0} \cdot g^{x^2}, h_{r_1}^x \cdot h_{r_0})$ and $\sigma_x = x$, where $\mathsf{vk}_f = (g_{b_0}, h_{r_1}, h_{r_0}) \in \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_2$, $x \in \mathbb{F}_p$, and g the generator of $\mathbb{G}_1$ (observe that the size of the verification key vk_f is $O(\lambda)$, i.e., does not depend on the degree d of the polynomial). Moreover, $\mathsf{Verify}_{\mathsf{VC}}(\mathsf{vk}_x, y, \pi_y)$ verifies the correctness of the computation by checking the equality $e(g, h^y) \overset{?}{=} e(\mathsf{vk}_x^0, \pi_y) \cdot e(g, \mathsf{vk}_x^1)$, 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 is because the executions of $\mathsf{ProbGen}_{\mathsf{VC}}$, $\mathsf{Verify}_{\mathsf{VC}}$, and the sizes of $(\mathsf{vk}_f, x, y, \pi_y)$ (that compose the inputs of $\mathsf{ProbGen}_{\mathsf{VC}}$ and $\mathsf{Verify}_{\mathsf{VC}}$), are independent of the polynomial degree d in terms of both capacity and time.

Improve the memory size bound. For $v \in \omega(1)$, our VCBF construction needs to face the efficient data structure for polynomial evaluation of Kedlaya and Umans [42]. In particular, they show that, for any constant $\delta > 0$, there exists a data structure D of size $d^{1+\delta}\lambda^{1+o(1)}$ that can be computed by preprocessing only the coefficients of $f(X) \in \mathbb{F}_p[X]$. An evaluator in $\mathcal{A}^{\omega(1)\text{-access}}$ can correctly evaluate $f(x)$ on every $x \in \{0,1\}^{\ell_{in}}$ in time $\mathsf{polylog}(d) \cdot \lambda^{1+o(1)}$, performing a non-constant number of random accesses and reading at most $\mathsf{polylog}(d)\lambda^{1+o(1)} \cdot w$ bits from D (with $w \in O(\lambda)$ we denote bit size of the elements contained in D). Hence, our VCBF construction can not achieve $(\epsilon, m, \ell_{rnd}, n)$ for $\epsilon \in \mathsf{negl}(\lambda)$ and $m \in \omega(\log(d)^{s_1}\lambda^{s_2})$ (for some positive $s_1, s_2 \in O(1)$) when n is close to or greater than $d^{1+\delta}\lambda^{1+o(1)}$. The above observation poses the natural question of whether an asymptotic VCBF (in the $\mathcal{A}^{\omega(1)\text{-access}}$ setting) that satisfies min-capacity for reasonably large m and

Table 2: 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	
		$\epsilon_1 = 2^{-90}$	$\epsilon_1 = 2^{-50}$
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 ($d \approx 78.20 \cdot 10^7 \approx \lambda^{2.95}$)	$\epsilon \approx 2^{-90}$, $n \approx 80.20\text{GB}$, $m \approx 80.20\text{GB}$	$\epsilon \approx 2^{-50}$, $n \approx 44.02\text{TB}$, $m \approx 80.20\text{GB}$

n super-polynomial in λ as in Corollary 2 (i.e., n asymptotically larger than the size $d^{1+\delta} \cdot \lambda^{1+o(1)}$ of the data structure of Kedlaya and Umans [42]). The answer to the important question requires a non-trivial and precise study that can be undertake in future works.

5.2 Concrete Instantiation

Table 2 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 $\mathcal{A}^{\text{1-access}}$. 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. Note 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 $\mathsf{A} \in \mathcal{A}^{\text{1-access}}$. Recall that in $\mathcal{A}^{\text{1-access}}$ we can identify attack strategies, primarily used in practice, that use a single adaptive random access to the memory such as pre-computed dictionaries (e.g., on input x, perform a random access to fetch the evaluation $f(x)$ stored at the x-th position). Hence, Corollary 1 and the estimations of the parameters reported in this section apply to these strategies.

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 2). 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 5 (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 $\approx 2^{-50}$. As discussed in Section 4, this trade-off is inevitable. 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 2 and Corollary 1): An adversary needs a small memory in order to retain very small advantages. This also relates to Theorem 5: 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 any adversary to retain a very small advantage $\epsilon \approx d/2^{\ell_{in}}$. Concretely, for $k = 100$GB and $d = 15.65 \cdot 10^8$ $(\lambda + 1 = 512)$ and $\ell_{in} = 128$ (as in Table 2), a memory of size $n = m$ may permit us 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:

1. As extensively discussed in Section 5.1, a natural and important next step is to study if a VCBF exists in stronger security models (e.g., $\mathcal{A}^{\omega(1)\text{-access}}$ setting), possibly obtaining a super-polynomial bound on n (at least in the computational-bounded setting).
2. 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 with new exciting properties.
3. The definition of minimum capacity (Definition 8) and our polynomial-based VCBF (Construction 1) do not take into account *amortization*: An evaluator that computes in parallel the VCBF on ℓ random points $(x_1, \ldots, x_\ell) \leftarrow\!\!\$\ \mathcal{X}^\ell$ (known in advance) will still incur a minimum capacity of m (and not $\ell \cdot m$). Hence, the minimum capacity does not scale linearly with the number ℓ of points computed in parallel. Studying amortization in VCBFs (as done by [6,4] for memory-hard functions) is an interesting future line of research.
4. Protocol Labs is investigating the use of VCBFs, improving proof of useful space in Filecoin [54]. However, several aspects of VCBF needs to be studied before achieving their goals, such as: *i*) removing the trusted setup or making it independent from the polynomial/function that needs to be verified (a candidate solution is to use a public-coin protocol that can be made non-interactive by using a random oracle) and, *ii*) analyzing the security of VCBF in the cost model [47].

Acknowledgments. We thank Irene Giacomelli and Luca Nizzardo for helpful discussions.

The authors were partially supported by Protocol Labs under the RFP-009 on Proof of Space and Useful Space. In addition, the second author was supported by the National Key R&D Program of China 2021YFB3100100 and CAS Project for Young Scientists in Basic Research Grant YSBR-035, the third author was supported by the Carlsberg Foundation under the Semper Ardens Research Project CF18-112 (BCM), and the fourth author was supported by Hong Kong Research Grants Council under grant GRF-16200721.

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., Blocki, J., Pietrzak, K.: Sustained space complexity. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 99–130. Springer (2018)
5. 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)
6. 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)
7. 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)
8. Aura, T., Nikander, P., Leiwo, J.: Dos-resistant authentication with client puzzles. In: International workshop on security protocols. pp. 170–177. Springer (2000)
9. Back, A.: Hashcash-a denial of service counter-measure (2002)
10. Bellare, M., Kane, D., Rogaway, P.: Big-key symmetric encryption: Resisting key exfiltration. In: Annual International Cryptology Conference. pp. 373–402. Springer (2016)
11. 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)
12. Biryukov, A., Khovratovich, D.: Egalitarian computing. In: Holz, T., Savage, S. (eds.) USENIX Security 2016. pp. 315–326. USENIX Association (Aug 2016)
13. 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)
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: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part I. LNCS, vol. 10031, pp. 126–158. Springer, Heidelberg (Dec 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.E.: Balloon hashing: A memory-hard function providing provable protection against sequential attacks. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part I. LNCS, vol. 10031, pp. 220–248. Springer, Heidelberg (Dec 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.) EUROCRYPT 2020, Part I. LNCS, vol. 12105, pp. 707–737. Springer, Heidelberg (May 2020)
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.) ACM CCS 2012. pp. 501–512. ACM Press (Oct 2012)
34. Fisch, B.: Tight proofs of space and replication. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part II. LNCS, vol. 11477, pp. 324–348. Springer, Heidelberg (May 2019)
35. 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)
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. Grunwald, P., Vitányi, P.: Shannon information and kolmogorov complexity. arXiv preprint cs/0410002 (2004)
39. 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)
40. Juels, A.: Client puzzles: A cryptographic countermeasure against connection depletion attacks. In: Proc. Networks and Distributed System Security Symposium (NDSS), 1999 (1999)
41. Kaliski, B.: Password-based cryptography specification. RFC 2898 (2000)
42. Kedlaya, K.S., Umans, C.: Fast modular composition in any characteristic. In: 2008 49th Annual IEEE Symposium on Foundations of Computer Science. pp. 146–155. IEEE (2008)
43. Li, M., Vitányi, P., et al.: An introduction to Kolmogorov complexity and its applications, vol. 3. Springer (2008)
44. Liu, Y., Pass, R.: On one-way functions and kolmogorov complexity. In: FOCS 2020, 61st Annual IEEE Symposium on Foundations of Computer Science
45. 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)
46. Merkle, R.C.: Secure communications over insecure channels. Communications of the ACM **21**(4), 294–299 (1978)
47. Moran, T., Orlov, I.: Simple proofs of space-time and rational proofs of storage. In: Annual International Cryptology Conference. pp. 381–409. Springer (2019)
48. Muchnik, A.A.: Kolmogorov complexity and cryptography. Proceedings of the Steklov Institute of Mathematics **274**(1), 193 (2011)
49. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
50. Neary, T., Woods, D.: Four small universal turing machines. Fundamenta Informaticae **91**(1), 123–144 (2009)
51. Papamanthou, C., Shi, E., Tamassia, R.: Signatures of correct computation. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 222–242. Springer, Heidelberg (Mar 2013)
52. Percival, C.: Stronger key derivation via sequential memory-hard functions (2009)
53. Pietrzak, K.: Simple verifiable delay functions. In: 10th innovations in theoretical computer science conference (itcs 2019). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2018)
54. Protocol Labs: Filecoin: A decentralized storage network (2017), https://filecoin.io/filecoin.pdf, last visited February 19, 2023.
55. Provos, N., Mazieres, D.: A future-adaptable password scheme. In: USENIX Annual Technical Conference, FREENIX Track. pp. 81–91 (1999)
56. 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)
57. Ren, L., Devadas, S.: Proof of space from stacked expanders. In: Theory of Cryptography Conference. pp. 262–285. Springer (2016)
58. Ren, L., Devadas, S.: Bandwidth hard functions for asic resistance. In: Theory of Cryptography Conference. pp. 466–492. Springer (2017)
59. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto (1996)
60. Souto, A., Teixeira, A., Pinto, A.: One-way functions using kolmogorov complexity. Proceedings of the Computability in Europe pp. 346–356 (2010)
61. 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)
62. 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)
63. Vitányi, P.: Personal webpage, https://homepages.cwi.nl/~paulv/kolmogorov.html

64. Wesolowski, B.: Efficient verifiable delay functions. Journal of Cryptology pp. 1–35 (2020)
65. Woods, D., Neary, T.: The complexity of small universal turing machines: A survey. Theoretical Computer Science **410**(4-5), 443–450 (2009)
66. 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. pp. 863–880. IEEE Computer Society Press (May 2017)
67. 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 3

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 2, 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 4

The proof follows the same strategy of [43, 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 $\alpha \in \{0,1\}^{|x|-c-e-v-1}$, 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 $\mathsf{T}'(\alpha, r) \overset{\text{def}}{=} \mathsf{T}(\alpha; r)$. 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 5

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

$$\Pr[(f(x_0), \ldots, f(x_d)) = \mathsf{T}(\alpha, x_0, \ldots, x_d; r)] = 1.^{18}$$

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:

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 $a = a_0 || \ldots || a_d$.

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

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:

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

where we used the fact that $|\alpha| \leq m = (d + 1)(\lambda - \ell_{in} - 2\log(\ell_{in}) - 1) - c' - \lambda - 2\log((d + 1)\lambda - c') - 2\log(\lambda + 1) - 4$ and $\log(m) \leq \log((d + 1)\lambda - c')$. This contradicts the fact that a is (c', ℓ_{rnd})-RND-incompressible.

A.4 Proof of Theorem 7

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 $(\epsilon', m, \ell_{rnd}, n)$-decomp-min-capacity (Definition 12) with respect to keys $(\mathsf{ek}, \mathsf{vk})$ where $\lambda \in \mathbb{N}, d \in \mathbb{N}, c \in \mathbb{N}, \epsilon_1 \in [0, 1]$,*

$$\begin{aligned}
m &= (d + 1)(\lambda - \ell_{in} - 2\log(\ell_{in}) - 1) - c' \\
&\qquad - \lambda - 2\log((d + 1)\lambda - c') - 2\log(\lambda + 1) - 4, \\
c' &= c + \ell_{rnd} + 2\log(\ell_{rnd}) + 1 + O(1), \\
\epsilon' &= \epsilon_1 + \frac{d + 1}{2^{\ell_{in}}}, \quad n = \frac{\epsilon_1 \cdot 2^{\ell_{in}}}{d} + 1.
\end{aligned}$$

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:

$$\begin{aligned}
&\Pr\left[\exists i \in [n], \mathsf{Eval}(\mathsf{ek}, x) = y_i \,\middle|\, \begin{array}{c} E^{\mathrm{RND}}_{c',\ell_{rnd}}, \\ x \leftarrow\!\!\$\, \{0,1\}^{\ell_{in}}, \\ \{y_i \leftarrow\!\!\$\, \mathsf{A}(\tau_i, x)\}_{i\in[n]} \end{array}\right] = \\
&\Pr\left[\exists i \in [n], \mathsf{Eval}(\mathsf{ek}, x) = y_i \,\middle|\, \begin{array}{c} E^{\mathrm{RND}}_{c',\ell_{rnd}}, \\ x \leftarrow\!\!\$\, \{0,1\}^{\ell_{in}}, \\ \{y_i = \mathsf{A}(\tau_i, x; r_i)\}_{i\in[n]} \end{array}\right] > \epsilon', \qquad (2)
\end{aligned}$$

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,

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

By leveraging Equation (2), we can conclude that $\Pr\big[x \in \mathcal{X} | x \leftarrow\!\!\$\ \{0,1\}^{\ell_{in}}\big] = \frac{|\mathcal{X}|}{2^{\ell_{in}}} > \epsilon'$ (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 $E^{\text{RND}}_{c',\ell_{rnd}}$, 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 5 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 7, we need to bound the probability of $E^{\text{RND}}_{c',\ell_{rnd}}$. Let E^{DET}_{c} be the following event

$$E^{\text{DET}}_{c} \overset{\text{def}}{=} \big(a \text{ is } c\text{-DET-incompressibile}\big)\,,$$

where $a = a_0||\dots||a_d$ are the coefficients of the polynomial $f_a(x)$ sampled by Setup. By using Theorem 4, 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 7. Since we only consider randomized adversaries with randomness space $\{0,1\}^{\ell_{rnd}}$, we have that E^{DET}_{c} implies $E^{\text{RND}}_{c',\ell_{rnd}}$, and $\neg E^{\text{RND}}_{c',\ell_{rnd}}$ implies $\neg E^{\text{DET}}_{c}$. Thus, $\Pr\big[\neg E^{\text{DET}}_{c}\big] \geq \Pr\Big[\neg E^{\text{RND}}_{c',\ell_{rnd}}\Big]$. Hence, we conclude that, given $r_i \leftarrow\!\!\$\ \{0,1\}^{\ell_{rnd}}$ and for $i \in [n]$, the following inequality holds:

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

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

$$\Pr\left[\exists i \in [n], \mathsf{Eval}(\mathsf{ek}, x) = y_i \,\middle|\, \begin{array}{c} E^{\text{RND}}_{c',\ell_{rnd}}, \\ x \leftarrow\!\!\$\ \{0,1\}^{\ell_{in}}, \\ \{y_i = \mathsf{A}(\tau_i, x; r_i)\}_{i\in[n]} \end{array}\right] \cdot \Pr\big[E^{\text{RND}}_{c',\ell_{rnd}}\big] \tag{3}$$

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

$$< \epsilon' + \frac{1}{2^c} - \frac{1}{2^{(d+1)\lambda}} = \epsilon_1 + \frac{d+1}{2^{\ell_{in}}} + \frac{1}{2^c} - \frac{1}{2^{(d+1)\lambda}} = \epsilon \tag{5}$$

where in Equation (3) we used the fact that Construction 1 satisfies $(\epsilon', m, \ell_{rnd}, n)$-decomp-min-capacity when $E^{\mathrm{RND}}_{c',\ell_{rnd}}$ occurs (Lemma 1) and $\Pr\left[E^{\mathrm{RND}}_{c',\ell_{rnd}}\right] \leq 1$. Conversely, in Equation (4) we used the fact that the advantage of A is at most 1 (even conditioned to $\neg E^{\mathrm{RND}}_{c',\ell_{rnd}}$) and $\Pr\left[\neg E^{\mathrm{RND}}_{c',\ell_{rnd}}\right] \leq \Pr\left[\neg E^{\mathrm{DET}}_{c}\right] < \frac{1}{2^c} - \frac{1}{2^{(d+1)\lambda}}$ when $a \leftarrow\!\!\$\; \{0,1\}^{(d+1)\lambda}$ (Theorem 3). This concludes the proof of Theorem 7.

A.5 Proof of Theorem 8

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 $(\epsilon, m, \ell_{rnd}, n_1)$-decomp-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

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

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 7-8. We now proceed with the case $1 < v \leq m$. Let n_1 as defined in Theorem 8. 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 8 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

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

By substituting $n_1 = \frac{\epsilon \cdot 2^{\ell_{in}}}{d} + 1$ as defined in Theorem 7 and setting $n = n_2$ we obtain the bound of n as defined in the Corollary 1.

A.7 Proof of Corollary 2

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.