Memory Checking Requires Logarithmic Overhead

Memory Checking Requires Logarithmic Overhead

Elette Boyle
Reichman University
Herzliya, Israel
NTT Research
Sunnyvale, USA
eboyle@alum.mit.edu

Ilan Komargodski
Hebrew University
Jerusalem, Israel
NTT Research
Sunnyvale, USA
ilank@cs.huji.ac.il

Neekon Vafa
Massachusetts Institute of Technology
Cambridge, USA
nvafa@mit.edu

ABSTRACT

We study the complexity of memory checkers with computational security and prove the first general tight lower bound.

Memory checkers, first introduced over 30 years ago by Blum, Evans, Gemmel, Kannan, and Naor (FOCS '91, Algorithmica '94), allow a user to store and maintain a large memory on a remote and unreliable server by using small trusted local storage. The user can issue instructions to the server and after every instruction, obtain either the correct value or a failure (but not an incorrect answer) with high probability. The main complexity measure of interest is the size of the local storage and the number of queries the memory checker makes upon every logical instruction. The most efficient known construction has query complexity $O(\log n/\log\log n)$ and local space proportional to a computational security parameter, assuming one-way functions, where n is the logical memory size. Dwork, Naor, Rothblum, and Vaikuntanathan (TCC '09) showed that for a restricted class of "deterministic and non-adaptive" memory checkers, this construction is optimal, up to constant factors. However, going beyond the small class of deterministic and non-adaptive constructions has remained a major open problem.

In this work, we fully resolve the complexity of memory checkers by showing that *any* construction with local space p and query complexity q must satisfy

$$p \geq \frac{n}{(\log n)^{O(q)}} .$$

This implies, as a special case, that $q \geq \Omega(\log n/\log\log n)$ in any scheme, assuming that $p \leq n^{1-\varepsilon}$ for $\varepsilon > 0$. The bound applies to any scheme with computational security, completeness $2/3$, and inverse polynomial in n soundness (all of which make our lower bound only stronger). We further extend the lower bound to schemes where the read complexity q_r and write complexity q_w differ. For instance, we show the tight bound that if $q_r = O(1)$ and $p \leq n^{1-\varepsilon}$ for $\varepsilon > 0$, then $q_w \geq n^{\Omega(1)}$. This is the first lower bound, for any non-trivial class of constructions, showing a read-write query complexity trade-off.

Our proof is via a delicate compression argument showing that a "too good to be true" memory checker can be used to compress random bits of information. We draw inspiration from tools recently developed for lower bounds for relaxed locally decodable codes. However, our proof itself significantly departs from these works, necessitated by the differences between settings.

STOC '24, June 24–28, 2024, Vancouver, BC, Canada

ACM ISBN 979-8-4007-0383-6/24/06
https://doi.org/10.1145/3618260.3649686

CCS CONCEPTS

• **Theory of computation** → Cryptographic protocols; Cryptographic primitives; • **Security and privacy** → **Mathematical foundations of cryptography**.

KEYWORDS

Lower Bound, Memory Checking, Computational Security

ACM Reference Format:
Elette Boyle, Ilan Komargodski, and Neekon Vafa. 2024. Memory Checking Requires Logarithmic Overhead. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing (STOC '24), June 24–28, 2024, Vancouver, BC, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3618260.3649686

1 INTRODUCTION

Consider a user who wishes to maintain and operate on a large database but does not have sufficient local memory. A natural idea is for the user to offload the database to a remote storage service and perform accesses remotely. This solution, however, introduces a trust concern: the user must trust the storage service to perform its accesses reliably. Blum, Evans, Gemmel, Kannan and Naor [4] addressed this issue more than 30 years ago by introducing the concept of a *memory checker*: a method for a user to use its small (but trusted) local storage to detect faults in the large untrusted storage service. Since cloud storage and cloud computing have become widespread and growing practices, the need to guarantee the integrity of remotely stored data is paramount. Beyond merely checking integrity of remotely stored data [4, 11, 12, 18, 28], memory checkers have found applications in various other real-world applications, for example, provable data possession and retrievability systems [3, 10, 20, 28, 35], various verifiable computation systems [5, 6, 8, 30, 34, 37, 39, 40], and many more.

A memory checker can be thought of as a proxy between the user and the untrusted remote storage. The checker receives from the user an adaptively generated sequence of read and write operations to a large unreliable memory. For each such logical request, it makes its own physical queries to the remote storage. The checker then uses the responses, together with a small reliable local memory, to either ascertain the correct answer to the logical request or report that the remote storage was faulty. The checker's assertions should be correct with high probability; typically, a small two-sided error is permitted. The main complexity measures of a memory checker are its **space complexity** (denoted p), the size of the reliable local memory in bits, and its **query complexity** (denoted q),

the number of physical queries made to the unreliable memory per logical user request.

Blum et al.'s main results are efficient memory checkers in two different settings, as stated next. Throughout, we use n to denote the (logical) memory size.

- **Construction 1**: Space complexity $p = O(\lambda)$, where λ is the size of a cryptographic key, and query complexity $q = O(\log n)$. These constructions assume that one-way functions exist and that the adversary who controls the unreliable memory is computationally efficient.
- **Construction 2**: Space complexity p and query complexity $q = O(n/p)$. This construction is statistically secure.

Naor and Rothblum [26] showed that any memory checker with a non-trivial query-space trade-off (i.e., $q = o(n/p)$) must be computationally secure and must be based on the existence of one-way functions. Thus, Construction 2 from above is optimal among all statistically secure constructions. Yet, whether more efficient computationally secure constructions than Construction 1 exist is a long-standing open problem. In particular, is the logarithmic query complexity necessary? In many applications, logarithmic overhead per query is a significant price to pay for data verification, so if more efficient constructions exist, they may be preferable.

While this problem has been open for more than three decades, only one work managed to make progress. This work, due to Dwork, Naor, Rothblum, and Vaikuntanathan [13], showed that logarithmic query complexity is inherent for a *restricted class of memory checkers*. They consider memory checkers where for each read/write operation made by the user, the locations that the checker accesses in the unreliable memory are fixed and known. They refer to such checkers as *deterministic and non-adaptive*. While this class captures many known memory checker constructions, it is obviously quite restrictive. For instance, a memory checker could be *non-deterministic*, i.e., decide on its queries to the unreliable memory in a probabilistic manner via randomness derived from freshly sampled coins. Alternatively, a memory checker could be *adaptive*, i.e., choose sequentially which locations in the remote storage it reads and writes, and choose these locations based on the contents of earlier read locations or its reliable local memory contents. Of course, a memory checker could be *both* non-deterministic and adaptive, potentially resulting with more efficient construction.

Indeed, there are many computational models where randomness and/or adaptivity are either necessary or make certain problems easier. For instance, in the context of two-party communication complexity it is well known (see [27]) that for every $k \in \mathbb{N}$, randomization is more powerful than determinism in k-round protocols, and further that having k rounds of adaptive communication is (exponentially) better than having only $k-1$ rounds. This shows that randomness and adaptivity are computational resources that have the potential to significantly improve the complexity of certain tasks. Additionally, in the context of *oblivious* RAM computation (which we shall discuss in length below), randomness is known to be necessary [17] and there is evidence that adaptivity is required in certain non-trivial regimes of parameters [9].

Thus, the main question we address in this work is as follows:

What is the achievable complexity of memory checkers?
Do memory checkers with sub-logarithmic query complexity exist (under any cryptographic assumption)?

1.1 Our Results

We fully resolve the complexity of memory checkers by showing that logarithmic query complexity is inherent, no matter how the scheme operates and no matter which computational assumptions are used. Specifically, we prove the following theorem.

Theorem 1 (Informal; see Theorem 5). *Every memory checker (with computational security) for a logical memory of size n that has query complexity q, and local state p, must satisfy $p \geq \frac{n}{(\log n)^{O(q)}}$.*

In particular, $q \geq \Omega(\log n/\log\log n)$ assuming that $p \leq n^{1-\varepsilon}$ for $\varepsilon > 0$. As mentioned, the above theorem applies to all possible schemes, including ones that use randomness and adaptivity to decide which locations to access in the remote storage, and also in the computational setting. Furthermore, the lower bound applies even to schemes where the local state is private (from the adversary). Lastly, the completeness of the memory checker in the theorem is $2/3$ and soundness is inverse polynomial in n. Note that these make our result only stronger because in constructions we usually aim for perfect (or near-perfect) completeness and negligible soundness.

The lower bound is tight up to the constant hidden in the $O(\cdot)$ notation. A memory checker construction with matching complexity, i.e., with $q = O(\log n/\log\log n)$, was given by Papamanthou and Tamassi [31]. However, their construction requires *private* local state (used to store a secret PRF key). We improve upon their result and show (for completeness) a construction with quantitatively matching complexity and with only (public) *reliable* local state. Our construction requires the existence of sub-exponentially secure one-way functions, as is needed in all other computationally secure memory checker constructions with similar properties.

Theorem 2 (Informal; see Corollary 5 in the full version). *Assume the existence of sub-exponentially secure one-way functions. For all sufficiently large $q \leq \log n/\log\log n$, there is a deterministic and non-adaptive memory checker for a logical memory of size n with* reliable *local state, perfect completeness, and (computational) negligible soundness in n, that has local space $p \leq \frac{n}{(\log n)^{\Omega(q)}}$.*

In particular, for some $q = \Theta(\log n/\log\log n)$, this construction has local space $p = \mathrm{polylog}(n)$.

We refer to Table 1 for a summary of known memory checker constructions.

Reads vs. writes. Our lower bound rules out memory checkers where the worst-case query complexity of all accesses is below (quasi-)logarithmic. However, not all types of accesses necessarily occur as often in applications. Depending on the application, it could be that read operations are far more frequent than writes, or vice versa. Our stated lower bound from above does not rule out a memory checker where the read complexity is, say, constant but writes have logarithmic complexity. In fact, no known lower bound, not even for restricted classes of constructions (e.g., deterministic and non-adaptive), rules out such a construction. We extend our lower bound from Theorem 1 to this setting and prove a general

Table 1: Complexity of memory checker constructions for a logical memory of size n.

Reference	Read Complexity	Write Complexity	Secret State	Remark
[4]	$O(\log n)$	$O(\log n)$	No	
[13]	$O(d \log_d n)$	$O(\log_d n)$	Yes	d is arbitrary
[13]	$O(\log_d n)$	$O(d \log_d n)$	Yes	d is arbitrary
[31]	$O(\log n / \log\log n)$	$O(\log n / \log\log n)$	Yes	
This work	$O(\log n / \log\log n)$	$O(\log n / \log\log n)$	No	

trade-off between the local reliable space, the read query complexity, and the write query complexity.

Theorem 3 (Informal; see Theorem 5). *Every memory checker (with computational security) for a logical memory of size n that has read-query complexity q_r, write-query complexity q_w, and local state p, must satisfy $p \geq \frac{n}{(q_r q_w \log n)^{O(q_r)}}$.*

In particular, the above theorem rules out a memory checker with constant read-query complexity and sub-polynomial write-query complexity, i.e., every memory checker with $q_r = O(1)$ and local space $p \leq n^{1-\Omega(1)}$ must have $q_w = n^{\Omega(1)}$. This bound is optimal due to a construction of Dwork et al. [13] who showed a construction with write-query (resp. read-query) complexity $O(d \log_d n)$ and read-query (resp. write-query) complexity $O(\log_d n)$ for every parameter d. Indeed, setting $d = n^{O(1)}$, we get a memory checker with $q_r = O(1)$ and $q_w = n^{O(1)}$, which is optimal according to our lower bound. Finally, we note that Theorem 1 is obtained as a special case of Theorem 3 with $q_r = q_w$.

Interestingly, we do not know if the "reverse" bound also holds in general, i.e., that if $q_w = O(1)$ then q_w must be large. We leave this as an intriguing open problem. We make initial steps towards this question by proving it for the restricted class of deterministic and non-adaptive memory checkers.

Theorem 4 (Informal; see Theorem 6 in the full version). *Every deterministic and non-adaptive memory checker (with computational security) for a logical memory of size n that has read-query complexity q_r, write-query complexity q_w, and local state p, must satisfy $p \geq \frac{n}{(q_r q_w \log n)^{O(\min\{q_w, q_r\})}}$.*

This theorem means that if either one of q_r or q_w are sub-(quasi)-logarithmic, then the other one must be much larger. In other words, the "reverse" bound of Theorem 3 holds for deterministic and non-adaptive constructions: if $q_w = O(1)$, then necessarily $q_r = n^{\Omega(1)}$ (as long as the local space is not too large, i.e., $p \leq n^{1-\Omega(1)}$). By the above-mentioned constructions of Dwork et al. [13] (that happen to be deterministic and non-adaptive), we conclude that our lower bound is tight, fully resolving the complexity of deterministic and non-adaptive memory checkers. Lastly, we mention that we provide some (weak) evidence that a "reverse" bound of Theorem 3 for general (not necessarily deterministic and non-adaptive) schemes would require relatively new ideas; see the full version for details.

We refer to Table 2 for a summary of known memory checker lower bounds.

Table 2: Lower bounds on the local space p of memory checkers for a logical memory of size n with q_r read query complexity and q_w write query complexity. "Det." is an abbreviation of "deterministic."

Reference	Space/Query Trade-off	Limitation
[13]	$p \geq \frac{n}{(\log n)^{O(\max\{q_r, q_w\})}}$	Det. & non-adaptive
This work	$p \geq \frac{n}{(\log n)^{O(q_r)}}$	None
This work	$p \geq \frac{n}{(\log n)^{O(\min\{q_r, q_w\})}}$	Det. & non-adaptive

1.2 Implications of our Lower Bounds

Lower bounds for memory checkers optimizing other metrics. As an immediate application of our lower bound, we get general lower bounds for memory checking in other models where different notions of efficiency are considered. We mention two recent works next. Mathialagan [24] extended the memory checking notion to deal with Parallel RAM (PRAM) machines, and suggested a construction for PRAMs with m CPUs with $O(\log N)$ query blowup and $O(\log N)$ depth blowup, relying on the existence of one-way functions. Since their constructions match (in query complexity) the best known construction in the RAM setting, along with our lower bound, we conclude that their scheme is optimal in this sense. Wang, Lu, Papamanthou, and Zhang [38] studied the locality of memory checkers (i.e., the number of non-contiguous memory regions a checker must query to verifiably answer a read or a write query). They adapted the lower bound of Dwork et al. [13] to conclude that $\Omega(\log n / \log\log n)$ locality is necessary for any deterministic and non-adaptive memory checker. Our lower bound implies (analogously) that the same lower bound applies to all possible schemes.

Impossibility of efficient black-box malicious Oblivious RAM compilers. As mentioned, memory checkers are used to solve the trust issue that arises when one offloads a memory to a remote and untrusted server. However, there is also a privacy concern in doing so which is not addressed by memory checkers. Since the remote server fully controls the memory and executes instructions in the user's behalf, then it can see the user's data and the program being executed. To obtain privacy, we need to hide the data (using say an encryption scheme) and also "scramble" the observed access patterns so that instructions look unrelated to the data or the program being executed. The tool that achieves the latter goal is called *Oblivious RAM* (ORAM), introduced in the seminal works of Goldreich and Ostrovsky [14, 17, 29].

The efficiency of ORAM schemes is measured (similarly to memory checkers) by the number of physical queries in the oblivious simulation per logical database query. We know that logarithmic overhead is unavoidable, for any construction, even ones that rely on cryptographic assumptions [7, 17, 21, 22]. We also have matching constructions with (worst-case) logarithmic overhead, where security is computational and relying on one-way functions [1, 2, 32].

We note however that in the ORAM setting, one typically assumes that the remote untrusted server is passive in the sense that it behaves honestly except that it tries to learn information about the underlying data or program from the observed access pattern. A natural question is whether it is possible to tolerate a malicious attacker that may not behave honestly and actually tamper with the memory while trying to learn something about the underlying data. It is well-known (e.g., [23]) that by naively compiling every instruction of the ORAM using a memory checker one obtains a "maliciously secure" ORAM that achieves both privacy and integrity, simultaneously. This however comes at a cost: every logical instruction will now require ORAMOverhead×MemoryCheckerOverhead physical accesses, which is roughly $\Theta(\log^2 n)$ (using [1, 4]).[1]

Mathialagan and Vafa [25] recently improved upon the above generic compiler and gave a construction (called MacORAMa) of a"maliciously secure" ORAM with overhead $O(\log n)$ (which is obviously optimal). They achieve their result by a tedious process of opening up every building block in [1, 2]'s construction and turning it into a maliciously secure building block using various constructions of memory checkers. As such, the construction is overall very long and complex. [25] ask if such a white-box construction and analysis is necessary or maybe there is a generic way to compile ORAMs into maliciously secure counterparts with only constant overhead. Towards this, a barrier was presented by [25]: such a "black-box" compiler would imply a memory checker with $O(1)$ query complexity. By our result (Theorem 1), such memory checkers do not exist, no matter what, and so there is no way to generically upgrade ORAMs into malicious with less than additional logarithmic blowup. See details and the formal statement in the full version.

It is crucial for this corollary that our lower bound in Theorem 1 applies to all constructions of memory checkers, including ones that use randomness and adaptivity. Indeed, the way Mathialagan and Vafa obtain the above-mentioned "barrier" is by using an ORAM and the malicious compiler to build a memory checker. Since ORAMs are inherently randomized and (as far as we know [9]) require adaptivity, their resulting memory checker is using randomness and adaptivity.

1.3 Offline vs. Online Memory Checkers

We have considered memory checkers that need to report either a correct value or an error (but never be wrong) after every logical instruction. This notion of memory checkers is known as "online" memory checkers. Prior works in this area additionally considered a weaker notion called "offline" memory checkers. The latter are required to report that some error has occurred only after a batch of requests. None of our lower bounds apply to this weaker notion, and this is not surprising: there exist offline memory checkers that achieve amortized $O(1)$ bandwidth (see [4] and [13, Section 5]); these are even statistically secure and do not require any cryptographic assumptions.

1.4 Organization

In Section 2, we provide a technical overview of our main lower bound. In Section 3, we prove our main lower bound. All other sections, including preliminaries, definitions, and other theorem statements mentioned in the introduction are deferred to the full version of the paper.

2 TECHNICAL OVERVIEW

We begin by describing the previous approach of Dwork et al. [13], which as we mentioned, only gives a lower bound for *deterministic and non-adaptive* memory checkers. We already mention that our approach significantly differs from theirs, and the main purpose of this part of the overview is to explain why their approach seems only applicable to the restricted class of deterministic and non-adaptive constructions.

At a high level, they proceed as follows. First, they prove a lower bound for a base case setting where the query complexity of the memory checker is 1; this is done by a compression argument. Then, they consider the general case, and show a reduction from high query complexity to lower complexity at the expense of decreasing the logical memory size, increasing the local space, and increasing the physical word size. The reduction is performed iteratively until they end up with a memory checker with query complexity 1, where they can invoke the base case lower bound. More details follow.

Base case. Suppose there is a deterministic and non-adaptive memory checker with query complexity 1: namely, a single *fixed* physical query takes place for each logical query. To obtain a lower bound for this base case, they invoke a compression argument, where Alice is to transmit a random string $x \sim \{0,1\}^n$ to Bob. Consider a sequence of logical operations that first writes 0 to each logical index, resulting in public database DB_0 and local state st_0. (Alice and Bob can share DB_0 and st_0 for free since there is no dependence on x.) Then, Alice uses the memory checker to write 1 to all $i \in [n]$ such that $x_i = 1$, resulting in public database DB_1 and local state st_1. Alice will send just st_1 to Bob.

Bob's decoding strategy is as follows: for all $i \in [n]$, use the memory checker to read logical index i using DB_0 and st_1. If the answer is a bit value $b \in \{0,1\}$ value, set $\tilde{x}_i = b$. Otherwise, if the answer is $\perp$, set $\tilde{x}_i = 1$. By soundness, for all i such that the answer is a bit value, we know $\tilde{x}_i = x_i$. By completeness,[2] since queries to logical indices i must induce disjoint physical queries (here relying on query complexity 1 and basic information encoding), if $x_i = 0$, then we must recover the bit value 0, since DB_0 and DB_1 are consistent at the corresponding ith physical query location. Therefore, for all $i \in [n]$, $\tilde{x}_i = x_i$. Since Alice and Bob communicated n bits of information, it follows by a counting argument that $|\mathsf{st}_1| \geq n$.

[1] For specific ORAM constructions such as Path ORAM there are more efficient composition methods, e.g., [33, 36], but since the ORAM itself is sub-optimal, they result in similar $\approx \log^2 n$ overall complexity.

[2] They assume throughout that logical reads and writes query the same physical locations, at the cost of a multiplicative factor of 2 blowup in query complexity in the ultimate scheme.

Reducing to the base case. Next, they show to reduce the query complexity of a general memory checker to 1, at the cost of degrading the other parameters (the logical memory size, the local space, and the physical word size). Roughly speaking, they condition on the following: either there is a set of sufficiently many physical locations that are sufficiently "heavy," in the sense that many logical queries touch that location, or there does not exist such a set. In either case, the assumption that the construction is deterministic and non-adaptive is crucial.

- If there does exist such a heavy set, then, they restrict the memory checker to logical indices that frequently touch that heavy set. In this case, they move the heavy part of the public database into the local state and reduce query complexity, at the cost of decreasing the logical memory size and increasing the local space.
- If there does *not* exist such a heavy set, they argue that one can prune off (not too many) logical indices so that the remaining logical indices have disjoint physical locations. By restricting the memory checker to these logical indices and increasing the physical word size, this becomes a memory checker with query complexity 1. (Then, the base case applies.)

Without significant new ideas, it seems hard to use the above template to go beyond deterministic and non-adaptive constructions, as we argue next. Both of the above main steps, i.e., the base case and the reduction to the base case, rely on this assumption. We would thus need to prove an analogue of the base case for general constructions and then devise a reduction that works even if the construction uses randomness and adaptivity. We believe that the harder task is the latter one. It is not at all clear how one could restrict logical indices in either of the two cases. In the first case, one would need to at least define the heavy set differently, and in the second case, because of randomness and adaptivity, it is possible that a memory checker can adaptively choose to have many logical queries overlapping in random locations, making no single location "heavy" and yet there will be no non-trivial subset of logical indices with disjoint physical locations.

2.1 Our Lower Bound

Like Dwork et al. [13], we will also use a compression argument using the local state of the memory checker, but this is where the resemblance of our proofs ends. Their proof restricts memory checkers into smaller and smaller ones, finally arriving at a base case compression argument. Ours, on the other hand, will directly run a compression argument in "one shot," without reducing the memory checker into smaller ones.

Below, let the memory checker use physical words (i.e., blocks) of size $w \leq \text{polylog}(n)$ bits. Furthermore, we let m denote the size (i.e., number of words) of the remote server.

Suppose Alice wishes to communicate a random string of $x \in \{0,1\}^n$ of Hamming weight k to Bob (where k is some parameter set later). Alice and Bob can first initialize a memory checker with "0"s logically written everywhere, producing some public database $DB_0 \in (\{0,1\}^w)^m$, independent of x. Then, Alice can use this memory checker to write the value "1" to each of these indices $i \in [n]$ where $x_i = 1$, which will result in some new, updated public database, $DB_1 \in (\{0,1\}^w)^m$. After doing this, Alice can send to Bob the resulting local state st_1 of the memory checker. Note, however, that Bob does not see the updated database entries DB_1.

Bob can then use the following decoding strategy to extract x: using st_1 from Alice and the (outdated) public database DB_0 from initialization, sequentially emulate a logical read operation on the memory checker for each $i \in [n]$, rewinding the local state st_1 back between each logical read. Each such query is equivalent to a true operation of the memory checker, with an adversarial remote memory that replaces the true DB_1 values with outdated ones from DB_0. From the syntax of the memory checker, for each i, Bob will receive either 0, 1, or $\perp$. By the *soundness* guarantee of the memory checker, if Bob receives 0 or 1 for some i, Bob knows it is the correct value of x_i (with high probability). However, Bob cannot conclude anything if receiving $\perp$ from the memory checker. Our hope will be to leverage the *completeness* guarantee of memory checker, which says that if the memory checker always sees correct (i.e., fresh) values of the public database—namely, DB_0 is equal to DB_1 in all queried locations—then Bob will derive the correct binary, non-$\perp$ value. The challenge remains of how to do so, given that DB_1 in fact could differ largely from DB_0.

More precisely, let $W \subseteq [m]$ represent the set of physical locations that were written to in Alice's k writes (so DB_0 and DB_1 differ only on W). In particular, $|W| \leq kq$, where q is the query complexity to remote memory for each logical request. If for a given read query, the physical locations accessed in $[m]$ indeed avoid W, then we would be done, since Bob's decoding strategy can recover x by invoking completeness (and soundness) of the memory checker. However, there is no reason that this guarantee should be true. For example, in Merkle-tree style constructions, the root of the Merkle tree is accessed for *all* logical queries.

One possible way to get around this is to partition the public database into "heavy" and "light" locations. For heavy locations (e.g., the root of a Merkle tree), one can add their contents to the local space (or equivalently, in the communication game, have Alice send the contents to Bob), and for the light locations (e.g., lower levels of a Merkle tree), we hope that W does not hit too many of them. It turns out, however, that such a naive approach will not give us a useful lower bound, as we explain a bit later (see Remark 1), after we develop some of our ideas further. Thus, we will need a more intricate query partitioning mechanism that we explain next.

Tri-partitioning the public database. We use a more fine-grained partition into *heavy*, *medium*, and *light* locations $z \in [m]$, denoted by the sets $H, M, L \subseteq [m]$, respectively, in the following sense: when taking a uniformly random logical index $i \sim [n]$ and reading it with local state st_1 and public database DB_1, and sampling a uniformly random one of the corresponding q physical queries, what is the probability that it equals z? For thresholds implicitly set later, H will contain the physical locations z with highest probability, L the lowest probability, and M everything in between. Importantly, the fact that we make the heavy and light sets further apart by introducing the middle set M is crucial for us; see Remark 1.

With this partition in hand, we adjust our communication protocol between Alice and Bob as follows. As mentioned above, we send the heavy locations in the clear; i.e., after Alice performs the k writes, Alice will additionally send over $DB_1|_H$ (instead

of just st_1), i.e., the updated contents of the physical locations in the heavy set H.[3]

Now, Bob has access to some "hybrid" public database $\widetilde{DB}$, defined as DB_1 on H and DB_0 on $L \cup M$, and Bob will try reading from $\widetilde{DB}$ instead of DB_0 (still using the updated local state st_1). The set of "bad" physical locations that we are worried about is $\text{BAD} := W \cap (M \cup L) \subseteq [m]$. As long as any given read from Bob avoids BAD in all of its q physical queries, we can invoke completeness to say that Bob will receive the correct answer for that logical index.

We can decompose BAD into $(W \cap M) \cup (W \cap L)$ and analyze both cases separately. First, the medium thresholds for M will have the property that for random $i \sim [n]$, the probability that *all* q physical queries for logical read i avoid M is at least (say) $99/100$. Second, the light threshold for L will have the property that for all $\ell \in L$, the probability (over random $i \sim [n]$ and a random one of the q physical queries) that the location is ℓ is at most δ. By a union bound, this means that a random read to $i \sim [n]$ will hit ℓ for *at least one* of the q physical queries with probability at most $q\delta$. Since $W \cap L \subseteq L$ and $|W \cap L| \leq kq$, the probability that a random logical read to $i \sim [n]$ hits $W \cap L$ is at most $k\delta q^2$. We now set $k = \Theta(1/(\delta q^2))$ so that this probability is at most (say) $1/100$. Therefore, in total, the probability that a random read to $i \sim [n]$ avoids BAD is at least $98/100$. This implies that Bob will be able to recover at least $\Omega(k)$ bits of information about the random string x. Thus, by sending st_1 and $DB_1|_H$, Alice has communicated $\Omega(k)$ bits about x.

Let $p = |\mathsf{st}_1|$ denote the local space of the memory checker. By a standard encoding lemma, saying that the most communication efficient way to transmit uniformly random ℓ bits of information is by sending them in the clear, we get the inequality

$$p + |H|w \geq \Omega(k).$$

Rearranging and using our setting of $k = \Theta(1/(\delta q^2))$, this means

$$p \geq \frac{1}{\Theta(\delta q^2)} - |H|w. \tag{1}$$

Remark 1 (On the necessity of the medium set). *We now explain why we need a "medium" set, M. If we had just a heavy and a light set, the same analysis as above applies. However, it is possible that $|H|w$ and $1/\Theta(\delta q^2)$ do not have enough of a gap, in which case the right-hand side of (1) becomes very small. The medium set allows for this gap to be large.*

To explicitly see why this approach fails if there is no medium set, consider the following setup. Suppose $w = 4\ln(m)$ and we have a distribution[4] over $[m]$ elements as follows:

$$\Pr[i] := \begin{cases} \frac{1}{2} & i = 1, \\ \frac{1 \pm o(1)}{2\ln(m)(i-1)} & i \geq 2. \end{cases}$$

For all choices of L and δ, as long as $L \neq \emptyset$, we have $1/\delta - |H|w \leq 2$, which is tight when $L = [m]$ and $\delta = 1/2$. (Note that if $L = \emptyset$, then Alice is sending the whole updated database to Bob, which has no compression.) This renders (1) useless.

[3]More precisely, for $H \subseteq [m]$ and $DB_1 \in (\{0,1\}^w)^m$, we define $DB_1|_H \in (\{0,1\}^w)^{|H|}$ to be the restriction of DB_1 to indices in H.

[4]This distribution comes from Goldreich [15].

All that is left is the following: how exactly do we set our thresholds for H, M, L (and thus δ and $|H|$) to maximize the right hand side, where the probability that all q physical queries avoid M is at least $99/100$? To do this, we prove a generic partition lemma:

Lemma 1 (Partition Lemma (informal); see Lemma 2). *Let X be a random variable supported on a finite set S. Let $\gamma > 1$, and let $c, n \in \mathbb{N}$. Then, there is a partition $S = L \sqcup M \sqcup H$ such that the following hold for some $\delta \geq 1/n$:*

- $\Pr[X = \ell] \leq \delta$ *for all* $\ell \in L$,
- $\Pr[X \in M] \leq 1/c$, *and*
- *The set H satisfies*

$$\frac{1}{\delta} - |H|\gamma > \frac{n}{(2\gamma)^c}.$$

Now, we can directly use Lemma 2 with $S = [m]$, $\gamma = \Theta(q^2 w)$, and $c = 100q$ on the distribution over $[m]$ described earlier. Plugging this back into (1), we immediately have

$$p > \frac{n}{(q^2 w)^{O(q)}}.$$

In particular, if $w \leq \text{polylog}(n)$ and $p \leq n^{1-\epsilon}$ for some $\epsilon > 0$, then $q = \Omega(\log n / \log\log n)$.

Read and write query complexities. We can modify the above analysis to the setting where the read query complexity (q_r) and the write query complexity (q_r) differ. Then, we set $c = 100q_r$ and $k = \Theta(1/\delta q_r q_w)$, which arrives at

$$p > \frac{n}{(q_r q_w w)^{O(q_r)}}.$$

Therefore, for example, if $q_r = O(1)$, $p \leq n^{1-\epsilon}$ for some $\epsilon > 0$, and $w \leq \text{polylog}(n)$, then $q_w = n^{\Omega(1)}$. See Corollary 2 for more details.

Loose ends. There are a couple of issues swept under the rug in the above exposition.

- First, the distribution over physical indices produced by the logical read can adaptively change after each write, so Bob does not know what H, M, or L are.
- Second, our setting of parameters is circular. That is, we set $k = \Theta(1/(\delta q^2))$, and k is part of the description of the communication game that Alice and Bob are playing. However, δ is only given after applying the partition lemma (Lemma 2), which itself depends adaptively on the behavior of the memory checker.

For the first issue, we observe that Bob only needs to know what H is, so that Bob can produce $\widetilde{DB}$. Specifying $H \subseteq [m]$ can be done with $|H| \cdot \lceil \log_2 m \rceil$ bits, so Alice can send this as well in her message to Bob for free as long as $w \geq \Omega(\log n)$ and $m \leq \text{poly}(n)$.[5]

For the second (and more challenging) issue, we modify the communication game as follows. Alice and Bob together effectively "guess" a good value of δ before beginning the protocol. In doing so, Alice can also always feed the memory checker the same number of writes (regardless of δ), and will instead set the "initialized" database DB_0 to be after some number of writes depending on δ. Thankfully,

[5]Note that the assumption that $m \leq \text{poly}(n)$ is somewhat without loss of generality. Indeed, any construction where $m = n^{\omega(1)}$ can be generically transformed into a construction where the physical database size is $\text{poly}(n)$ (by "hashing" the memory space via a pseudorandom function). See the full version for more details.

our proof of the partition lemma additionally guarantees that there are most $c = \Theta(q)$ possible choices for δ, so guessing δ correctly occurs with noticeable $\Theta(1/q)$ probability.

Connection to Relaxed Locally Decodable Codes. Our technique for partitioning $[m]$ into heavy, medium, and light queries is inspired from a work of Goldreich [15] in revisiting lower bounds on the length of relaxed locally decodable codes (rLDCs).[6] In their setting, partitioning indices of the codeword into heavy, medium, and light is done in a different manner and for different reasons. For example, instead of having Alice send Bob the heavy set (as is done above), in the rLDC setting, each logical index $i \in [n]$ has a *different* corresponding heavy set H_i, so Alice cannot afford to send the database for all H_i. Instead, in the rLDC setting, Bob enumerates over all possible choices of $DB_1|_{H_i}$ and checks a certain consensus condition. Furthermore, in the rLDC setting, there is no notion of "writes," as encodings of different messages should have large Hamming distance. As a result, there's no equivalent of a DB_0 that makes sense for the rLDC setting, as there are no local ways to update the codeword. On the other hand, rLDCs are secure against all computationally unbounded adversaries that change a certain fraction of the codeword, whereas memory checkers need to be secure only for computationally bounded adversaries that can tamper with the whole memory.

3 MAIN LOWER BOUND

We begin by describing our main technical theorem:

Theorem 5 (Main Theorem). *Consider a memory checker for a logical memory of size n and with logical word size $w_\ell = 1$, supporting $2n$ logical queries. Assume that the physical database is of size m and with physical word size w, the read query complexity is q_r, the write query complexity is q_w, and the local space is p. If completeness is $99/100$ and soundness is $1/(10^4 \cdot n \cdot q_r \cdot 2^{q_r})$, then there is a universal constant $C \le 10^3$ such that for $n \ge C$,*

$$p \ge \frac{n}{(Cq_rq_w(w+\log m))^{C\cdot q_r}} - \log q_r - C.$$

Next, we state a couple of corollaries of this theorem. These corollaries, together with Theorem 5, are the precise and elaborate versions of Theorems 1 and 3 from the introduction. The first corollary gives a (quasi-)logarithmic lower bound on the worst-case query complexity of every memory checker where the reads and writes cost the same. The second corollary allows us to conclude that if the read query complexity is small, then the write complexity must be large. In the full version, we show how these corollaries follow from Theorem 5. We prove Theorem 5 in Section 3.1.

Corollary 1. *In the setting of Theorem 5 and further assuming that $q = \max\{q_w, q_r\}$, $m \le \mathrm{poly}(n)$, and $w \le \mathrm{polylog}(n)$, if completeness is $2/3$ and soundness is $1/n^{1+o(1)}$, it holds that*

$$p \ge \frac{n}{(\log n)^{O(q)}} - O(\log\log n).$$

In particular, if $p < n^{1-\epsilon}$ for some $\epsilon > 0$, then $q \ge \Omega(\log n/\log\log n)$.

[6] A locally decodable code is an error correcting code that allows for recovery of any particular input bit of the message with a small number of queries to the possibly corrupted codeword. A *relaxed* locally decodable code (rLDC) is one that allows the recovery procedure to output $\perp$ if the codeword is indeed corrupted.

Corollary 2. *In the setting of Theorem 5 and further assuming that $m \le \mathrm{poly}(n)$, $w \le \mathrm{polylog}(n)$, and $p \le n^{1-\varepsilon}$ for some $\varepsilon > 0$, if completeness is $2/3$ and soundness is $1/n^{1+o(1)}$, it holds that,*

- *If $q_r = o(\log n/\log\log n)$, then $q_w = (\log n)^{\omega(1)}$.*
- *If $q_r = O(1)$, then $q_w = n^{\Omega(1)}$.*

Remark 2 (On super-polynomial public database size). *It makes sense to assume that m is bounded by some a priori unspecified polynomial in n (i.e., $m \le \mathrm{poly}(n)$, as we assume in Corollaries 1 and 2), and in this case the lower bounds from Theorem 5 and Corollaries 1 and 2 are unconditional. But what if m is super-polynomial in n? Interestingly, we can still get a meaningful result. First, one can generically reduce the public database size from $m \le 2^{\mathrm{poly}(n)}$ to $m \le \mathrm{poly}(n)$ in any memory checker construction, by hashing the address space using a PRF (see the full version for details). This transformation, on its own, relies on the existence of one-way functions, but we recall that a memory checker satisfying a non-trivial relationship between p and q (i.e., $p \cdot q = o(n)$) already implies the existence of (infinitely often) one-way functions [26] which in turn can be used to get an (infinitely often) PRF [16, 19].*

3.1 Proof of Theorem 5

As explained in the technical overview, the proof proceeds by a compression argument where we utilize a memory checker to convey information from Alice to Bob. Furthermore, the main technical tool that we use is a partition lemma that allows us to classify and split physical locations into heavy, medium, and light. We first state and prove the partition lemma inspired by [15, Claim 2.6] and then proceed with the main proof.

Lemma 2. *Let X be a random variable supported on a finite set S. Let $\gamma > 1$, and let $c, n \in \mathbb{N}$. Then, there is a partition $S = L \sqcup M \sqcup H$ such that the following hold for some $\delta \ge 1/n$:*

- *$\Pr[X = \ell] \le \delta$ for all $\ell \in L$,*
- *$\Pr[X \in M] \le 1/c$, and*
- *The set H satisfies*

$$\frac{1}{\delta} - |H|\gamma > \frac{n}{(2\gamma)^c}.$$

Moreover, there exists $i \in [c]$ such that the conditions above hold for $\delta = (2\gamma)^{i-1}/n$.

Looking ahead, we use this last property of δ to argue that for fixed γ, c and n, one can guess a valid value of δ with probability $1/c$ without knowing the distribution of X.

Proof of Lemma 2. We define the sets

$$B_1 := \left\{ s \in S : \Pr[X = s] \in \left[0, \frac{2\gamma}{n}\right) \right\},$$

$$B_i := \left\{ s \in S : \Pr[X = s] \in \left[\frac{(2\gamma)^{i-1}}{n}, \frac{(2\gamma)^{i}}{n}\right) \right\} \text{ for } 2 \le i \le c-1,$$

$$B_c := \left\{ s \in S : \Pr[X = s] \in \left[\frac{(2\gamma)^{c-1}}{n}, \infty\right) \right\},$$

where we have $S = \bigsqcup_{i\in[c]} B_i$ by construction. By an averaging argument, we know there must exist some particular index $j \in [c]$ such that $\Pr[X \in B_j] \le 1/c$. We then set $L = \bigcup_{i\le j-1} B_i$, $M = B_j$, and $H = \bigcup_{i\ge j+1} B_i$. Clearly, $\Pr[X \in M] \le 1/c$ by construction.

Since for all $h \in H$ we have $\Pr[X = h] \geq (2\gamma)^j/n$, by summing over $h \in H$, we know $|H| \leq n/(2\gamma)^j$. On the other hand, by definition of L, we know that for all $\ell \in L$, $\Pr[X = \ell] < (2\gamma)^{j-1}/n$. (Note that this is vacuously true for $j = 1$.) We have $(2\gamma)^{j-1}/n \geq 1/n$ since $\gamma > 1$ and $j \geq 1$, so we can set $\delta = (2\gamma)^{j-1}/n$. Combining the inequalities above, we have

$$\frac{1}{\delta} - |H|\gamma \geq \frac{n}{(2\gamma)^{j-1}} - \frac{n}{(2\gamma)^j} \cdot \gamma = \frac{n}{2 \cdot (2\gamma)^{j-1}} > \frac{n}{(2\gamma)^c},$$

as desired. □

We now proceed with the proof of the main theorem, Theorem 5.

We start by setting a few parameters in anticipation of applying the partition lemma (Lemma 2) to a distribution over physical memory locations. Let $c = 100q_r$ and $\gamma = 200q_rq_w(w + \log m + 2)$ as needed for Lemma 2. For $j \in [c]$, let $\delta_j := (2\gamma)^{j-1}/n$ as given by Lemma 2, and let $k_j := \lfloor 1/(100\delta_j q_r q_w) \rfloor$, where one should think of k_j (for $j \in [c]$) as the number of 1s written to the memory checker. Later on in the proof, the choice of these parameter settings will become more clear. (Specifically, c and k_j are set by Claim 2, and γ is set at the end of the proof.) Observe that since $\delta_j \geq 1/n$, we have $k_j \leq 1/(100\delta_j) \leq n/100$. Throughout, we assume n is a multiple of 10 for simplicity, but our proof can be easily modified to handle arbitrary, sufficiently large n.

Using the memory checker, we directly construct a public-coin protocol for Alice and Bob to send c strings $x^{(1)}, \cdots, x^{(c)}$, distributed as follows: for each $j \in [c]$, $x^{(j)} \in \{0,1\}^{9n/10+k_j}$ is independently chosen uniformly at random subject to the constraint that $\|x^{(j)}\|_0 = k_j$ (i.e., the Hamming weight of each $x^{(j)}$ is k_j). At a high level, the size of Alice's message will be closely related to the local space p of the memory checker, so success of this protocol will yield a lower bound on p.

Protocol description. Before Alice sees $x^{(1)}, \cdots, x^{(c)}$, Alice and Bob together (using shared randomness) run a memory checker and logically write 0 to all indices $i \in [n]$. Then, Alice and Bob together sample $j^* \sim [c]$ independently and uniformly at random. We urge the reader to think of j^* as a guess for $j \in [c]$ for which $\delta = (2\gamma)^{j-1}/n$ will appear when applying Lemma 2 later on in the protocol. Since the memory checker could be adaptive, we may not know the right value of j beforehand, so we guess it. For simplicity, on a first pass, one may think of j as being "fixed" to the right value, although Alice has no way of knowing this beforehand.

Alice and Bob then sample uniformly random $y \in \{0,1\}^n$ with $\|y\|_0 = n/10 - k_{j^*}$ and logically write 1 to all indices $i \in [n]$ such that $y_i = 1$ in a uniformly random order. This gives Alice and Bob the public database $DB_0 \in (\{0,1\}^w \cup \{\perp\})^m$ after these writes as well as the local state $\mathsf{st}_0 \in \{0,1\}^p$. (This can all be formalized by Alice and Bob sharing a sufficiently long uniformly random string and running the memory checker on that string.)

For the remainder of the protocol, Alice and Bob agree on some mapping $\pi : [9n/10+k_{j^*}] \to [n]$ that maps $[9n/10+k_{j^*}]$ bijectively to the indices $i \in [n]$ where $y_i = 0$. For notational simplicity, we set $x = x^{(j^*)}$ and $k = k_{j^*}$ for the rest of the proof.

Alice's encoding. In short, for all $j \neq j^*$, Alice will directly encode $x^{(j)}$, but for $j = j^*$, Alice will write 1 to logical indices $\pi(x) := \{\pi(i) : i \in [9n/10 + k], x_i = 1\}$ of the memory checker in a random order and send some information related to the memory checker at the end of the writes.

More precisely, Alice tosses coins and uses the memory checker (with DB_0 and st_0) to write 1 to every logical index in the set $\pi(x)$ in a uniformly random order. This produces a new public database DB_1 and local state st_1. Now, Alice defines a distribution $\mathcal{D}$ over $[m]$ as follows:

(1) Sample $i \sim [n]$ uniformly at random.
(2) Use the memory checker to read index i from local space st_1 and public database DB_1. This defines a distribution over sequences of length q_r of the physical database locations, corresponding to the locations accessed for logical read i. Sample such a sequence R from this distribution (i.e., $|R| = q_r$).
(3) Finally, sample $v \sim R$ uniformly at random and output v.

Alice now applies Lemma 2 to this distribution $\mathcal{D}$ with parameters c and γ to partition the physical locations into $[m] = L \sqcup M \sqcup H$ and getting a parameter δ of the form $\delta = (2\gamma)^{\widetilde{j}-1}/n$ for some $\widetilde{j} \in [c]$ (e.g., choosing the smallest possible $\widetilde{j}$ for which the above holds). If $\widetilde{j} \neq j^*$, Alice aborts and the whole protocol fails (This can be formalized by Alice sending the all 0s string.) See Figure 1 for explicit details.

We make the following claim, whose proof we defer to the full version:

Claim 1. *The random variables j^* and $\widetilde{j}$ are independent. In particular, $\Pr\left[\widetilde{j} = j^*\right] = 1/c$, where the probability is over all randomness sampled by Alice and Bob in the protocol.*

For the rest of the protocol description, we condition on the event that $\widetilde{j} = j^*$. Thus, Alice sends the following information to Bob:

- A direct encoding of $\left(x^{(1)}, \cdots, x^{(j^*-1)}, x^{(j^*+1)}, \cdots, x^{(c)}\right)$,
- The updated local state st_1,
- A description of $H \subseteq [m]$,
- $DB_1|_H$, i.e., the contents of DB_1 at the locations in H, and
- Some auxiliary string aux (specified later in the proof) which will help Bob complete Alice's information into a full description of x.

Because the physical database has size m, the description of H can be represented using $\lceil \log \binom{m}{|H|} \rceil \leq |H| \lceil \log(m) \rceil$ bits. As such, the total size of this message (in bits) can be upper bounded[7] by

$$\log\left(\prod_{j \in [c], j \neq j^*} \binom{9n/10 + k_j}{k_j}\right) + p + |\mathsf{aux}| + |H|(w + \log m + 2) + 1.$$

Bob's decoding. Given H and $DB_1|_H$ from Alice, Bob can recreate some partially updated public database $\widetilde{DB} \in (\{0,1\}^w \cup \{\perp\})^m$, which is defined as DB_1 on locations in H and DB_0 on $\overline{H} = L \sqcup M$. In short, Bob's strategy will simulate the memory checker on the next logical read to all possible $i \in [n]$, rewinding back each time, using $\widetilde{DB}$ and st_1 from Alice.

[7] As a technicality, as stated, the length of this message is not fixed, but rather is a random variable that depends on the protocol's execution. Our proof, essentially, will show that with sufficiently high probability, this random variable can be upper bounded by a small enough fixed quantity to invoke the communication complexity lower bound and obtain a lower bound on $p = |\mathsf{st}_1|$.

Shared by Alice and Bob: $j^*, DB_0, \mathrm{st}_0, y, \pi$
Alice's Private Input: $\left(x^{(1)}, \cdots, x^{(c)}\right)$

(1) Let $x := x^{(j^*)}$, and let $x^{(-j^*)}$ be a direct encoding of $\left(x^{(1)}, \cdots, x^{(j^*-1)}, x^{(j^*+1)}, \cdots, x^{(c)}\right)$.
(2) Let $\pi(x) := \{\pi(\ell) \in [n] : \ell \in [9n/10 + k_{j^*}], x_\ell = 1\}$.
(3) Using DB_0 and st_0, use the memory checker to write "1" to all indices in $\pi(x) \subseteq [n]$ in a uniformly random order. This results in updated DB_1 and st_1.
(4) Using DB_1 and st_1, apply Lemma 2 to the resulting distribution $\mathcal{D}$ to get $[m] = H \sqcup M \sqcup L$ and $\widetilde{j} \in [c]$.
(5) If $j^* \neq \widetilde{j}$, immediately abort the whole protocol.
(6) Run Steps 2-4 of Bob's strategy (Figure 2) to generate the subset $U \subseteq [n]$.
(7) Let aux be an encoding of the subset $U \cap \pi(x) \subseteq U$.

Output: $\left(x^{(-j^*)}, \mathrm{st}_1, H, DB_1|_H, \mathrm{aux}\right)$.

Figure 1: Alice's Encoding Strategy.

More formally, let $r \in \{0,1\}^t$ denote the random bits used by the memory checker (for this final read). Let $MC(i, DB, \mathrm{st}; r) \in \{0, 1, \perp\}$ denote the output of the memory checker upon performing a logical read to index $i \in [n]$ from local state st and public database DB using randomness r. Let $\mathsf{BitMajority} : \{0,1,\perp\}^{2^t} \to \{0,1,\perp\}$ be the function that takes in the *bit-wise* majority of the inputs, meaning outputting the majority bit if such a bit exists, and otherwise, outputting $\perp$. For each $i \in [n]$, Bob will set

$$\tilde{z}_i := \mathsf{BitMajority}\left(MC\left(i, \widetilde{DB}, \mathrm{st}_1; r\right)_{r \in \{0,1\}^t}\right).$$

That is, Bob will brute force over all $i \in [n]$ and random bits used by the memory checker for this read, and Bob will deduce a bit value $\tilde{z}_i$ for z_i, the ith logical bit of the memory checker, if there is a strict majority of choices of randomness that agree on a bit (not $\perp$) to output. Otherwise, Bob will put some placeholder value for the ith bit and use aux from Alice to fill it in. Reading off the values at the indices in $\pi([9n/10 + k]) \subset [n]$ will then yield Bob's output $\tilde{x} \in \{0,1\}^{9n/10+k}$. See Figure 2 for explicit details. Note that Bob's decoding strategy may not be computationally efficient; however, this is not an issue for the information theoretic compression argument.

We now analyze how successful Bob will be. At a high level:

- We use completeness of the memory checker to argue that whenever $\widetilde{DB}$ and DB_1 are consistent for a read (which will happen pretty often per Claim 2 below), then Bob learns z_i.
- We use soundness to argue that Bob never learns the wrong value of *any* z_i (but could get $\perp$ from the memory checker).

This communicates some information from Alice to Bob, which we show is enough to get the desired lower bound. The string aux is used so that Alice can complete Bob's partial information into full information about x.

Recall that DB_0 and DB_1 can differ only at locations that Alice accessed in writing indices $\pi(i)$ where $x_i = 1$. Since $\|x\|_0 \leq k$, this is at most $k \cdot q_w$ locations, which we will call $W \subseteq [m]$. Define $\mathrm{BAD} := W \cap (L \cup M) = W \cap \overline{H}$ to be the set of locations that Alice

Shared by Alice and Bob: $j^*, DB_0, \mathrm{st}_0, y, \pi$
Alice's Message: $\left(x^{(-j^*)}, \mathrm{st}_1, H, DB_1|_H, \mathrm{aux}\right)$.

(1) Parse $x^{(-j^*)}$ as $\left(x^{(1)}, \cdots, x^{(j^*-1)}, x^{(j^*+1)}, \cdots, x^{(c)}\right)$.
(2) Using DB_0 from the shared input and H and $DB_1|_H$ from Alice, define the database $\widetilde{DB} \in (\{0,1\}^w \cup \{\perp\})^m$ as follows:

$$\widetilde{DB}[v] := \begin{cases} DB_1[v] & \text{if } v \in H, \\ DB_0[v] & \text{otherwise.} \end{cases}$$

(3) For all $i \in [n]$, let

$$\tilde{z}_i := \mathsf{BitMajority}\left(MC\left(i, \widetilde{DB}, \mathrm{st}_1; r\right)_{r \in \{0,1\}^t}\right) \in \{0, 1, \perp\}.$$

(4) Define $U := \{i \in [n] : \tilde{z}_i = \perp\}$.
(5) Parsing $\mathrm{aux} \subseteq U$, define $z \in \{0,1\}^n$ by

$$z_i := \begin{cases} \tilde{z}_i & \text{if } \tilde{z}_i \neq \perp, \\ 1 & \text{if } \tilde{z}_i = \perp \text{ and } i \in \mathrm{aux}, \\ 0 & \text{if } \tilde{z}_i = \perp \text{ and } i \notin \mathrm{aux}. \end{cases}$$

(6) Define $x \in \{0,1\}^{9n/10+k_{j^*}}$ by $x_\ell := z_{\pi(\ell)} \in \{0,1\}$.

Output: $\left(x^{(1)}, \cdots, x^{(j^*-1)}, x, x^{(j^*+1)}, \cdots, x^{(c)}\right)$.

Figure 2: Bob's Decoding Strategy.

wrote to that are not included in H. That is, DB_1 and $\widetilde{DB}$ differ only at physical locations in BAD. (We emphasize that Bob does not know the set BAD.)

We now argue that Bob's physical queries avoid BAD with constant probability. Let $R(i, DB, \mathrm{st}; r) \subseteq [m]$ denote the set of (at most q_r) physical locations queried by the memory checker upon performing logical read to index $i \in [n]$ using public database DB, local state st, and internal randomness r.

Claim 2. *Assuming $\widetilde{j} = j^*$,*

$$\Pr_{i \sim [n], r \sim \{0,1\}^t}\left[R\left(i, DB_1, \mathrm{st}_1; r\right) \cap BAD = \emptyset\right] \geq \frac{98}{100}.$$

Proof. Since $\mathrm{BAD} = (W \cap M) \cup (W \cap L)$, we can argue the two cases separately.

For $X \sim \mathcal{D}$, we know that $\Pr[X \in M] \leq 1/c = 1/(100q_r)$. By construction of $\mathcal{D}$ and the union bound, this implies

$$\begin{aligned} &\Pr_{i \sim [n], r \sim \{0,1\}^t}\left[R(i, DB_1, \mathrm{st}_1; r) \cap (W \cap M) \neq \emptyset\right] \\ &\leq \Pr_{i \sim [n], r \sim \{0,1\}^t}\left[R(i, DB_1, \mathrm{st}_1; r) \cap M \neq \emptyset\right] \\ &\leq q_r \cdot \Pr_{X \sim \mathcal{D}}[X \in M] \leq \frac{1}{100}. \end{aligned}$$

For the other case, by definition of L and δ_{j^*}, and for $X \sim \mathcal{D}$, we know that

$$\Pr_{X \sim \mathcal{D}}[X \in W \cap L] = \sum_{\ell \in W \cap L} \Pr_{X \sim \mathcal{D}}[X = \ell] \leq |W \cap L| \delta_{j^*}.$$

Thus, by construction of $\mathcal{D}$,

$$\begin{aligned}&\Pr_{i\sim[n],r\sim\{0,1\}^t}[R(i,DB_1,\mathrm{st}_1;r)\cap(W\cap L)\neq\emptyset]\\&\leq q_r\cdot\Pr[X\in W\cap L]\leq q_r\delta_{j^*}|W\cap L|\\&\leq q_r\delta_{j^*}|W|\leq kq_wq_r\delta_{j^*}\leq\frac{1}{100},\end{aligned}$$

where the last inequality holds since $k=k_{j^*}=\lfloor 1/(100\delta_{j^*}q_rq_w)\rfloor$. By a union bound over both cases, since $\mathrm{BAD}=(W\cap M)\cup(W\cup L)$, we have

$$\Pr_{i\sim[n],r\sim\{0,1\}^t}[R(i,DB_1,\mathrm{st}_1;r)\cap\mathrm{BAD}\neq\emptyset]\leq\frac{1}{100}+\frac{1}{100}=\frac{2}{100}.$$

□

Given the above claim, we can invoke completeness of the memory checker, since avoiding BAD means that the physical locations accessed are correct. Explicitly, we invoke completeness on the sequence of logical queries that writes 0 everywhere, writes 1 to $n/10$ random locations denoted by z, and lastly reads to a random location, denoted by i. We will use completeness for the last logical read, i.e., query number $n+n/10+1$.

Let r_1 and r_2 denote the randomness of the memory checker used for all of the logical writes and randomness of the memory checker used for the final logical read, respectively. Let z_i denote the ith logical bit of the memory checker, i.e., $z_i=y_i\vee 1[i\in\pi(x)]$. By completeness, we know that

$$\Pr_{r_1,r_2,z,i}[MC(i,DB_1,\mathrm{st}_1;r_2)=z_i]\geq\frac{99}{100},$$

where we emphasize that DB_1 and st_1 are random variables that depend on r_1 and z. Since the event that $\widetilde{j}=j^*$ is independent of the memory checker by Claim 1, we know

$$\Pr_{r_1,r_2,z,i}\left[MC(i,DB_1,\mathrm{st}_1;r_2)=z_i\;\middle|\;\widetilde{j}=j^*\right]\geq\frac{99}{100}.$$

Given $\widetilde{j}=j^*$, we can invoke Claim 2 and the inclusion-exclusion principle to get

$$\Pr_{r_1,r_2,z,i}\left[\begin{array}{l}MC(i,DB_1,\mathrm{st}_1;r_2)=z_i\text{ and}\\R(i,DB_1,\mathrm{st}_1;r_2)\cap\mathrm{BAD}=\emptyset\end{array}\;\middle|\;\widetilde{j}=j^*\right]\geq\frac{99}{100}+\frac{98}{100}-1=\frac{97}{100}.$$

If $R(i,DB_1,\mathrm{st}_1;r_2)\cap\mathrm{BAD}=\emptyset$, we know that the memory checker using $\widetilde{DB}$ and DB_1 are identical (since they are different only in BAD), so we have

$$\Pr_{r_1,r_2,z,i}\left[\begin{array}{l}MC\left(i,\widetilde{DB},\mathrm{st}_1;r_2\right)=z_i\text{ and}\\R(i,DB_1,\mathrm{st}_1;r_2)\cap\mathrm{BAD}=\emptyset\end{array}\;\middle|\;\widetilde{j}=j^*\right]\geq\frac{97}{100},$$

which, by dropping the conjunction with the second event, implies

$$\Pr_{r_1,r_2,z,i}\left[MC\left(i,\widetilde{DB},\mathrm{st}_1;r_2\right)=z_i\;\middle|\;\widetilde{j}=j^*\right]\geq\frac{97}{100}.$$

Then, by an averaging argument[8],

$$\Pr_{r_1,z}\left[\Pr_{i,r_2}\left[MC\left(i,\widetilde{DB},\mathrm{st}_1;r_2\right)=z_i\;\middle|\;r_1,z,\widetilde{j}=j^*\right]>\frac{9}{10}\;\middle|\;\widetilde{j}=j^*\right]\geq\frac{2}{3}.$$

Let G denote these "good" values of (r_1,z) (with density at least $2/3$). By another averaging argument, by restricting to a good value of (r_1,z),

$$\Pr_{i\sim[n]}\left[\Pr_{r_2}\left[MC\left(i,\widetilde{DB},\mathrm{st}_1;r_2\right)=z_i\mid(r_1,z)\in G,i,\widetilde{j}=j^*\right]>\frac{1}{2}\;\middle|\;(r_1,z)\in G,\widetilde{j}=j^*\right]\geq\frac{8}{10}.$$

Since $\tilde{z}_i=\mathrm{BitMajority}\left(MC\left(i,\widetilde{DB},\mathrm{st}_1;r\right)_{r\in\{0,1\}^t}\right)$, this inner event would correspond to success for Bob, so

$$\Pr_{i\sim[n]}\left[\tilde{z}_i=z_i\;\middle|\;(r_1,z)\in G,\widetilde{j}=j^*\right]\geq\frac{8}{10}.$$

Let $I\subseteq[n]$ denote these "good" values of i, where we have $|I|\geq 8n/10$. Assuming $\widetilde{j}=j^*$ and $(r_1,z)\in G$, we notice that this inner probability event corresponds to whether Bob decodes correctly on read i. Assuming $\widetilde{j}=j^*$ and $(r_1,z)\in G$, we therefore know that $\tilde{z}_i=z_i$ is the correct value for all $i\in I$, so Bob can recover a constant fraction of the memory checker's logical bits. That is, there exists $I\subseteq[n]$ with $|I|\geq 8n/10$ such that

$$\left(\widetilde{j}=j^*\wedge(r_1,z)\in G\right)\implies\forall i\in I,\tilde{z}_i=z_i.$$

Now, we argue that with good probability, for all $i\in[n]$, $\tilde{z}_i\in\{z_i,\perp\}$. More precisely:

Claim 3. *There exists a "good" set G' satisfying $\Pr_{r_1,z}[(r_1,z)\in G']\geq 2/3$ and*

$$\left(\widetilde{j}=j^*\wedge(r_1,z)\in G'\right)\implies\forall i\in[n],\tilde{z}_i\in\{z_i,\perp\}.$$

That is, assuming $\widetilde{j}=j^*$ and $(r_1,z)\in G'$, then Bob does not decode "incorrectly" for any $i\in[n]$. We defer the proof of the claim to the full version, but very briefly, we use soundness of the memory checker against the computationally efficient adversary described in Figure 3.

In summary, assuming throughout that $\widetilde{j}=j^*$ and $(r_1,z)\in G\cap G'$, none of $\tilde{z}_i$ will be the incorrect bit, and moreover, for $8n/10$ values of $i\in[n]$, $\tilde{z}_i$ will be the correct bit. In particular, Bob can use the map π to pass these values to x, where Bob now only needs to learn which bits of the remaining at most $n-8n/10=n/5$ placeholder values of x are 1.

We now specify Alice's auxiliary string. Alice can run Bob's whole strategy and deduce where Bob will put placeholder values. Alice will simply send some compressed representation of these placeholder values. Explicitly, Alice must send at most $n/5$ bits of z in the worst case. This will be at most

$$\left\lceil\log\binom{n/5}{j}\right\rceil$$

[8]Here and later, we use the averaging argument that for all $\epsilon,\epsilon_1,\epsilon_2\in(0,1)$ such that $\epsilon_1\cdot\epsilon_2\geq\epsilon$, if $\Pr_{X,Y}[f(X,Y)=1]\geq 1-\epsilon$, then $\Pr_X[\Pr_Y[f(X,Y)=1|X]>1-\epsilon_1]\geq 1-\epsilon_2$.

- For the first $n + n/10$ queries (i.e., for the logical writes):
  - Behave honestly, and record all physical queries made by the memory checker.
- Let $DB_1 \in (\{0,1\}^w \cup \{\perp\})^m$ denote the updated (honest) database after the $n + n/10$ queries.
- For query $n + n/10 + 1$ (i.e., for the logical read):
  - Sample $j^* \sim [c]$ uniformly at random, and let $k := k_{j^*}$.
  - Let $DB_0 \in (\{0,1\}^w \cup \{\perp\})^m$ be the database rewound to the end of the first $n + n/10 - k$ logical queries.
  - Generate $\widehat{DB} \in (\{0,1\}^w \cup \{\perp\})^m$ so that for all $v \in [m]$, $\widehat{DB}[v] = DB_0[v]$ with probability $1/2$ and $\widehat{DB}[v] = DB_1[v]$ with probability $1/2$, independently over all $v \in [m]$.[a]
  - Perform all physical queries from the memory checker with respect to $\widehat{DB}$.

[a] Even if $m = n^{\omega(1)}$, the adversary can lazily generate $DB_0, DB_1, \widehat{DB}$ on the fly by storing all physical queries from the memory checker.

Figure 3: Description of Memory Checking Adversary (for Soundness).

bits for some $j \le k$, where j corresponds to the number of 1s in the $n/5$ placeholder bits. Since $k \le n/100$, this quantity is maximized when $j = k$. As a result, we have the bound

$$|\mathsf{aux}| \le \left\lceil \log \binom{n/5}{k} \right\rceil.$$

Moreover, notice that by definition of G and G' and the union bound, the probability that $\widetilde{j} = j^*$ and $(r_1, z) \in G \cap G'$ is at least $1/c \cdot 1/3 = 1/(300q_r)$. Therefore, the success probability of this protocol is at least $1/(300q_r)$.

We are finally ready to invoke a compression lemma (see details in the full version). By the information that Alice is communicating to Bob, we have

$$\log\left(\prod_{j\in[c], j\neq j^*} \binom{9n/10 + k_j}{k_j}\right) + p + |\mathsf{aux}| + |H|(w + \log m + 2) + 1$$
$$\ge \log\left(\prod_{j\in[c]} \binom{9n/10 + k_j}{k_j}\right) - \log(300q_r).$$

This simplifies to

$$p + |\mathsf{aux}| + |H|(w + \log m + 2) \ge \log \binom{9n/10 + k}{k} - \log(300q_r) - 1,$$

which implies, by using the standard bounds on Binomial coefficients $(n/k)^k \le \binom{n}{k} \le (en/k)^k$, that

$$p \ge \log \binom{9n/10}{k} - \log \binom{n/5}{k} - |H|(w + \log m + 2) - \log(300q_r) - 2$$
$$\ge k \log(45/(10e)) - |H|(w + \log m + 2) - \log(300q_r) - 2.$$

By further simplifying, we get that

$$p \ge \frac{k}{2} - |H|(w + \log m + 2) - \log(300q_r) - 2$$
$$\ge \frac{1}{200q_r q_w \delta_{j^*}} - |H|(w + \log m + 2) - \log(300q_r) - \frac{5}{2}$$
$$= \frac{1}{200q_r q_w}\left(\frac{1}{\delta_{j^*}} - |H| \cdot \underbrace{200q_r q_w(w + \log m + 2)}_{\gamma}\right) - \log(300q_r) - \frac{5}{2}.$$

Finally, using the guarantee from Lemma 2, we get that

$$p > \frac{1}{200q_r q_w} \cdot \frac{n}{(400q_r q_w(w + \log m + 2))^c} - \log(300q_r) - \frac{5}{2}$$
$$\ge \frac{n}{(400q_r q_w(w + \log m + 2))^{100q_r+1}} - \log(q_r) - 11,$$
$$\ge \frac{n}{(600q_r q_w(w + \log m))^{202q_r}} - \log(q_r) - 11,$$

as desired.

ACKNOWLEDGMENTS

We thank Moni Naor and Omri Weinstein for very useful discussions. For the third author, research was partially done at NTT Research. His research is further supported in part by DARPA under Agreement No. HR00112020023, NSF CNS-2154149, NSF DGE-2141064, and a Simons Investigator award.

REFERENCES

[1] Gilad Asharov, Ilan Komargodski, Wei-Kai Lin, Kartik Nayak, Enoch Peserico, and Elaine Shi. 2023. OptORAMa: Optimal Oblivious RAM. *J. ACM* 70, 1 (2023), 4:1–4:70. https://doi.org/10.1145/3566049 4

[2] Gilad Asharov, Ilan Komargodski, Wei-Kai Lin, and Elaine Shi. 2023. Oblivious RAM with Worst-Case Logarithmic Overhead. *J. Cryptol.* 36, 2 (2023), 7. https://doi.org/10.1007/S00145-023-09447-5 4

[3] Giuseppe Ateniese, Randal C. Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary N. J. Peterson, and Dawn Xiaodong Song. 2007. Provable data possession at untrusted stores. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*, Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson (Eds.). ACM, 598–609. https://doi.org/10.1145/1315245.1315318 1

[4] Manuel Blum, William S. Evans, Peter Gemmell, Sampath Kannan, and Moni Naor. 1994. Checking the Correctness of Memories. *Algorithmica* 12, 2/3 (1994), 225–244. https://doi.org/10.1007/BF01185212 1, 3, 4

[5] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. 2021. Halo Infinite: Proof-Carrying Data from Additive Polynomial Commitments. In *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 12825)*, Tal Malkin and Chris Peikert (Eds.). Springer, 649–680. https://doi.org/10.1007/978-3-030-84242-0_23 1

[6] Jonathan Bootle, Alessandro Chiesa, Yuncong Hu, and Michele Orrù. 2022. Gemini: Elastic SNARKs for Diverse Environments. In *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 13276)*, Orr Dunkelman and Stefan Dziembowski (Eds.). Springer, 427–457. https://doi.org/10.1007/978-3-031-07085-3_15 1

[7] Elette Boyle and Moni Naor. 2016. Is There an Oblivious RAM Lower Bound?. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, Madhu Sudan (Ed.). ACM, 357–368. https://doi.org/10.1145/2840728.2840761 4

[8] Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi Vesely. 2021. Proofs for Inner Pairing Products and Applications. In *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 13092)*, Mehdi Tibouchi and

Huaxiong Wang (Eds.). Springer, 65–97. https://doi.org/10.1007/978-3-030-92078-4_3 1

[9] David Cash, Andrew Drucker, and Alexander Hoover. 2020. A Lower Bound for One-Round Oblivious RAM. In *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 12550)*, Rafael Pass and Krzysztof Pietrzak (Eds.). Springer, 457–485. https://doi.org/10.1007/978-3-030-64375-1_16 2, 4

[10] David Cash, Alptekin Küpçü, and Daniel Wichs. 2017. Dynamic Proofs of Retrievability Via Oblivious RAM. *J. Cryptol.* 30, 1 (2017), 22–57. https://doi.org/10.1007/S00145-015-9216-2 1

[11] Dwaine E. Clarke, G. Edward Suh, Blaise Gassend, Ajay Sudan, Marten van Dijk, and Srinivas Devadas. 2005. Towards Constant Bandwidth Overhead Integrity Checking of Untrusted Data. In *2005 IEEE Symposium on Security and Privacy (S&P 2005), 8-11 May 2005, Oakland, CA, USA*. IEEE Computer Society, 139–153. https://doi.org/10.1109/SP.2005.24 1

[12] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptol. ePrint Arch.* (2016), 86. http://eprint.iacr.org/2016/086 1

[13] Cynthia Dwork, Moni Naor, Guy N. Rothblum, and Vinod Vaikuntanathan. 2009. How Efficient Can Memory Checking Be?. In *Theory of Cryptography, 6th Theory of Cryptography Conference, TCC 2009, San Francisco, CA, USA, March 15-17, 2009. Proceedings (Lecture Notes in Computer Science, Vol. 5444)*, Omer Reingold (Ed.). Springer, 503–520. https://doi.org/10.1007/978-3-642-00457-5_30 2, 3, 4, 5

[14] Oded Goldreich. 1987. Towards a Theory of Software Protection and Simulation by Oblivious RAMs. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, Alfred V. Aho (Ed.). ACM, 182–194. https://doi.org/10.1145/28395.28416 3

[15] Oded Goldreich. 2023. On the Lower Bound on the Length of Relaxed Locally Decodable Codes. *Electron. Colloquium Comput. Complex.* TR23-064 (2023). ECCC:TR23-064 https://eccc.weizmann.ac.il/report/2023/064 6, 7

[16] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. 1986. How to construct random functions. *J. ACM* 33, 4 (1986), 792–807. https://doi.org/10.1145/6490.6503 7

[17] Oded Goldreich and Rafail Ostrovsky. 1996. Software Protection and Simulation on Oblivious RAMs. *J. ACM* 43, 3 (1996), 431–473. https://doi.org/10.1145/233551.233553 2, 3, 4

[18] William Eric Hall and Charanjit S. Jutla. 2005. Parallelizable Authentication Trees. In *Selected Areas in Cryptography, 12th International Workshop, SAC 2005, Kingston, ON, Canada, August 11-12, 2005, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 3897)*, Bart Preneel and Stafford E. Tavares (Eds.). Springer, 95–109. https://doi.org/10.1007/11693383_7 1

[19] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. 1999. A Pseudorandom Generator from any One-way Function. *SIAM J. Comput.* 28, 4 (1999), 1364–1396. https://doi.org/10.1137/S0097539793244708 7

[20] Ari Juels and Burton S. Kaliski Jr. 2007. Pors: proofs of retrievability for large files. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*, Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson (Eds.). ACM, 584–597. https://doi.org/10.1145/1315245.1315317 1

[21] Ilan Komargodski and Wei-Kai Lin. 2021. A Logarithmic Lower Bound for Oblivious RAM (for All Parameters). In *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part IV (Lecture Notes in Computer Science, Vol. 12828)*, Tal Malkin and Chris Peikert (Eds.). Springer, 579–609. https://doi.org/10.1007/978-3-030-84259-8_20 4

[22] Kasper Green Larsen and Jesper Buus Nielsen. 2018. Yes, There is an Oblivious RAM Lower Bound!. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 10992)*, Hovav Shacham and Alexandra Boldyreva (Eds.). Springer, 523–542. https://doi.org/10.1007/978-3-319-96881-0_18 4

[23] Jacob R. Lorch, Bryan Parno, James W. Mickens, Mariana Raykova, and Joshua Schiffman. 2013. Shroud: ensuring private access to large-scale data in the data center. In *Proceedings of the 11th USENIX conference on File and Storage Technologies, FAST 2013, San Jose, CA, USA, February 12-15, 2013*, Keith A. Smith and Yuanyuan Zhou (Eds.). USENIX, 199–214. https://www.usenix.org/conference/fast13/technical-sessions/presentation/lorch 4

[24] Surya Mathialagan. 2023. Memory Checking for Parallel RAMs. In *Theory of Cryptography - 21st International Conference, TCC 2023, Taipei, Taiwan, November 29 - December 2, 2023, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 14370)*, Guy N. Rothblum and Hoeteck Wee (Eds.). Springer, 436–464. https://doi.org/10.1007/978-3-031-48618-0_15 3

[25] Surya Mathialagan and Neekon Vafa. 2023. MacORAMa: Optimal Oblivious RAM with Integrity. In *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part IV (Lecture Notes in Computer Science, Vol. 14084)*, Helena Handschuh and Anna Lysyanskaya (Eds.). Springer, 95–127. https://doi.org/10.1007/978-3-031-38551-3_4 4

[26] Moni Naor and Guy N. Rothblum. 2009. The complexity of online memory checking. *J. ACM* 56, 1 (2009), 2:1–2:46. https://doi.org/10.1145/1462153.1462155 2, 7

[27] Noam Nisan and Avi Wigderson. 1993. Rounds in Communication Complexity Revisited. *SIAM J. Comput.* 22, 1 (1993), 211–219. https://doi.org/10.1137/0222016 2

[28] Alina Oprea and Michael K. Reiter. 2007. Integrity Checking in Cryptographic File Systems with Constant Trusted Storage. In *Proceedings of the 16th USENIX Security Symposium, Boston, MA, USA, August 6-10, 2007*, Niels Provos (Ed.). USENIX Association. https://www.usenix.org/conference/16th-usenix-security-symposium/integrity-checking-cryptographic-file-systems-constant 1

[29] Rafail Ostrovsky. 1990. Efficient Computation on Oblivious RAMs. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, Harriet Ortiz (Ed.). ACM, 514–523. https://doi.org/10.1145/100216.100289 3

[30] Alex Ozdemir, Riad S. Wahby, Barry Whitehat, and Dan Boneh. 2020. Scaling Verifiable Computation Using Efficient Set Accumulators. In *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, Srdjan Capkun and Franziska Roesner (Eds.). USENIX Association, 2075–2092. https://www.usenix.org/conference/usenixsecurity20/presentation/ozdemir 1

[31] Charalampos Papamanthou and Roberto Tamassia. 2011. Optimal and Parallel Online Memory Checking. *Cryptology ePrint Archive* (2011). 2, 3

[32] Sarvar Patel, Giuseppe Persiano, Mariana Raykova, and Kevin Yeo. 2018. PanORAMa: Oblivious RAM with Logarithmic Overhead. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, Mikkel Thorup (Ed.). IEEE Computer Society, 871–882. https://doi.org/10.1109/FOCS.2018.00087 4

[33] Ling Ren, Christopher W. Fletcher, Xiangyao Yu, Marten van Dijk, and Srinivas Devadas. 2013. Integrity verification for path Oblivious-RAM. In *IEEE High Performance Extreme Computing Conference, HPEC 2013, Waltham, MA, USA, September 10-12, 2013*. IEEE, 1–6. https://doi.org/10.1109/HPEC.2013.6670339 4

[34] Srinath T. V. Setty. 2020. Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup. In *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 12172)*, Daniele Micciancio and Thomas Ristenpart (Eds.). Springer, 704–737. https://doi.org/10.1007/978-3-030-56877-1_25 1

[35] Hovav Shacham and Brent Waters. 2013. Compact Proofs of Retrievability. *J. Cryptol.* 26, 3 (2013), 442–483. https://doi.org/10.1007/S00145-012-9129-2 1

[36] Emil Stefanov, Marten van Dijk, Elaine Shi, T.-H. Hubert Chan, Christopher W. Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. 2018. Path ORAM: An Extremely Simple Oblivious RAM Protocol. *J. ACM* 65, 4 (2018), 18:1–18:26. https://doi.org/10.1145/3177872 4

[37] Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. 2018. Doubly-Efficient zkSNARKs Without Trusted Setup. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*. IEEE Computer Society, 926–943. https://doi.org/10.1109/SP.2018.00060 1

[38] Weijie Wang, Yujie Lu, Charalampos Papamanthou, and Fan Zhang. 2023. The Locality of Memory Checking. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*, Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda (Eds.). ACM, 1820–1834. https://doi.org/10.1145/3576915.3623195 3

[39] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. 2019. Libra: Succinct Zero-Knowledge Proofs with Optimal Prover Computation. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 11694)*, Alexandra Boldyreva and Daniele Micciancio (Eds.). Springer, 733–764. https://doi.org/10.1007/978-3-030-26954-8_24 1

[40] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. 2020. Transparent Polynomial Delegation and Its Applications to Zero Knowledge Proof. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*. IEEE, 859–876. https://doi.org/10.1109/SP40000.2020.00052 1

Received 12-NOV-2023; accepted 2024-02-11