# The Complexity of Memory Checking with Covert Security

Elette Boyle* Ilan Komargodski† Neekon Vafa‡

**Abstract**

A memory checker is an algorithmic tool used to certify the integrity of a database maintained on a remote, unreliable, computationally bounded server. Concretely, it allows a user to 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. A recent result due to Boyle, Komargodski, and Vafa (BKV, STOC '24) showed a tradeoff between the size of the local storage and the number of queries the memory checker makes to the server upon every logical instruction. Specifically, they show that every non-trivial memory checker construction with inverse-polynomial soundness and local storage at most $n^{1-\epsilon}$ must make $\Omega(\log n / \log \log n)$ queries, and this is tight up to constant factors given known constructions. However, an intriguing question is whether natural relaxations of the security guarantee could allow for more efficient constructions.

We consider and adapt the notion of *covert* security to the memory checking context, wherein the adversary can effectively cheat while taking the risk of being caught with constant probability. Notably, BKV's lower bound does not apply in this setting.

We close this gap and prove that $\Omega(\log n / \log \log n)$ overhead is unavoidable even in the covert security setting. Our lower bound applies to any memory checker construction, including ones that use randomness and adaptivity and ones that rely on cryptographic assumptions and/or the random oracle model, as long as they satisfy a natural "read-only reads" property. This property requires a memory checker not to modify contents of the database or local storage in the execution of a logical read instruction.

*Reichman University and NTT Research. Email: eboyle@alum.mit.edu.
†Hebrew University and NTT Research. Email: ilank@cs.huji.ac.il.
‡MIT. Email: nvafa@mit.edu.

# 1 Introduction

Memory checking, introduced by Blum, Evans, Gemmel, Kannan, and Naor [BEG+94], is a central algorithmic tool used to certify the integrity of a database that is maintained on a remote unreliable server [HJ05, CSG+05, OR07, CD16]. A memory checker uses a small reliable local storage and serves as a proxy between the user and the remote database, allowing verifiable read and write queries. Since there is no way to prevent a rogue server from corrupting the memory altogether, a memory checker has the following best possible soundness guarantee: whatever query is made by the user, either it is executed correctly or the user is notified that the server is 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 $p$), i.e., the size of the reliable local storage in bits, and its query complexity (denoted $q$), the number of physical queries made to the unreliable memory per logical user request. Memory checkers have been studied both in the information-theoretic setting (where the server is computationally unbounded) and in the computational setting (where the server runs in polynomial time). In the information-theoretic setting it is known that memory checkers must be "trivial" in the sense that $p{\cdot}q = \Omega(n)$, where $n$ is the memory size [BEG+94, NR09]. This is true even for a very weak variant of memory checkers where completeness and soundness are both constant. In the computational setting, however, significantly more efficient schemes exist. Specifically, there exist constructions where $q = O(\log n/ \log\log n)$ and $p$ is only proportional to the computational security parameter [BEG+94, PT11, BKV24].

For many years the question of whether the above parameters were optimal was largely open, with only one lower bound ruling out more efficient constructions of restricted form [DNRV09]. Very recently, Boyle, Komargodski, and Vafa [BKV24] proved an unrestricted lower bound, ruling out any construction with $q = o(\log n/ \log\log n)$ queries, no matter the cryptographic assumptions being made and not limiting in any way the behavior of the memory checker. Nevertheless, a drawback of [BKV24]'s result is that it applies only to memory checkers that have a strong soundness guarantee, i.e., inverse polynomial soundness, roughly $1/n^{1+o(1)}$. Standard approaches toward soundness amplification are not known within the setting of memory checkers, and at best would degrade the bound of [BKV24] to a meaningless guarantee.

**Memory checkers with covert security.** We investigate the possibility of memory checkers with weaker soundness guarantees, considering a version of the *covert security* model [AL10] within the context of memory checkers. In the covert security model, the guarantee is that if a malicious server tries to cheat in order to break the integrity of the system, then it will be caught with some constant probability. Unlike the standard security model that does not allow the adversary to cheat at all, with covert security, the adversary can effectively cheat while taking the risk of being caught. In the context of memory checking, covert security corresponds to relaxing the soundness parameter to be constant instead of negligible. Importantly, the lower bound of [BKV24] leaves open the possibility of attaining significantly more efficient memory checkers within this model, ideally even ones that have only constant overhead.

The main justification for studying covert security in our context is that in many practical applications of memory checkers, it makes sense that the server's fear of being caught cheating is enough of a deterrent to prevent it from cheating. For example, a cloud service provider that is being caught cheating risks hurting their reputation and future business, and therefore has no incentive in trying to cheat in the first place. Moreover, in blockchain applications of memory checking,

Figure 1: Summary of known lower bounds on the size of the local storage $p$ of memory checkers (with computational security) for a logical memory of size $n$ with query complexity $q$.

| **Reference** | **Space/Query tradeoff** | **Soundness** | **Limitation of the Scheme** |
|---|---|---|---|
| [DNRV09] | $p \geq n/(\log n)^{O(q)}$ | Constant | Deterministic and non-adaptive |
| [BKV24] | $p \geq n/(\log n)^{O(q)}$ | $1/n^{1+o(1)}$ | None |
| This work | $p \geq n/(\log n)^{O(q)}$ | Constant | Read-only reads |

money staked on a cryptographic proof could be lost if verification fails. In this setting, a constant probability of forfeiting money is enough of a disincentive to prevent malicious activity.

Since the original work that introduced the covert security model [AL10], it has been widely studied in the classical secure computation literature with the aim of designing secure protocols that are much more efficient than the ones known to satisfy the stronger malicious security notion. In particular, covert security has enabled asymptotic efficiency gains in terms of computational overhead and communication complexity (e.g., [AL10, GMS08, HL10]). As memory checking is itself a special case of secure computation, we view this possibility as (a priori) quite plausible in light of efficiency gains in the literature when relaxing other forms of secure computation from malicious to covert security. More than that, since logarithmic factors in $n$ in query complexity (as achieved by standard memory checkers) can be prohibitively expensive in practice, we believe that the question of relaxing memory checkers to covert security to bypass this logarithmic blowup is a significant one.

**Our main result.** We prove that relaxing the security guarantee of memory checkers to covert essentially *does not allow for asymptotically better constructions*, separating memory checking from other secure computation functionalities. Our lower bound applies to all memory checker constructions satisfying a very natural "read-only reads" property, which says that the memory checker does not make any writes to the database or modify the local storage while handling a read instruction. Notably, our lower bound applies to constructions that utilize randomness and adaptivity, and also to constructions that rely on cryptographic assumptions and/or the random oracle model.

**Theorem 1** (Main result stated informally; see Theorem 3)**.** *Every memory checker with read-only reads (with computational security) for a logical memory of size $n$ that has query complexity $q$, a local storage of size $p$, completeness $2/3$, and soundness $1/3$, must satisfy $p \geq n/(\log n)^{O(q)}$.*

As a special case, $q = \Omega(\log n/ \log\log n)$ assuming $p = n^{1-\Omega(1)}$, even if completeness and soundness are $2/3$ and $1/3$, respectively.

In fact, our formal theorem (see Theorem 3) is more fine-grained in the sense that we separately consider the read- and write-query complexities and obtain a trade-off curve. Specifically, letting $q_r$ and $q_w$ be the read- and write-query complexities of the memory checker, respectively, we prove that under the same conditions as in Theorem 1, it must be that $p \geq n/(q_r q_w \log n)^{O(q_r)}$. This implies, for example, that as long as $p = n^{1-\Omega(1)}$, if $q_r = O(1)$, then $q_w = n^{\Omega(1)}$ (i.e., a memory checker with constant read-query complexity must have polynomial write-query complexity), even if the memory checker has only constant soundness.

See Figure 1 for a summary of known lower bounds in the computational setting.

**On the read-only reads assumption.** As mentioned, the "read-only reads" property says that the memory checker does not modify contents of the database or local storage in the execution of a logical read query. We maintain that this is a natural state of affairs, which is upheld by all existing constructions of memory checkers from the literature to our knowledge. (In fact, one could argue that the natural definition of a memory checker should require this property, but for generality, we do not define memory checkers this way.)

As an illustration, the classical and optimal construction of Blum et al. [BEG+94] (and its refinement due to Tamassia and Papamanthou [PT11]) arranges the memory content in the leaves of a Merkle tree [Mer89], wherein every internal node essentially "authenticates" the content of its two children. Thus, to read the content of a logical memory cell, the memory checker merely queries the nodes on the path from the root to the associated leaf, verifying consistency of each parent-child nodes in this path. In this construction, it is obvious that reading a (logical) memory cell's content requires only reading from the memory. In fact, while one can devise contrived schemes that do not comply, it is not clear how these (or any method of writing during reads) could intuitively aid in boosting soundness or completeness.

We remark that the need for this assumption comes for a technical reason in the proof of Theorem 1 on which we elaborate in Section 2. We believe that the statement of Theorem 1 should hold even without this assumption; proving this, however, is left as an open problem.

**Implications of our result.** Memory checkers have been extended and generalized in various ways and have even been used in numerous applications (including provable data possession and retrievability systems [ABC+07, JJ07, OR07, SW13, CKW17], and cryptographic verifiable computation systems [WTS+18, XZZ+19, Set20, ZXZS20, OWWB20, BMM+21, BDFG21, BCHO22, AST24, STW24, ST25], to name a few). Therefore, naturally, our lower bound directly implies a lower bound for many of these extensions. We mention a few direct implications for concreteness.

Mathialagan [Mat23] constructed memory checkers for *Parallel* RAM (PRAM) machines that has $O(\log N)$ query blowup and $O(\log N)$ depth blowup, relying on the existence of one-way functions and achieving negligible soundness. Using our lower bound, one can conclude that in terms of query blowup, their scheme is optimal (among schemes with read-only reads), even if only constant soundness is targeted. Wang, Lu, Papamanthou, and Zhang [WLPZ23] studied the *locality* of memory checkers (i.e., the number of non-contiguous memory regions a checker must query to verifiably answer a query). Using our lower bound, we conclude that $\Omega(\log n/\log\log n)$ locality is necessary for any possible scheme (with read-only reads), even if only constant soundness is required.

Lastly, we mention a connection between memory checkers and maintainable vector commitments [SCP+22, WUP23]. A maintainable vector commitment (MVC) is, in particular, a vector commitment (VC), which allows a prover to compute a succinct digest of a vector along with proofs for each position. A verifier who has the digest can later verify a proof that a reported value in a given position is correct (a Merkle tree is the canonical implementation of a VC). The maintainable aspect of a MVC allows changing a particular entry in the vector in sub-linear time (i.e., by updating only a fraction of the proofs). Thus, in spirit, a MVC implies a memory checker,[1] and so memory checker lower bounds (ours and of [BKV24]) directly imply analogous lower bounds for MVCs.

[1]This implication is likely known by experts, but we have not seen it mentioned anywhere. We additionally note that we are not aware of an existing "vanilla" definition of MVC since they are defined with additional useful properties, like aggregation. Thus, we avoid coming up with such a definition merely for the sake of formalizing an implication to memory checkers.

**Amplification via repetition.** A natural attempt toward addressing weak soundness guarantees would be to generically amplify soundness of the original scheme via some form of repetition (and invoke the result of [BKV24]). Due to the online nature of memory checkers, a method of repetition that seems to make sense is to maintain multiple independent instances of the scheme, executing every logical instruction independently and sequentially across all instances. Concretely, to execute two logical operations $\mathsf{op}_1, \mathsf{op}_2$, we first execute $\mathsf{op}_1$ across all instances sequentially and when this is over we execute $\mathsf{op}_2$ across all instances sequentially. For a read instruction, we will use the majority, i.e., the value that appears as the response from most instances.

This approach fails due to two main reasons. First, it is not clear that this form of repetition reduces soundness at all; indeed, this form of repetition is essentially parallel across logical queries, and so runs into the issues raised by Bellare, Impagliazzo, and Naor [BIN97], showing that parallel repetition does not necessarily amplify soundness in a computational setting. Second, even if parallel repetition would amplify soundness, we need to amplify from constant to inverse polynomial. In the best-case scenario, this would require having $\Omega(\log n)$ independent instances running in parallel, which would cause a logarithmic blowup in both space and query complexity, enough to destroy this regime of bounds.

## 1.1 Related Work

**Covert deterministic and non-adaptive memory checkers.** Before the work of [BKV24], Dwork, Naor, Rothblum, and Vaikuntanathan [DNRV09] showed that logarithmic query complexity is inherent for a *restricted class* of memory checkers. Specifically, their lower bound applies to 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*. Their lower bound does rule out even covert secure constructions (of deterministic and non-adaptive form).

Formally, the assumptions of being "deterministic and non-adaptive" (as in [DNRV09]) and having "read-only reads" are incomparable, as far as we can tell. However, we view the "read-only reads" assumption as comparatively more mild. Indeed, determinism and non-adaptivity are features that are known to improve complexity in various contexts. For instance, binary search is an adaptive procedure, and there is no analogous non-adaptive variant with similar complexity. Also, examples exist in which $r$-round communication protocols are exponentially more efficient than $(r-1)$-round communication protocols [NW93]. Lastly, data structures that use randomness give significant efficiency gains (Cuckoo hashing, for instance). On the other hand, we do not see a scenario where violating the "read-only reads" assumption would allow for efficiency gains.

**Online vs. offline memory checkers.** In this paper, we consider *online* memory checkers, i.e., memory checkers that need to report either a correct value or an error (but never be wrong) after every logical instruction, before seeing the next instruction. A natural variant is called *offline* memory checkers, wherein it is required to report whether an error has occurred at the end of a batch of instructions. Blum et al. [BEG+94] constructed offline memory checkers that achieve amortized $O(1)$ overhead per instruction (see also [DNRV09, Section 5]), and so our lower bounds obviously do not apply (in fact, these constructions are even statistically secure).

## 2 Technical Overview

We begin by recapping the previous lower bound of Boyle, Komargodski, and Vafa [BKV24] (henceforth referred to as BKV) and describing what fails in the case of constant soundness error.

At a high level, the proof idea is a compression argument: Alice can write to a random subset of the logical indices of a memory checker, and by sending the resulting local storage to Bob (along with short extra helper information we describe below), Bob can recover a large fraction of the entropy of the random subset. This places a lower bound on the size of the local storage.

In more detail, suppose the memory checker uses physical words (i.e., blocks) of size $w$ bits. Let $m$ denote the size (i.e., number of physical words) on the remote server. To set up the compression argument, suppose that Alice wishes to communicate a uniformly random string $x \in \{0,1\}^n$ of fixed Hamming weight $k$ to Bob (where $k$ is a parameter set later). Alice and Bob can initialize an empty (all "0"s) memory checker, which results in a public database $DB_0 \in (\{0,1\}^w)^m$ and local storage $\mathsf{st}_0 \in \{0,1\}^p$. Then, Alice can write "1" to all logical indices in support$(x) := \{i \in [n] : x_i = 1\}$, resulting in updated public database $DB_1$ and $\mathsf{st}_1$. After this, Alice can send Bob $\mathsf{st}_1$.

Bob can attempt the following decoding strategy to recover $x$. Using the outdated $DB_0$ but fresh $\mathsf{st}_1$, Bob can run the memory checker and perform a read instruction to all $i \in [n]$, resulting in values in $\{0,1,\perp\}$, where $\perp$ indicates failure. The hope is to apply completeness and soundness of the memory checker (respectively) to argue that:

1. Bob will often recover many binary (non-$\perp$) values, *by completeness*, and

2. Bob's binary values will not be incorrect, *by soundness*.

If these two conditions are satisfied, then Bob will recover lots of the logical memory (and hence lots of $x$), which allows BKV to give a lower bound on $p$ by a standard communication argument.

To address Item 1, BKV [BKV24] introduce a tri-partition of the public database of the form $[m] = H \sqcup M \sqcup L$, denoting heavy, medium, and light physical locations, respectively. Here "heavy" and "light" are with respect to how frequently logical reads to a random $i \in [n]$ touch a given physical location. BKV have Alice additionally send $H \subseteq [m]$ and $DB_1|_H \in (\{0,1\}^w)^{|H|}$ (i.e., the updated public database locations on $H$) to Bob, allowing Bob to compute a "hybrid" physical database $\widetilde{DB}$ defined by

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

BKV now argue that for some well-chosen $H$ and $k$, Bob can recover at least $\approx 4n/5$ binary values for the logical memory by running the memory checker with $\mathsf{st}_1$ (still) and $\widetilde{DB}$.

Now, even if these values for the logical memory are all "0" and do not intersect support$(x)$, if we assume these binary values are all correct, this narrows a superset of support$(x)$ from size $n$ to $n/5$, resulting in

$$\log \binom{n}{k} - \log \binom{n/5}{k} = \Omega(k)$$

bits of information being sent from Alice to Bob (for, say, $k \leq n/100$). Loosely speaking, BKV's tri-partition lemma is set up so that these $\Omega(k)$ bits of information sent from Alice to Bob sufficiently dominate $|H|$, consequently giving a lower bound on $p = |\mathsf{st}_1|$.

The crucial part for us is addressing Item 2. Note that in the compression argument, Bob only learns information from Alice if Bob does not learn any *incorrect* binary values. For example, for all we know, it is possible that using $\widetilde{DB}$ instead of $DB_1$ could cause the memory checker to always return "0", in which case Bob does not learn anything. This is where soundness comes in. To apply soundness, we need to make sure that Bob's decoding strategy coincides with an *efficient* (i.e., probabilistic polynomial time) attack on the memory checker.

The critical question: Can $k$ and $H$ (used to compute $\widetilde{DB}$) be computed efficiently? Let us discuss these separately.

- **Challenge 1: Computing $k$.** BKV's partition argument shows that it is always true that a "correct" $k$ exists and moreover it can be set to one of $O(q)$ possibilities. BKV deal with this in the soundness argument by simply having the adversary "guess" the correct value of $k$, resulting in a $O(q)$ factor loss in soundness error.

- **Challenge 2: Computing $H$.** The heavy set $H$ is only defined existentially as part of the tri-parition lemma. In BKV [BKV24], this is dealt with by having the adversary essentially brute-force "guess" $H$, as well. Namely, for each of the $q$ physical locations read during the logical read, the adversary guesses at random whether the corresponding location is heavy: i.e., whether $\widetilde{DB}$ agrees with $DB_0$ or $DB_1$. This results in a further $2^q$ factor loss in soundness error.

What is common to both of the above challenges of computing $k$ and $H$ is that they could *adaptively change throughout the writes of the memory checker.* That is, the database $DB$ and local storage $\mathsf{st}$ change with each logical write, making the "correct" corresponding values of $k$ and $H$ change as well. More than that, the value of $k$ and the partition to $H, M$, and $L$ in BKV [BKV24, Lemma 3] are given purely in the analysis and they depend on the exact probability distribution of the memory checker, which the adversary need not have access to. In fact, this distribution could depend on the private randomness and/or private local storage of the memory checker.

## 2.1 Our Idea: Learning an Approximate Partition

As mentioned, the value of $k$ and the partition to $H, M, L$ could depend on the private local storage of the memory checker, and so it is infeasible to efficiently compute them exactly; but, there is hope we could *approximate* them. This is where we diverge from BKV. The idea of approximation leads to the following two questions: (1) What is a reasonable notion of approximation for a set $H$? and (2) given whatever approximation of $H$ we can come up with, does it suffice for the rest of argument to go through? We elaborate on these points next.

1. We begin by showing there is indeed a suitable notion of an *approximation* of a heavy set $H$ and parameter $k$ that is sufficient for our lower bound argument. Concretely, the notion of approximation we use allows some of the elements that should have been placed in $M$ to be placed in $H$ or $L$ (and vice versa), while still preserving a sufficiently large gap between $L$ and $H$. (See Theorem 2 for a precise statement.)

2. We show that if one has enough samples from the memory checker's logical read behavior, then there is a way to efficiently *learn* such an approximation of a heavy set $H$ and parameter $k$, simultaneously. That is, with enough samples from the memory checker, generating our approximation is computationally efficient.

3. To generate these samples from the memory checker's read behavior, the adversary can issue additional reads before the attack, even if we assume the local storage of the memory checker is *secret* from the adversary. That is, by issuing logical reads, the memory checker does not need to read the local storage of the memory checker to generate samples needed to run the efficient algorithm from Theorem 2.

A careful analysis of Item 2 is the main technical portion of this paper, which we give in Section 4 (Theorem 2) and outline below. Note that Items 1 and 2 force us to change what Alice sends in the compression argument (Theorem 3), as compared to BKV. (See Figure 6 for a description of the adversary used when invoking soundness in the proof of the main theorem.) Further note that there are additional technical issues that we gloss over here that we defer to the proof of Theorem 3.

**Efficiently learning an approximate heavy set.** We first recall how the proof of the partition lemma in BKV [BKV24, Lemma 3] works, which is itself inspired by Goldreich [Gol23, Claim 2.6]. Roughly speaking, [BKV24, Lemma 3] partitions elements of $[m]$ into a "histogram" with $c$ geometrically increasing buckets, depending on the probability of each given $j \in [m]$. By averaging, one of these buckets must have total probability at most $1/c$. Taking $M$ to be this bucket and $H$ and $L$ to be the other two sides of $M$ on the histogram, we get enough of a multiplicative "gap" between $H$ and $L$ so that Item 4 of Theorem 2 (or rather, its equivalent in [BKV24, Lemma 3]) goes through. This "gap" is what enables the final lower bound on $p = |\mathsf{st}_1|$.

However, in a computational setting, one cannot efficiently compute the precise probabilities of querying each $j \in [m]$, since one only has sample access to the distribution. As a result, we can only work with *approximations* to the probabilities of each $j \in [m]$. Instead of directly trying to compute these $c$ histogram buckets, we can work with "cushioned" buckets $\{\widehat{C}_i\}_{i\in[c]}$ with a wider probability range that *cover* $[m]$ but do not *partition* $[m]$; that is, we allow overlap between buckets. The hope is that each cushioned bucket $\widehat{C}_i$, defined in terms of the sample probabilities that we can efficiently compute, contains $B_i$, where $\{B_i\}_{i\in[c]}$ form a geometric partition defined in terms of the *true* probabilities of $\mathcal{D}$. For appropriately chosen probability ranges for the buckets, this can be shown to be true using Chernoff bounds (see Claim 1 in Theorem 2). Finally, we argue that setting $M$ to (essentially) be the $\widehat{C}_i$ with smallest total probability is good enough, even though $\{\widehat{C}_i\}_{i\in[c]}$ do not form a partition. See Figure 2 and the pseudocode in Figure 3 for more details.

**Read-only reads.** We now explain where the "read-only reads" assumption is needed. Recall that this assumption states that logical read instructions only cause physical reads (i.e., not physical writes) and do not change the local storage. As mentioned, this property is attained by all non-contrived existing memory checker constructions. We now describe why we need this assumption.

In short, there are two possible routes for our lower bound argument to go through. For Bob to recover the contents at logical index $i$ given $\mathsf{st}_1$ and $\widetilde{DB}$, either (a) Bob rewinds to the previous local storage and database between each logical read, or (b) Bob sequentially reads all $i \in [n]$. For route (a), it is not clear how constant soundness can be sufficient, since, for example, the memory checker can randomly choose an $(n/10)$-sized subset to be incorrect on for that final read. In such a case, we cannot rule out that Bob recovers no information about $x$. (Note that this is the route that BKV [BKV24] take, which explains why they have a factor $n$ loss, on top of the $O(q \cdot 2^q)$ loss.) On the other hand, for route (b), the local storage and $DB_1$ could "update" between each $i \in [n]$. For any reads that query an old, corrupted part of $\widetilde{DB}$, we have no way to invoke completeness of

the next read, as $DB_1$ and $\mathsf{st}_1$ could be updated to junk. Moreover, Alice cannot send the correct freshly updated $\mathsf{st}_1$ and/or $DB_1$ after each logical read, as that would blow up the communication argument. The "read-only reads" assumption makes sure that route (b) is possible; sequential logical reads that Bob does do not impact $\mathsf{st}_1$ nor $DB_1$, even if the logical reads query incorrect contents in $\widetilde{DB}$.

# 3 Preliminaries

For a natural number $n \in \mathbb{N}$, we let $[n] := \{k \in \mathbb{N} : 1 \leq k \leq n\}$. All logarithms with the notation log are in base 2, unless otherwise specified; the notation ln is reserved for the natural logarithm.

For a finite distribution $\mathcal{D}$, we write $X \sim \mathcal{D}$ to denote that the random variable $X$ is distributed according to $\mathcal{D}$. For a finite set $S$, we abuse notation and write $X \sim S$ to denote that the random variable $X$ is distributed uniformly over $S$. For subsets $A_1, \cdots, A_\ell \subseteq U$, we write $U = A_1 \sqcup \cdots \sqcup A_\ell$ to indicate that $\{A_i\}_{i\in[\ell]}$ partition $U$, i.e., $U = \cup_{i\in[\ell]} A_i$ and $A_i \cap A_j = \emptyset$ for all $i, j \in [\ell], i \neq j$. For a string $x \in \{0,1\}^\ell$, we let $\|x\|_0$ denote the Hamming weight of $x$, i.e., $\|x\|_0 = |\{i \in [\ell] : x_i = 1\}|$. We say that an algorithm $\mathcal{A}$ is PPT if it runs in probabilistic polynomial time.

**Compression lemma.** We state the well-known lemma typically used in compression arguments (and used in [BKV24]). The lemma roughly says that any method for encoding a uniformly random $\ell$-bit string with fewer than $\ell$ bits must lose information about the string.

**Lemma 1** (E.g., [DTT10])**.** *Let $S$ be a finite set. Consider the one-way communication problem of sending uniformly random $x \sim S$ from Alice to Bob in the public coin setting where the length of Alice's message to Bob is fixed. If Bob succeeds in outputting $x$ with probability $\delta$ (over $x$ and the public coins), then Alice's (binary) message to Bob must be at least* $\log_2(|S|) - \log_2(1/\delta)$ *bits long.*

## 3.1 Memory Checking

Here, we define the notion of memory checkers that we consider. We directly use many of the same explanations, definitions, etc. as in Boyle et al. [BKV24] throughout this section.

**Random-access machines.** A RAM is an interactive Turing machine that consists of a remote memory and a user. The memory is indexed by the logical address space $[n]$. We refer to each memory word also as a block and we use $w_\ell$ to denote the bit-length of each block. The user can issue read/write instructions to the memory of the form $(\mathsf{op}, \mathsf{addr}, \mathsf{data})$, where $\mathsf{op} \in \{\mathsf{read}, \mathsf{write}\}$, $\mathsf{addr} \in [n]$ and $\mathsf{data} \in \{0,1\}^{w_\ell} \cup \{\bot\}$. If $\mathsf{op} = \mathsf{read}$, then $\mathsf{data} = \bot$ and the returned value is the content of the block located in logical address $\mathsf{addr}$ in the memory. If $\mathsf{op} = \mathsf{write}$, then the memory data in logical address $\mathsf{addr}$ is updated to data. The user can perform arbitrary computation.

**Memory checkers.** A memory checker [BEG+94, NR09, DNRV09], at a high level, is a simulation of a standard RAM so that it has the additional guarantee that the RAM's answers to $\mathsf{read}$ instructions are reliable even if the remote memory is being tampered with. We formalize a memory checker as an additional interactive Turing machine, placed "between" the user and the remote memory (so they no longer interact directly). The memory checker gets logical queries to a size $n$ memory from the user and processes them into a—*possibly adaptively generated and using*

*randomness*—sequence of read/write instructions to a physical memory. Responses are sent back to the memory checker that processes them until it finally sends a response to the user (if needed). We emphasize that the user issues instructions for the *logical* memory but the simulation is done using a *physical* memory that could possibly be bigger. We let $[m]$ be the address space of the physical memory and assume each word is of size $w$ bits. What makes the task of designing a memory checker non-trivial is the fact that we wish to limit the local space of the memory checker, where by "local space" we mean the number of bits $p$ preserved in between user instructions (so it is not a true space complexity parameter, as the memory checker could use an unbounded number of bits in the middle of carrying out a user's query).

Formally, the interface of a memory checker is as follows. We assume that at the start of the system, the local storage of the memory checker is empty $\mathsf{st} = \perp^p$ and the physical memory $DB$ is initialized with all $\perp$s. Furthermore, the memory checker is parametrized by logical and physical memory size and block size $(n, w_\ell)$ and $(m, w)$, respectively, as well as possibly a computational security parameter given in unary. We suppress these for convenience when clear from context.

- $\langle(\mathsf{data}', \mathsf{st}'), DB'\rangle \leftarrow (\mathsf{MemCheckerOp}(\mathsf{st}, \mathsf{op}, \mathsf{addr}, \mathsf{data}) \iff DB)$. This interactive protocol between the memory checker with local storage $\mathsf{st} \in \{0,1\}^p$ and the physical memory $DB \in (\{0,1\}^w \cup \{\perp\})^m$ is executed upon a logical instruction $(\mathsf{op}, \mathsf{addr}, \mathsf{data})$, where $\mathsf{op} \in \{\mathsf{read}, \mathsf{write}\}$, $\mathsf{addr} \in [n]$, and $\mathsf{data} \in \{0,1\}^{w_\ell} \cup \{\perp\}$. The output of this protocol consists of a data entry $\mathsf{data}' \in \{0,1\}^{w_\ell} \cup \{\perp\}$ (where $\mathsf{data}' = \perp$ if $\mathsf{op} = \mathsf{write}$ or if the memory checker detects tampering) and the new local storage of the memory checker $\mathsf{st}'$, where $|\mathsf{st}'| = p$. The updated physical memory, after the interaction ends, is denoted $DB'$.

  The protocol may consist of multiple rounds, where in each round the memory checker issues a "physical" RAM query of the form $(\widehat{\mathsf{op}}, \widehat{\mathsf{addr}}, \widehat{\mathsf{data}})$, where $\widehat{\mathsf{op}} \in \{\mathsf{read}, \mathsf{write}\}$, $\widehat{\mathsf{addr}} \in [m]$, and $\widehat{\mathsf{data}} \in \{0,1\}^w$. For every such query, the physical memory $DB$ responds with a value $\widehat{\mathsf{data}}' \in \{0,1\}^w \cup \{\perp\}$, where $\widehat{\mathsf{data}}' = DB[\widehat{\mathsf{addr}}]$ if $\widehat{\mathsf{op}} = \mathsf{read}$ and $\widehat{\mathsf{data}}' = \perp$ otherwise.

**Read-only reads.** As explained in the introduction, we add the constraint that during logical reads, memory checkers do not make any physical writes to the database or modify the local storage.

**Definition 1** (Read-only reads). *A memory checker is said to have the "read-only reads" property if upon every logical* $\mathsf{read}$ *instruction* $\mathsf{op} = \mathsf{read}$*, it holds that* $\mathsf{st}' = \mathsf{st}$ *and all physical queries have* $\widehat{\mathsf{op}} = \mathsf{read}$.

The main complexity measure of a memory checker is the communication complexity of the protocols, measured by the total number of queries issued by the memory checker per logical instruction. This is made precise in the following definition.

**Definition 2** (Query complexity). *We define the* read query complexity*, denoted $q_r$ (and* write query complexity*, denoted $q_w$) to be the worst-case number of physical queries made by the memory checker for any given logical read (and logical write, respectively). The* query complexity*, denoted $q$, is defined as $q = \max\{q_r, q_w\}$.*

**Remark 1** (Parameters convention). *We shall assume that all of the above procedures/protocols can be implemented by machines that run in polynomial time in $n$, the logical memory size. This implicitly bounds $m \leq 2^{\mathrm{poly}(n)}$, as physical indices need $\log m$ bits to represent. If unspecified,*

*we assume that $m$ is bounded by a polynomial in $n$. We remark that this is (essentially) without loss of generality because memory checkers with a fixed number of supported logical queries, can be generically transformed (via a PRF) into ones where $m \in \mathrm{poly}(n)$; see [BKV24, Appendix B]. Also, unless otherwise specified, we assume $w_\ell, w \leq \mathrm{polylog}(n)$.*

**Completeness and soundness.** Loosely speaking, completeness of a memory checker says that in an honest execution, i.e., without an adversary tampering with the memory, the user should get the "correct" answer. Soundness says that if the adversary is actively trying to tamper with the memory, then the user will either get the "correct" answer or an abort symbol $\perp$ indicating that something went wrong. Importantly, the memory checker should not respond with a wrong value. Below, we formalize these properties.

We first define the ideal memory functionality, i.e., a memory that is executed by a trusted party and can provide with the "correct" value of every memory cell at any point in time. We refer to this functionality as the *logical memory snapshots functionality.*

**Definition 3** (Logical memory snapshot functionality)**.** *A logical memory snapshot functionality is an ideal primitive that gets as input logical instructions and outputs the full state (so called "snapshot") of the logical memory after each instruction. Initially the memory is assumed to be empty, indicated by $\perp$ in each cell. More precisely, the functionality receives (adaptively) a stream of logical instructions of the form* $(\mathsf{op}, \mathsf{addr}, \mathsf{data})$*, where* $\mathsf{op} \in \{\mathsf{read}, \mathsf{write}\}$*,* $\mathsf{addr} \in [n]$*, and* $\mathsf{data} \in \{0,1\}^{w_\ell}$*. Each* read *instruction is ignored. Each* write *instruction is executed, namely, for* $(\mathsf{write}, \mathsf{addr}, \mathsf{data})$ *the value in index* addr *is updated to* data*. After processing each instruction, a snapshot of the memory is generated, denoted* SnapshotMem*.*

*For a sequence of $l$ logical instructions* $\{(\mathsf{op}_i, \mathsf{addr}_i, \mathsf{data}_i)\}_{i\in[l]}$*, fed into the ideal functionality, we have $l$ snapshots* $\mathsf{SnapshotMem}_1, \dots, \mathsf{SnapshotMem}_l$*. The $i$th snapshot satisfies for every* $\mathsf{addr} \in [n]$ *that* $\mathsf{SnapshotMem}_i[\mathsf{addr}] = \mathsf{data} \in \{0,1\}^{w_\ell} \cup \{\perp\}$ *if there exists a $j < i$ such that* $(\mathsf{op}_j, \mathsf{addr}_j, \mathsf{data}_j) = (\mathsf{write}, \mathsf{addr}, \mathsf{data})$ *and there is no $j < j' < i$ with* $(\mathsf{op}_{j'}, \mathsf{addr}_{j'}, \mathsf{data}_{j'}) = (\mathsf{write}, \mathsf{addr}, \mathsf{data}')$ *where* $\mathsf{data}' \neq \mathsf{data}$*.*

Now, we can define completeness. We require that with probability $c(n)$ over the internal randomness of the memory checker, the memory checker's outputs are all correct (i.e., equal to the corresponding value in the $i$th logical memory snapshot). Furthermore, we do not allow the logical queries to be adaptively chosen based on previous physical queries made by the memory checker. We emphasize that this definition of completeness is rather weak, and specifically, it is weaker than some definitions that were formalized in prior works. However, since we prove a lower bound (meaning we rule out constructions satisfying a weak notion of completeness), these weakenings only make our result stronger. Essentially, all known non-contrived constructions achieve perfect completeness ($c(n) = 1$), the strongest definition of completeness.

**Definition 4** ($c$-completeness)**.** *Let $l \in \mathbb{N}$ and consider a sequence of $l$ logical instructions* $I = \{(\mathsf{op}_i, \mathsf{addr}_i, \mathsf{data}_i)\}_{i\in[l]}$*, where* $\mathsf{op} \in \{\mathsf{read}, \mathsf{write}\}$*,* $\mathsf{addr} \in [n]$*, and* $\mathsf{data} \in \{0,1\}^{w_\ell}$*.*

*We say that a memory checker has* completeness $c$ *if the following holds for every polynomial $l = l(n)$ and $I$ as above:*

$$\Pr\left[\mathsf{CompletenessExpt}(I) = 1\right] \geq c(n),$$

*where the above probability is taken over the randomness used in the experiment* $\mathsf{CompletenessExpt}(I)$ *that is defined next.*

$\mathsf{CompletenessExpt}(I)$:

1. *Initialize a memory checker by setting the local storage to* $\mathsf{st} = \perp^p$*, and initialize the physical memory* $DB$ *to* $\perp^m$.
2. *For* $j = 1, \ldots, l$:
   (a) *Run the protocol between the memory checker and the physical memory*

   $$\mathsf{MemCheckerOp}(\mathsf{st}, \mathsf{op}_j, \mathsf{addr}_j, \mathsf{data}_j) \Longleftrightarrow DB.$$

   *This protocol outputs a value* $\mathsf{data}'_j \in \{0,1\}^{w_\ell} \cup \{\perp\}$. *It also outputs an updated* $\mathsf{st}'$ *and an updated physical memory* $DB'$*, both of which are used in the next iteration.*

   (b) *Feed* $(\mathsf{op}_j, \mathsf{addr}_j, \mathsf{data}_j)$ *into the logical memory snapshot functionality, getting back a snapshot* $\mathsf{SnapshotMem}_j$.

   (c) *If* $\mathsf{op}_j = \mathsf{read}$ *and* $\mathsf{data}'_j \neq \mathsf{SnapshotMem}_j[\mathsf{addr}_j]$*, output 0 and abort.*
3. *Output* 1.

**Remark 2** (Completeness amplification)**.** *Our definition of completeness allows for amplification by repetition. That is, by running k independent instances of a memory checker at once (sequentially within each logical query) and outputting the majority response, we can improve completeness exponentially in k. This comes at the cost of an* $O(k)$ *multiplicative blowup in query complexity, local space, and public database size.*

For soundness, we consider a malicious PPT adversary playing the role of the physical memory in the interaction with the memory checker. We require that for any such adversary, the probability that the memory checker outputs an *incorrect* value $\sigma \in \{0,1\}^{w_\ell}$ is at most $s(n)$. Below, we formalize the standard definition of soundness for memory checkers [BEG+94, NR09, DNRV09] (used both in upper and lower bounds).

**Definition 5** ($s$-soundness)**.** *Let* $l \in \mathbb{N}$ *and consider a sequence of* $l$ *logical instructions* $I = \{(\mathsf{op}_i, \mathsf{addr}_i, \mathsf{data}_i)\}_{i \in [l]}$*, where* $\mathsf{op} \in \{\mathsf{read}, \mathsf{write}\}$, $\mathsf{addr} \in [n]$*, and* $\mathsf{data} \in \{0,1\}^{w_\ell}$.

*We say that a memory checker has* soundness error $s(n)$ *if for every stateful PPT adversary* $\mathcal{A}$ *and every polynomial* $l = l(n)$ *and* $I$ *as above, it holds that*

$$\Pr[\mathsf{SoundnessExpt}_{\mathcal{A}}(I) = 1] \leq s(n),$$

*where the probability is over the randomness used in the experiment* $\mathsf{SoundnessExpt}_{\mathcal{A}}(I)$ *that is defined next.*

$\mathsf{SoundnessExpt}_{\mathcal{A}}(I)$:

1. *Initialize a memory checker by setting the local storage to* $\mathsf{st} = \perp^p$.
2. *For* $j = 1, \ldots, l$:
   (a) *Run the protocol between the memory checker and* $\mathcal{A}$*, acting as the physical memory:*

   $$\mathsf{MemCheckerOp}(\mathsf{st}, \mathsf{op}_j, \mathsf{addr}_j, \mathsf{data}_j) \Longleftrightarrow \mathcal{A}.$$

*The adversary $\mathcal{A}$ has read access to everything except for the local storage of the memory checker and the random tape of the memory checker. This protocol outputs a value* $\mathsf{data}'_j \in \{0,1\}^{w_\ell} \cup \{\perp\}$*. It also outputs an updated* $\mathsf{st}'$ *and an updated physical memory* $DB'$*, both of which are used in the next iteration.*

(b) *Feed* $(\mathsf{op}_j, \mathsf{addr}_j, \mathsf{data}_j)$ *into the logical memory snapshot functionality, getting back a snapshot* $\mathsf{SnapshotMem}_j$*.*

(c) *If* $\mathsf{op}_j = \mathsf{read}$ *and* $\mathsf{data}'_j \notin \{\mathsf{SnapshotMem}_j[\mathsf{addr}_j], \perp\}$*, output* $1$ *and abort.*

3. *Output* $0$*.*

**Remark 3** (Completeness and soundness parameters)**.** *In our lower bound, we rule out a memory checker where completeness is* $c = c(n) = 2/3$ *and soundness error is* $s = s(n) = 1/3$*.*

**About our definitions.** First, we emphasize that our soundness notion is the standard one, appearing in all prior works studying memory checkers (e.g., [BEG+94, NR09, DNRV09]), wherein this definition is used for upper and lower bounds. The only exception to this statement that we are aware of is the recent work of Boyle et al. [BKV24] that considered a weaker notion of soundness where the requirement is that a single logical read index fixed before running the protocol is protected. We do not know how to avoid polynomial soundness loss with their weaker definition and leave it as an open problem.

Second, we mention a few ways in which our lower abound applies broadly. Our lower bound applies to memory checkers that could have *secret* local storage, even though there are memory checker constructions that have *public but reliable* local storage (e.g., a Merkle tree [Mer89] or the UOWHF construction in Blum et al. [BEG+94].) Furthermore, our lower bound applies even when $w_\ell = 1$. Lastly, our lower bound does not require the memory checker's internal operations to be efficiently computable. In fact, the local space is just the number of bits preserved *in between* user queries, so it is not a true space complexity parameter, as the memory checker could use an unbounded number of bits in the middle of carrying out a user instruction.

# 4 Computing a Partition

We now give an effective version of the partition lemma shown by BKV [BKV24, Lemma 3], in the sense that finding a valid partition of the public database is computationally efficient when one can generate samples from random logical reads to the database. Later, this will be used in the lower bound to argue that we can invoke memory checking soundness on the *efficient* adversary that can compute a partition and perform a replay attack accordingly. (See Figure 6 and its explanation in Theorem 3 for its exact use.)

We briefly discuss the differences between Theorem 2 and [BKV24, Lemma 3]. Item 3 shows a probability bound of $4/(c-1)$ of hitting $M$ instead of $1/c$, but this constant factor difference will not be essential for us. Similarly, Item 4 has $2c$ in the exponent in the denominator instead of $c$, which is also immaterial for our application.

**Theorem 2** (Effective version of Lemma 3 of [BKV24])**.** *There is a polynomial time algorithm* $\mathsf{Partition}^{\mathcal{D}}(1^\gamma, 1^c, 1^n, m)$ *with sample access to a distribution* $\mathcal{D}$ *with the following guarantee. Suppose* $\gamma, c, n \in \mathbb{N}$ *with* $\gamma, c \geq 2$*, and suppose* $\mathcal{D}$ *is a distribution over* $[m]$ *where* $m \leq 2^{\mathrm{poly}(n)}$*. Then, with*

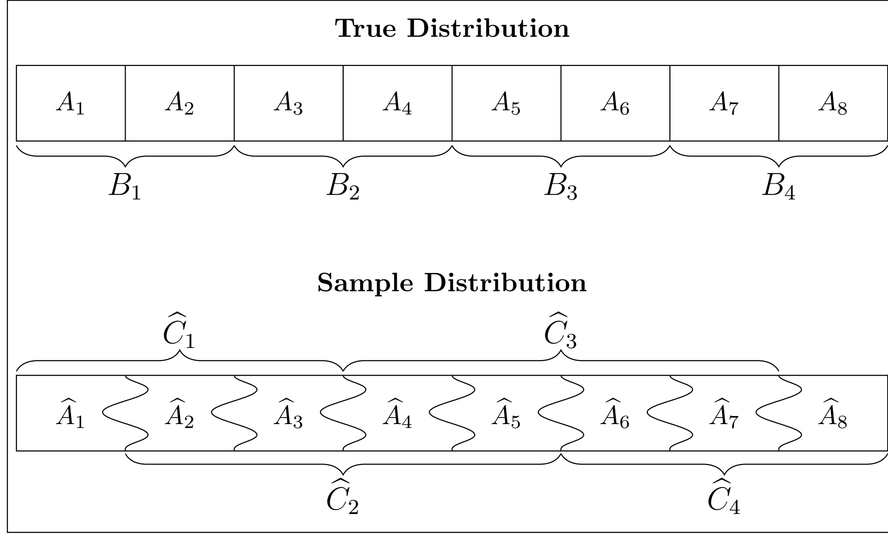


Figure 2: We graphically sketch the idea behind using the "cushioned buckets" $\widehat{C}_i$ in Theorem 2 and in Figure 3. Here, $c = 4$. Note that Claim 1 shows that $B_i \subseteq \widehat{C}_i$ with sufficiently high probability. Furthermore, note that each $\widehat{A}_i$ is contained in at most 2 different cushioned buckets $\widehat{C}_i$, showing that $\{\widehat{C}_i\}_{i\in[c]}$ form (at most) a double cover.

*probability at least* $1 - 1/n$, Partition *outputs some* $H \subseteq [m]$ *and* $i \in [c]$ *such that there exists* $L, M \subseteq [m]$ *such that:*

1. $[m] = H \sqcup M \sqcup L$;
2. *For all* $\ell \in L$, $\Pr_{X\sim\mathcal{D}}[X = \ell] \le (2\gamma)^{2i-2}/n$;
3. $\Pr_{X\sim\mathcal{D}}[X \in M] \le 4/(c-1)$; *and*
4. *The set* $H$ *satisfies*
$$\frac{n}{(2\gamma)^{2i-2}} - |H|\gamma > \frac{n}{(2\gamma)^{2c}}.$$

*Moreover,* Partition *uses* $t = \Theta(n^2 \log(m+n))$ *samples from* $\mathcal{D}$ *and is deterministic (ignoring the randomness in the samples from* $\mathcal{D}$*).*

*Proof of Theorem 2.* We describe the algorithm in Figure 3. Let

$$p_j := \Pr_{X\sim\mathcal{D}}[X = j].$$

Define the "true" buckets

$$A_1 := \left\{ j \in [m] : p_j \in \left[0, \frac{2\gamma}{n}\right) \right\},$$
$$A_i := \left\{ j \in [m] : p_j \in \left[\frac{(2\gamma)^{i-1}}{n}, \frac{(2\gamma)^i}{n}\right) \right\} \text{ for } i \in \{2, \cdots, 2c-1\},$$
$$A_{2c} := \left\{ j \in [m] : p_j \in \left[\frac{(2\gamma)^{2c-1}}{n}, \infty\right) \right\}.$$

$\mathsf{Partition}^{\mathcal{D}}(1^\gamma, 1^c, 1^n, m)$:

1. Let $t := \lceil 100n^2 \ln(m+n) \rceil$. Generate $t$ samples $a_1, \cdots, a_t \sim \mathcal{D}$.
2. For[a] $j \in [m]$, let
$$\widehat{p}_j := \frac{|\{k \in [t] : a_k = j\}|}{t}.$$
Note that
$$\sum_{j \in [m]} \widehat{p}_j = 1. \tag{1}$$
3. Define the buckets
$$\begin{aligned}
\widehat{A}_1 &:= \left\{ j \in [m] : \widehat{p}_j \in \left[0, \frac{2\gamma}{n}\right) \right\}, \\
\widehat{A}_i &:= \left\{ j \in [m] : \widehat{p}_j \in \left[\frac{(2\gamma)^{i-1}}{n}, \frac{(2\gamma)^i}{n}\right) \right\} \text{ for } i \in \{2, \cdots, 2c-1\}, \\
\widehat{A}_{2c} &:= \left\{ j \in [m] : \widehat{p}_j \in \left[\frac{(2\gamma)^{2c-1}}{n}, \infty\right) \right\}.
\end{aligned}$$
Note that these buckets partition $[m]$.
4. Define the "cushioned" buckets
$$\begin{aligned}
\widehat{C}_1 &:= \widehat{A}_1 \sqcup \widehat{A}_2 \sqcup \widehat{A}_3, \\
\widehat{C}_i &:= \widehat{A}_{2i-2} \sqcup \widehat{A}_{2i-1} \sqcup \widehat{A}_{2i} \sqcup \widehat{A}_{2i+1} \text{ for } i \in \{2, \cdots, c-1\}, \\
\widehat{C}_c &:= \widehat{A}_{2c-2} \sqcup \widehat{A}_{2c-1} \sqcup \widehat{A}_{2c}.
\end{aligned}$$
Note that these buckets cover $[m]$. All elements in $\widehat{A}_1$ and $\widehat{A}_{2c}$ are covered once, while all other elements are covered twice.
5. Fix some
$$i^* \in \arg\min_{i \in \{2, \cdots, c\}} \left\{ \sum_{j \in \widehat{C}_i} \widehat{p}_j \right\}.$$
6. Set
$$H := \bigcup_{i \geq i^*+1} \widehat{C}_i.$$
7. Output $(H, i^*)$.

[a] While this is phrased as iterating over $j \in [m]$, the algorithm can instead iterate over just the $t$ samples to lazily generate non-zero $\widehat{p}_j$, which runs in time $\mathrm{poly}(n, \log m)$ instead of $\Omega(m)$.

Figure 3: $\mathsf{Partition}$ algorithm, as in Theorem 2.

Note that these buckets partition $[m]$. Now, define larger buckets

$$B_i := A_{2i-1} \sqcup A_{2i} \text{ for } i \in [c].$$

Observe that these larger buckets also partition $[m]$.

We begin by showing two auxiliary claims, Claims 1 and 2. Using these claims, we prove Items 1 to 4 in Claims 3, 4, 5, and 6, respectively.

**Claim 1.** *For the cushioned buckets $\widehat{C}_i$ defined in Figure 3,*

$$\Pr_{a_1,\cdots,a_t\sim\mathcal{D}}\left[\forall i \in [c], B_i \subseteq \widehat{C}_i\right] \geq 1 - \frac{1}{n}.$$

*Proof.* We consider three cases and union bound over them: $i \in \{2, \cdots, c-1\}$, $i = c$, and $i = 1$.

**Case 1**: $i \in \{2, \cdots, c-1\}$. In this case,

$$\widehat{C}_i = \widehat{A}_{2i-2} \sqcup \widehat{A}_{2i-1} \sqcup \widehat{A}_{2i} \sqcup \widehat{A}_{2i+1} = \left\{j \in [m] : \widehat{p}_j \in \left[\frac{(2\gamma)^{2i-3}}{n}, \frac{(2\gamma)^{2i+1}}{n}\right)\right\},$$
$$B_i = A_{2i-1} \sqcup A_{2i} = \left\{j \in [m] : p_j \in \left[\frac{(2\gamma)^{2i-2}}{n}, \frac{(2\gamma)^{2i}}{n}\right)\right\}.$$

The multiplicative Chernoff bound says that

$$\Pr\left[\widehat{p}_j \notin \left[\frac{p_j}{2}, \frac{3p_j}{2}\right]\right] \leq 2e^{-p_j t/12}.$$

For all $j \in B_i$, we know

$$p_j \geq \frac{(2\gamma)^{2i-2}}{n} \geq \frac{1}{n},$$

so

$$\Pr\left[\widehat{p}_j \notin \left[\frac{p_j}{2}, \frac{3p_j}{2}\right]\right] \leq 2e^{-t/(12n)}. \tag{2}$$

Since $t \geq 100n \ln(m+n)$, union bounding over all possible $j \in [m]$ will result in total success probability at least $1 - 1/(3n)$. Because $2\gamma > 2$, we have (with high probability) that if $p_j \geq (2\gamma)^{2i-2}/n$, then

$$\widehat{p}_j \geq \frac{p_j}{2} > \frac{p_j}{2\gamma} \geq \frac{(2\gamma)^{2i-3}}{n},$$

and similarly, if $p_j < (2\gamma)^{2i}/n$,

$$\widehat{p}_j \leq \frac{3p_j}{2} \leq (2\gamma)p_j \leq \frac{(2\gamma)^{2i+1}}{n}.$$

Therefore, combining these implications, with high probability we have $B_i \subseteq \widehat{C}_i$ for $i \in \{2, \cdots, c-1\}$.

**Case 2**: $i = c$. In this case,

$$\widehat{C}_c = \widehat{A}_{2c-2} \sqcup \widehat{A}_{2c-1} \sqcup \widehat{A}_{2c} = \left\{j \in [m] : \widehat{p}_j \in \left[\frac{(2\gamma)^{2c-3}}{n}, \infty\right)\right\},$$
$$B_c = A_{2c-1} \sqcup A_{2c} = \left\{j \in [m] : p_j \in \left[\frac{(2\gamma)^{2c-2}}{n}, \infty\right)\right\}.$$

By the same multiplicative Chernoff bound, since it still holds that $p_j \ge 1/n$, if $p_j \ge (2\gamma)^{2c-2}/n$, then with high probability,

$$\widehat{p}_j \ge \frac{p_j}{2} > \frac{p_j}{2\gamma} \ge \frac{(2\gamma)^{2c-3}}{n}.$$

Therefore, $B_c \subseteq \widehat{C}_c$.

**Case 3**: $i = 1$. In this case,

$$\widehat{C}_1 = \widehat{A}_1 \sqcup \widehat{A}_2 \sqcup \widehat{A}_3 = \left\{ j \in [m] : \widehat{p}_j \in \left[0, \frac{(2\gamma)^3}{n}\right) \right\},$$
$$B_1 = A_1 \sqcup A_2 = \left\{ j \in [m] : p_j \in \left[0, \frac{(2\gamma)^2}{n}\right) \right\}.$$

Here, we cannot invoke a multiplicative Chernoff bound because $p_j$ could be very small. Instead, we invoke an additive Chernoff bound that says

$$\Pr[\widehat{p}_j \ge p_j + \varepsilon] \le e^{-2\varepsilon^2 t}.$$

We can plug in $\varepsilon = 1/n$ to get

$$\Pr\left[\widehat{p}_j \ge p_j + \frac{1}{n}\right] \le e^{-2t/n^2}.$$

Since $t \ge 2n^2 \ln(m+n)$, this failure probability is at most $1/(m+n)^4$, and when union bounded over all $j \in [m]$, the probability of success for all $j$ is at least $1 - 1/(3n)$. Therefore, if $p_j < (2\gamma)^2/n$, we have (with high probability)

$$\widehat{p}_j < p_j + \frac{1}{n} < \frac{(2\gamma)^2}{n} + \frac{(2\gamma)^2}{n} = \frac{2 \cdot (2\gamma)^2}{n} < \frac{(2\gamma)^3}{n}.$$

This shows that $B_1 \subseteq \widehat{C}_1$, concluding the proof of the claim. □

**Claim 2.** $\sum_{j \in \widehat{C}_{i^*}} \widehat{p}_j \le 2/(c-1)$.

*Proof.* This follows since the $\widehat{C}_i$ form (at most) a double cover of $[m]$. For $i \in [c]$, let $\alpha_i := \sum_{j \in \widehat{C}_i} \widehat{p}_j$ be the (sample) weight of $\widehat{C}_i$. By construction of $\widehat{C}_i$,

$$\begin{aligned}\sum_{i \in \{2, \cdots c\}} \alpha_i \le \sum_{i \in [c]} \alpha_i = \sum_{i \in [c]} \sum_{j \in \widehat{C}_i} \widehat{p}_j &= \sum_{j \in \widehat{A}_i} \widehat{p}_j + \sum_{j \in \widehat{A}_{2c}} \widehat{p}_j + 2 \sum_{2 \le i \le 2c-1} \sum_{j \in \widehat{A}_i} \widehat{p}_j \\ &\le 2 \sum_{i \in [2c]} \sum_{j \in \widehat{A}_i} \widehat{p}_j \\ &= 2 \sum_{j \in [m]} \widehat{p}_j = 2,\end{aligned}$$

where the penultimate equality follows because $\widehat{A}_i$ form a partition of $[m]$, and the last equality holds by (1). Therefore, by an averaging argument, there must exist some $i \in \{2, \cdots, c\}$ for which $\alpha_i \le 2/(c-1)$. By construction of $i^*$, we have shown the desired claim. □

Now, we construct $M$ and $L$. Let $M := B_{i^*} \setminus H$ and $L := \bigcup_{i \le i^*-1} B_i$.

**Claim 3.** *(Item 1) $H$, $M$, and $L$ partition $[m]$.*

*Proof.* Clearly, $M$ and $H$ are disjoint by construction of $M$. $M$ and $L$ are disjoint because $B_i$ form a partition of $[m]$. To see why $L$ and $H$ are disjoint, we can invoke Claim 1 to see that

$$L = \bigcup_{i \le i^*-1} B_i \subseteq \bigcup_{i \le i^*-1} \widehat{C}_i = \bigcup_{i \le i^*-1} \widehat{A}_{2i+1} \subseteq \bigcup_{i \le i^*} \widehat{A}_{2i-1}.$$

On the other hand,

$$H = \bigcup_{i \ge i^*+1} \widehat{C}_i \subseteq \bigcup_{i \ge i^*+1} \widehat{A}_{2i-2} \subseteq \bigcup_{i \ge i^*} \widehat{A}_{2i}.$$

Since $\widehat{A}_i$ form a partition of $[m]$, $L$ and $H$ are disjoint.

Lastly, to see that they union to $[m]$, we can again invoke Claim 1 to see that

$$M \cup H = (B_{i^*} \setminus H) \cup H = B_{i^*} \cup H = B_{i^*} \cup \bigcup_{i \ge i^*+1} \widehat{C}_i \supseteq B_{i^*} \cup \bigcup_{i \ge i^*+1} B_i = \bigcup_{i \ge i^*} B_i.$$

Combining with the definition of $L$, we have

$$L \cup M \cup H = \left( \bigcup_{i \le i^*-1} B_i \right) \cup \left( \bigcup_{i \ge i^*} B_i \right) = \bigcup_{i \in [c]} B_i = [m],$$

as desired. □

**Claim 4.** *(Item 2) For all $\ell \in L$, $\Pr_{X \sim \mathcal{D}}[X = \ell] \le (2\gamma)^{2i^*-2}/n$.*

*Proof.* This follows from the definition of $L$. Explicitly,

$$L = \bigcup_{i \le i^*-1} B_i = \bigcup_{i \le 2i^*-2} A_i = \left\{ j \in [m] : p_j \in \left[ 0, \frac{(2\gamma)^{2i^*-2}}{n} \right) \right\}.$$

□

**Claim 5.** *(Item 3) $\Pr_{X \sim \mathcal{D}}[X \in M] \le 4/(c-1)$.*

*Proof.* By Claim 1, we have

$$\Pr_{X \sim \mathcal{D}}[X \in M] = \sum_{j \in B_{i^*} \setminus H} p_j \le \sum_{j \in B_{i^*}} p_j \le \sum_{j \in \widehat{C}_{i^*}} p_j.$$

Since $i^* \ne 1$ by choice of $i^*$, and by (2), we know with high probability that

$$\sum_{j \in \widehat{C}_{i^*}} p_j \le \sum_{j \in \widehat{C}_{i^*}} 2\widehat{p}_j.$$

By combining all inequalities above and Claim 2, we have

$$\Pr_{X \sim \mathcal{D}}[X \in M] \le \sum_{j \in \widehat{C}_{i^*}} p_j \le 2 \sum_{j \in \widehat{C}_{i^*}} \widehat{p}_j \le \frac{4}{c-1},$$

as desired. □

**Claim 6.** *(Item 4) The set $H$ satisfies*

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

*Proof.* Since

$$H = \bigcup_{i \geq i^*+1} \widehat{C}_i = \bigcup_{i \geq 2i^*} \widehat{A}_i,$$

we know that for all $h \in H$, we have $\widehat{p}_h \geq (2\gamma)^{2i^*-1}/n$. Therefore, by summing up all $\widehat{p}_h$, (1), and counting,

$$|H| \cdot \frac{(2\gamma)^{2i^*-1}}{n} \leq \sum_{h \in H} \widehat{p}_h \leq \sum_{j \in [m]} \widehat{p}_j = 1,$$

from which it follows that $|H| \leq n/(2\gamma)^{2i^*-1}$. Therefore,

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

as desired. □

□

# 5 Main Lower Bound

**Theorem 3** (Main Theorem)**.** *There is a universal constant $C \leq 10^3$ for which the following holds. Consider a memory checker with read-only reads for a logical memory of size $n$ with secret local storage and with logical word size $w_\ell = 1$. Assume that the memory checker has physical database size $m$, physical word size $w$, read query complexity $q_r \leq n/C$, write query complexity $q_w$, and local space $p$. If completeness is $99/100$ and soundness error is $1/3$, then for $n \geq C$,*

$$p \geq \frac{n}{(Cq_r q_w(w + \log m))^{C \cdot q_r}} - \log q_r - C.$$

**Comparison between Theorem 3 and BKV [BKV24, Theorem 5].** The main differences between the statements of Theorem 3 and [BKV24, Theorem 5] is the requirement on soundness (and the read-only reads condition). While their soundness error needs to be at most $O(1/(q_r \cdot 2^{q_r}))$, ours only needs to be at most $1/3$. In short, our proof circumvents this by showing that an efficient adversary can *learn* a sufficiently good partitioning of the physical memory by issuing $\text{poly}(n)$ queries to the memory checker instead of guessing. Besides the description of the adversary used to invoke soundness of the memory checker, the rest of the proof is very similar to [BKV24, Theorem 5]. We borrow notation and more from [BKV24] throughout the proof.

We now state some corollaries of the main theorem, similarly to [BKV24] except with a significantly weakened requirement on the soundness error. Corollary 1 gives a quasi-logarithmic lower bound on the query complexity $q$ of memory checkers, when not differentiating between read and write query complexities. We emphasize that this bound on $q$ is tight up to constant factors; see [BEG+94, PT11, BKV24] for standard constructions. Corollary 2 shows that if $q_r$ is small,

then $q_w$ is large; this is similarly tight up to constant factors in the exponent due to the read-write trade-off of DNRV [DNRV09].

As an aside, these corollaries assume $m \le \text{poly}(n)$ and $w \le \text{polylog}(n)$. We give brief justification for these standard settings of parameters. [BKV24, Appendix B] shows how to obtain $m \le \text{poly}(n)$ generically (using PRFs) when the number of logical queries is bounded above by a fixed polynomial (see also [BKV24, Remark 5]). For larger physical word sizes $w$, [BKV24, Remark 6] shows that if $w = (\log n)^{\omega(1)}$, one can indeed get better memory checkers with query complexity $O(\log_w n)$, assuming the existence of sub-exponentially secure one-way functions. For example, for $w = n^{\varepsilon}$ for some $\varepsilon \in (0, 1)$, there is a memory checker with query complexity $O(1/\varepsilon)$.

**Corollary 1.** *Consider a memory checker with read-only reads for a logical memory of size $n$ with secret local storage and with logical word size $w_\ell = 1$. Assume that the memory checker has physical database size $m \le \text{poly}(n)$, physical word size $w \le \text{polylog}(n)$, query complexity $q$, and local space $p$. If completeness is $2/3$ and soundness error is $1/3$, then*

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

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

*Proof of Corollary 1 assuming Theorem 3.* First, we apply completeness amplification from $2/3$ to $99/100$ by parallel repetition, increasing $q_r$, $q_w$, $m$ and $w$ by at most an $O(1)$ factor. Suppose that $q \ge \log n / \log \log n$. Then, there exists a constant such that $(\log n)^{C \cdot q} = \omega(n)$, making the inequality trivial. Therefore, we can assume $q = \max\{q_r, q_w\} \le \log n / \log \log n$. Thus, $\log q_r = O(\log \log n)$. Furthermore, since $m \le \text{poly}(n)$ and $w \le \text{polylog}(n)$ by assumption, when applying Theorem 3, the denominator becomes $(\log n)^{O(q)}$, as desired. □

**Corollary 2.** *Consider a memory checker with read-only reads for a logical memory of size $n$ with secret local storage and with logical word size $w_\ell = 1$. Assume that the memory checker has physical database size $m \le \text{poly}(n)$, physical word size $w \le \text{polylog}(n)$, read query complexity $q_r$, write query complexity $q_w$, and local space $p \le n^{1-\varepsilon}$ for $\varepsilon > 0$, with completeness $2/3$ and soundness error $1/3$. Then:*

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

*Proof of Corollary 2 given Theorem 3.* First, we apply completeness amplification from $2/3$ to $99/100$ by parallel repetition, increasing $q_r$, $q_w$, $m$ and $w$ by at most an $O(1)$ factor. For $q_r = o(n)$, we can apply Theorem 3 and re-arrange to get

$$q_w \ge \frac{n^{\Theta(1/q_r)}}{q_r \log n}.$$

The result immediately follows. □

Finally, we prove our main theorem, Theorem 3.

*Proof of Theorem 3.* We begin by defining some notation. Suppose $DB \in (\{0,1\}^w \cup \{\perp\})^m$ is a public database, and $\mathsf{st} \in \{0,1\}^p$ is a (secret) local storage. Let $R(i, DB, \mathsf{st}; r) \subseteq [m]$ 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 storage $\mathsf{st}$, and internal randomness $r$ for the logical read.

**"Next-read" distribution.** For a database $DB$ and storage $\mathsf{st}$, define the next-read distribution $\mathcal{D}_{DB,\mathsf{st}}$ as follows:

1. Sample $i \sim [n]$ uniformly at random.
2. Sample internal randomness $r$ for the memory checker uniformly at random.
3. Sample and output a uniformly random element of $R(i, DB, \mathsf{st}; r)$.

Let $c = 100q_r + 1$, and let $\gamma = 200q_r q_w(w + \log m + 2)$ in preparation for applying Theorem 2. For $j \in [c]$, let $\delta_j := (2\gamma)^{2j-2}/n \geq 1/n$ as in Theorem 2, and let $k_j := \lfloor 1/(100\delta_j q_r q_w) \rfloor \leq n/100$. We assume $n$ is a multiple of 10 throughout for simplicity.

We use the memory checker to directly construct a public-coin protocol for Alice and Bob to send a string $x$ chosen uniformly at random from $\{0,1\}^{9n/10+k_{j^*}}$ such that $\|x\|_0 = k_{j^*}$. This construction directly follows [BKV24, Theorem 5], with the main difference being the soundness argument.

**Protocol description.** Alice and Bob share randomness before starting the protocol. First, Alice and Bob run a memory checker and logically write 0 to all logical indices $i \in [n]$. Next, Alice and Bob sample $j^* \sim [c]$ uniformly at random. Next, Alice and Bob will together randomly sample a uniform $y \in \{0,1\}^n$ such that $\|y\|_0 = n/10 - k_{j^*}$, and they will write 1 to all logical indices $i \in [n]$ such that $y_i = 1$ in a uniformly random order. This process yields

$$DB_0 \in (\{0,1\}^w \cup \{\perp\})^m\,, \quad \mathsf{st}_0 \in \{0,1\}^p,$$

representing the public database and local storage of the memory checker. Based on $y$, Alice and Bob will agree on a fixed map $\pi : [9n/10 + k_{j^*}] \to [n]$ that maps $[9n/10 + k_{j^*}]$ bijectively to the logical indices $i \in [n]$ such that $y_i = 0$. For simplicity of notation, let $k := k_{j^*}$ and $\pi(x) := \{\pi(i) : i \in [9n/10 + k], x_i = 1\} \subseteq [n]$.

**Alice's encoding.** Alice will directly encode $x^{(-j^*)} = (x^{(1)}, \cdots, x^{(j^*-1)}, x^{(j^*+1)}, \cdots, x^{(c)})$. For $x := x^{(j^*)}$, Alice will use $DB_0$ and $\mathsf{st}_0$ to write $\pi(x)$ to the memory checker in a uniformly random order, resulting in updated $DB_1$ and $\mathsf{st}_1$. Then, Alice will run $\mathsf{Partition}^{\mathcal{D}_{DB_1,\mathsf{st}_1}}$ on the next-read distribution $\mathcal{D}_{DB_1,\mathsf{st}_1}$ to receive $(H, \widetilde{j})$. Alice will then send $\mathsf{st}_1$, $H$, $DB_1|_H$, and auxiliary information $\mathsf{aux}$ to Bob to help complete Bob's partial information into full information about $x$. See Figure 4 for more technical details.

**Alice's Strategy**

**Shared by Alice and Bob**: $j^*, DB_0, \mathsf{st}_0, y, \pi$
**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 $\mathsf{st}_0$, 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 $\mathsf{st}_1$.
4. Using $DB_1$ and $\mathsf{st}_1$ to define the next-read distribution $\mathcal{D}_{DB_1, \mathsf{st}_1}$, generate $\left(H, \widetilde{j}\right) \leftarrow \mathsf{Partition}^{\mathcal{D}}(1^\gamma, 1^c, 1^n, m)$.
5. Run Steps 2-4 of Bob's strategy (Figure 5) to generate the subset $U \subseteq [n]$.
6. Let $\mathsf{aux}$ be an encoding of the subset $U \cap \pi(x) \subseteq U$.

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

Figure 4: Alice's encoding strategy in the proof of Theorem 3.

The length of Alice's message can be directly bounded above by

$$\left\lceil \log\left(\prod_{\substack{j\in[c]\\ j\neq j^*}} \binom{9n/10 + k_j}{k_j}\right)\right\rceil + |\mathsf{st}_1| + |H|\lceil\log(m)\rceil + |H|\lceil\log(2^w + 1)\rceil + |\mathsf{aux}| \quad (3)$$

$$\leq \log\left(\prod_{j\in[c], j\neq j^*} \binom{9n/10 + k_j}{k_j}\right) + p + |\mathsf{aux}| + |H|(w + \log m + 2) + 1. \quad (4)$$

**Bob's decoding.** Bob will directly decode $x^{(-j^*)}$. To recover $x$, Bob first computes a hybrid database $\widetilde{DB}$, which is $DB_1$ on $H$ and $DB_0$ on $[m] \setminus H$. Then, Bob will use $\mathsf{st}_1$ from Alice and $\widetilde{DB}$ to perform memory checker reads to all indices $i \in [n]$. Let $\tilde{z}_i \in \{0, 1, \perp\}$ denote outcome of each of these reads. Bob will then keep all of these values that are binary and fill in any $\perp$ values using $\mathsf{aux}$ from Alice. See Figure 5 for more technical details.

**Analysis.** Let $z \in \{0,1\}^n$ be the logical memory of the memory checker after writing $y$ and $x$ to their respective parts, namely $z_i = y_i \vee 1[i \in \pi(x)]$ (making $\|z\|_0 = n/10$). Let $\rho$ denote randomness to specify a uniformly random ordering of the support of $z$, representing the order that the support of $y$ and $\pi(x)$ are written into the memory checker.

**Claim 7.** *The distribution of $j^*$ is independent of $z$ and $\rho$. In particular, since $j^* \sim [c]$ uniformly at random,* $\Pr\left[\widetilde{j} = j^*\right] = 1/c$.

**Bob's Strategy**

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

1. Decode $x^{(-j^*)}$ into $\big(x^{(1)}, \cdots, x^{(j^*-1)}, x^{(j^*+1)}, \cdots, x^{(c)}\big)$.
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 $i \in [n]$, perform a logical read to index $i$ using database $\widetilde{DB}$ and local storage $\mathsf{st}_1$, and let $\tilde{z}_i \in \{0, 1, \perp\}$ denote the outcome.
4. Define $U := \{i \in [n] : \tilde{z}_i = \perp\}$.
5. Parsing $\mathsf{aux} \subseteq U$, define $z \in \{0,1\}^n$ by

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

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

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

Figure 5: Bob's decoding strategy in the proof of Theorem 3.

*Proof of Claim 7.* The proof follows by the same reasoning as BKV [BKV24, Claim 1]. In short, for each value of $j^*$, writing $n/10 - k_{j^*}$ uniformly random distinct indices in a uniformly random order and then writing to $k_{j^*}$ uniformly random distinct indices (outside of those already written to) in a uniformly random order is equivalent to writing $n/10$ uniformly random distinct indices in a uniformly random order. To see this more formally, the probability of a particular $z$ and $\rho$ can be written as the reciprocal of

$$\binom{n}{n/10 - k_{j^*}} \cdot (n/10 - k_{j^*})! \cdot \binom{9n/10 + k_{j^*}}{k_{j^*}} \cdot k_{j^*}! = \frac{n!}{(9n/10)!} = \binom{n}{n/10} \cdot (n/10)!,$$

which has no dependence on $j^*$. □

**Using completeness.** Throughout, we will assume that the call to $\mathsf{Partition}()$ from Alice succeeds. This occurs with probability at least $1 - 1/n$, and hence can be ignore up to an additive $1/n$ in the success probability of the protocol. Fix $\mathcal{D} = \mathcal{D}_{DB_1,\mathsf{st}_1}$ to be the next-read distribution. Fix $H \sqcup M \sqcup L = [m]$ and $\widetilde{j}$ as the output of $\mathsf{Partition}$, which we know satisfy the following properties:

1. For all $\ell \in L$, $\Pr_{X \sim \mathcal{D}}[X = \ell] \leq (2\gamma)^{2\widetilde{j}-2}/n = \delta_{\widetilde{j}}$;
2. $\Pr_{X \sim \mathcal{D}}[X \in M] \leq 4/(c-1) = 1/(25q_r)$; and
3. The set $H$ satisfies
$$\frac{1}{\delta_{\widetilde{j}}} - |H|\gamma > \frac{n}{(2\gamma)^{2c}}.$$

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

$$R(i, DB_1, \mathsf{st}_1; r) \cap \mathrm{BAD} = \emptyset,$$

then

$$DB_1|_{R(i,DB_1,\mathsf{st}_1;r)} = \widetilde{DB}|_{R(i,DB_1,\mathsf{st}_1;r)},$$

allowing us to invoke completeness of the memory checker. To argue that $R(i, DB_1, \mathsf{st}_1; r) \cap \mathrm{BAD} = \emptyset$ occurs with decent probability, we show the following claim, analogous to Claim 2 of [BKV24]:

**Claim 8.** *Assuming* $\widetilde{j} = j^*$,

$$\Pr_{i \sim [n], r}[R(i, DB_1, \mathsf{st}_1; r) \cap BAD = \emptyset] \geq \frac{19}{20}.$$

*Proof of Claim 8.* We can decompose $\mathrm{BAD} = (W \cap L) \cup (W \cap M)$ and analyze the two cases separately.

For $W \cap L$, by definition of $L$, we know

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

Therefore, by construction of $\mathcal{D}$ and the union bound, we have

$$\begin{aligned}\Pr_{i\sim[n],r}[R(i,DB_1,\mathsf{st}_1;r)\cap(W\cap L)\neq\emptyset] &\leq q_r\cdot\Pr_{X\sim\mathcal{D}}[X\in W\cap L]\\ &\leq k_{j^*}q_wq_r\delta_{j^*}\leq\frac{1}{100},\end{aligned}$$

by our setting of the parameter $k_{j^*}=\lfloor 1/(100\delta_{j^*}q_rq_w)\rfloor$.

For $W\cap M$, notice that $\Pr_{X\sim\mathcal{D}}[X\in M]\leq 1/(25q_r)$. By construction of $\mathcal{D}$ and the union bound,

$$\begin{aligned}\Pr_{i\sim[n],r}[R(i,DB_1,\mathsf{st}_1;r)\cap(W\cap M)\neq\emptyset] &\leq \Pr_{i\sim[n],r}[R(i,DB_1,\mathsf{st}_1;r)\cap M\neq\emptyset]\\ &\leq q_r\cdot\Pr_{X\sim\mathcal{D}}[X\in M]\leq\frac{1}{25}.\end{aligned}$$

By union bounding over both cases, we have

$$\Pr_{i\sim[n],r}[R(i,DB_1,\mathsf{st}_1;r)\cap\mathrm{BAD}\neq\emptyset]\leq\frac{1}{100}+\frac{1}{25}=\frac{1}{20},$$

as desired. □

Now, we can use completeness of the memory checker to argue that Bob will recover the *correct, binary* values for a constant fraction of $z$ (conditioned on $\widetilde{j}=j^*$), in the following sense:

**Claim 9.**

$$\Pr\left[|\{i\in[n]:\tilde{z}_i=z_i\}|\geq\frac{4n}{5}\;\middle|\;\widetilde{j}=j^*\right]\geq\frac{7}{10}.$$

*Proof of Claim 9.* To apply memory checking completeness, consider the sequence of operations that writes 0 to all $i\in[n]$, writes 1 to all of $z$ in the order $\rho$, and then reads to index $i$ to receive an outcome in $\{0,1,\perp\}$. By the "read-only reads" property, this outcome is identical to Bob's $\tilde{z}_i$, since the only difference in between the two is logical reads. Therefore, by completeness, for each $i\in[n]$, we have

$$\Pr_r[\tilde{z}_i=z_i\mid R(i,DB_1,\mathsf{st}_1;r)\cap\mathrm{BAD}=\emptyset]\geq\frac{99}{100},$$

where the probability here (and later) is implicitly also over the randomness of $z$, $\rho$, and the randomness of the memory checker up until the logical read. By Claim 7, we know that we also have

$$\Pr_r\left[\tilde{z}_i=z_i\;\middle|\;R(i,DB_1,\mathsf{st}_1;r)\cap\mathrm{BAD}=\emptyset,\widetilde{j}=j^*\right]\geq\frac{99}{100},$$

incorporating the distribution over $j^*$ as well. By taking the convex combination over $i\in[n]$, we have

$$\Pr_{i\sim[n],r}\left[\tilde{z}_i=z_i\;\middle|\;R(i,DB_1,\mathsf{st}_1;r)\cap\mathrm{BAD}=\emptyset,\widetilde{j}=j^*\right]\geq\frac{99}{100}.$$

By the bound $\Pr[A|C]\geq\Pr[A|B,C]\cdot\Pr[B|C]$, we can use the previous line and Claim 8 to get

$$\begin{aligned}\Pr_{i\sim[n],r}\left[\tilde{z}_i=z_i\;\middle|\;\widetilde{j}=j^*\right] &\geq \Pr_{i\sim[n],r}\left[\tilde{z}_i=z_i\;\middle|\;R(i,DB_1,\mathsf{st}_1;r)\cap\mathrm{BAD}=\emptyset,\widetilde{j}=j^*\right]\\ &\quad\cdot\Pr_{i\sim[n],r}\left[R(i,DB_1,\mathsf{st}_1;r)\cap\mathrm{BAD}=\emptyset\;\middle|\;\widetilde{j}=j^*\right]\\ &\geq\frac{99}{100}\cdot\frac{19}{20}>\frac{47}{50}.\end{aligned}$$

Therefore, the expected number of $i \in [n]$ such that $\tilde{z}_i = z_i$ (given $\widetilde{j} = j^*$) is at least $47n/50$. By Markov's inequality, this implies

$$\Pr\left[|\{i \in [n] : \tilde{z}_i = z_i\}| \geq \frac{4n}{5} \,\middle|\, \widetilde{j} = j^*\right] \geq \frac{7}{10},$$

as desired. □

**Using soundness.** We now describe the computationally efficient attack that the adversary carries out, against which we apply soundness of the memory checker. Consider the memory checker that first writes 0 to all logical indices $i \in [n]$, and then writes $z$ in a random order specified $\rho$. After this point, the adversary will choose $t = \Theta(n^2 \log(m+n))$ uniformly random logical read indices $i \in [n]$ and record uniformly random a physical query location from the resulting physical reads. Next, the adversary will run $\mathsf{Partition}(1^\gamma, 1^c, 1^n, m)$ using this list of samples it has generated to compute a heavy set $H$ and corresponding index $\widetilde{j} \in [c]$. From this index, the adversary will compute $k := k_{\widetilde{j}}$ and $\widetilde{DB}$ according to $k$ just as Bob does. Finally, the adversary will perform logical reads to all $i \in [n]$. We give more technical details for the adversary in Figure 6.

Note that conditioned on $\widetilde{j} = j^*$, by the "read-only reads" property, the values $\tilde{z}_i$ recovered by Bob are identical to the values the adversary recovers from the last $n$ logical reads. As a result, since $j^*$ is uniform and independent of the full view of the memory checker (Claim 7), we have the following, due to soundness:

$$\Pr\left[\exists i \in [n], \tilde{z}_i \notin \{z_i, \bot\} \,\middle|\, \widetilde{j} = j^*\right] \leq \frac{1}{3}.$$

Negating, this becomes

$$\Pr\left[\forall i \in [n], \tilde{z}_i \in \{z_i, \bot\} \,\middle|\, \widetilde{j} = j^*\right] \geq \frac{2}{3}. \tag{5}$$

Combining Claim 9 and (5) via the union bound, and the fact that $\Pr\left[\widetilde{j} = j^*\right] = 1/c$, we have

$$\begin{aligned}
&\Pr\left[(\forall i \in [n], \tilde{z}_i \in \{z_i, \bot\}) \wedge \left(|\{i \in [n] : \tilde{z}_i = z_i\}| \geq \frac{4n}{5}\right) \wedge \left(\widetilde{j} = j^*\right)\right] \geq \\
&\Pr\left[(\forall i \in [n], \tilde{z}_i \in \{z_i, \bot\}) \wedge \left(|\{i \in [n] : \tilde{z}_i = z_i\}| \geq \frac{4n}{5}\right) \,\middle|\, \widetilde{j} = j^*\right] \cdot \Pr\left[\widetilde{j} = j^*\right] \\
&\geq \left(1 - \frac{3}{10} - \frac{1}{3}\right) \cdot \frac{1}{c} \geq \frac{1}{3c} = \frac{1}{300q_r + 3} \geq \frac{1}{303q_r}.
\end{aligned}$$

Since we have assumed throughout that $\mathsf{Partition}$ is successful, union bounding over the case that $\mathsf{Partition}$ fails, we know that $\mathsf{Partition}$ succeeds and the three conditions above hold with probability at least $\alpha := 1/(303q_r) - 1/n$. Assuming $q_r \leq n/500$, we have $\alpha \geq 1/(800q_r)$.

Note that if all of the above conditions hold, then Bob successfully recovers Alice's input. Since $|\{i \in [n] : \tilde{z}_i = z_i\}| \geq 4n/5$, Alice's string $\mathsf{aux}$ can be used to encode a subset of the $n/5$ remaining indices in $[n]$ of size at most $k$ (where $k \leq n/100$). Therefore,

$$|\mathsf{aux}| \leq \left\lceil \log \binom{n/5}{k} \right\rceil \leq \log \binom{n/5}{k} + 1.$$

**Description of $\mathcal{A}$ (for soundness)**

- For the first $n + n/10$ queries (i.e., for writing 0 to all of $[n]$ and then 1 to $z$):
  - 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.[a]
- Initialize an empty list $D$.
- For the next $t = \Theta(n^2 \log(m+n))$ reads to random logical locations $i \sim [n]$:
  - Behave honestly, and add a uniformly random one of the (at most) $q_r$ physical locations to the list $D$.
- Compute $(H, \widetilde{j}) \leftarrow \mathsf{Partition}^D(1^\gamma, 1^c, 1^n, m)$, where the samples are read linearly from the list $D$.
- Let $k := k_{\widetilde{j}} = \lfloor n/(100(2\gamma)^{2\widetilde{j}-2} q_r q_w) \rfloor$, and 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.
- Define the database $\widetilde{DB} \in (\{0,1\}^w \cup \{\perp\})^m$ as follows:

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

- For the last $n$ queries (i.e., for the final logical reads):
  - Perform all physical queries from the memory checker with respect to $\widetilde{DB}$.

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

Figure 6: Description of memory checking adversary, to apply soundness of the memory checker in Theorem 3.

As such, by (4), the length of Alice's message can be directly bounded above by

$$\log\left(\prod_{j\in[c],j\neq j^*}\binom{9n/10+k_j}{k_j}\right)+\log\binom{n/5}{k}+p+|H|(w+\log m+2)+2.$$

We now invoke the compression lemma (Lemma 1).[2] Since the probability of success is at least $\alpha$, we have

$$\begin{aligned}&\log\left(\prod_{j\in[c],j\neq j^*}\binom{9n/10+k_j}{k_j}\right)+\log\binom{n/5}{k}+p+|H|(w+\log m+2)+2\\ \geq\ &\log\left(\prod_{j\in[c]}\binom{9n/10+k_j}{k_j}\right)-\log(1/\alpha),\end{aligned}$$

which simplifies to

$$p\geq\log\binom{9n/10+k}{k}-\log\binom{n/5}{k}-|H|(w+\log m+2)-\log(1/\alpha)-2.$$

By using the standard bound on binomial coefficients $(a/b)^b\leq\binom{a}{b}\leq(ea/b)^b$, this becomes

$$\begin{aligned}p&\geq k\log(9n/(10k))-k\log(en/(5k))-|H|(w+\log m+2)-\log(1/\alpha)-2\\ &=k\log(9/(2e))-|H|(w+\log m+2)-\log(1/\alpha)-2\\ &\geq\frac{k}{2}-|H|(w+\log m+2)-\log(1/\alpha)-2\\ &\geq\frac{1}{200\delta_{j^*}q_rq_w}-|H|(w+\log m+2)-\log(1/\alpha)-\frac{5}{2}\\ &\geq\frac{1}{200q_rq_w}\left(\frac{1}{\delta_{j^*}}-|H|\cdot\underbrace{200q_rq_w(w+\log m+2)}_{\gamma}\right)-\log(1/\alpha)-\frac{5}{2}.\end{aligned}$$

Using the guarantee (Item 3) from Partition, we have

$$\begin{aligned}p&\geq\frac{1}{200q_rq_w}\cdot\frac{n}{(400q_rq_w(w+\log m+2))^{2c}}-\log(1/\alpha)-\frac{5}{2}\\ &=\frac{1}{200q_rq_w}\cdot\frac{n}{(400q_rq_w(w+\log m+2))^{200q_r+2}}-\log(1/\alpha)-\frac{5}{2}\\ &\geq\frac{n}{(400q_rq_w(w+\log m+2))^{203q_r}}-\log(800q_r)-\frac{5}{2}\\ &\geq\frac{n}{(600q_rq_w(w+\log m))^{406q_r}}-\log(q_r)-13,\end{aligned}$$

as desired. □

[2]As stated, the length of this message is technically not fixed, but is rather a random variable that depends on the protocol's execution. Our proof will implicitly show that with sufficiently high probability, this random variable can be upper bounded by a small enough fixed quantity to invoke Lemma 1 and obtain a lower bound on $p=|\mathsf{st}_1|$.

## Acknowledgements

We thank the anonymous reviewers for their valuable feedback. The first author is supported in part by AFOSR Award FA9550-21-1-0046 and ERC Project HSS (852952). The second author is supported in part by a grant from the Israel Science Foundation (ISF Grant No. 1774/20), by a grant from the US-Israel Binational Science Foundation and the US National Science Foundation (BSF-NSF Grant No. 2020643), and by the European Union (ERC, SCALE,101162665). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them. For the third author, research was partially done at NTT Research. The third author is further supported in part by DARPA under Agreement No. HR00112020023, NSF CNS-2154149, NSF DGE-2141064, and a Simons Investigator Award.

## References

[ABC+07] Giuseppe Ateniese, Randal C. Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary N. J. Peterson, and Dawn Xiaodong Song. Provable data possession at untrusted stores. In *CCS*, pages 598–609. ACM, 2007. 4

[AL10] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *J. Cryptol.*, 23(2):281–343, 2010. 2, 3

[AST24] Arasu Arun, Srinath T. V. Setty, and Justin Thaler. Jolt: Snarks for virtual machines via lookups. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part VI*, volume 14656 of *Lecture Notes in Computer Science*, pages 3–33. Springer, 2024. 4

[BCHO22] Jonathan Bootle, Alessandro Chiesa, Yuncong Hu, and Michele Orrù. Gemini: Elastic SNARKs for diverse environments. In *EUROCRYPT*, pages 427–457, 2022. 4

[BDFG21] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Halo infinite: Proof-carrying data from additive polynomial commitments. In *CRYPTO*, pages 649–680, 2021. 4

[BEG+94] Manuel Blum, Will Evans, Peter Gemmell, Sampath Kannan, and Moni Naor. Checking the correctness of memories. *Algorithmica*, 12:225–244, 1994. 2, 4, 5, 9, 12, 13, 19

[BIN97] Mihir Bellare, Russell Impagliazzo, and Moni Naor. Does parallel repetition lower the error in computationally sound protocols? In *FOCS*, pages 374–383, 1997. 5

[BKV24] Elette Boyle, Ilan Komargodski, and Neekon Vafa. Memory checking requires logarithmic overhead. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 1712–1723. ACM, 2024. 2, 3, 4, 5, 6, 7, 8, 9, 11, 13, 19, 20, 21, 24

[BMM+21] Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi Vesely. Proofs for inner pairing products and applications. In *ASIACRYPT*, pages 65–97, 2021. 4

[CD16] Victor Costan and Srinivas Devadas. Intel SGX explained. *IACR Cryptol. ePrint Arch.*, page 86, 2016. 2

[CKW17] David Cash, Alptekin Küpçü, and Daniel Wichs. Dynamic proofs of retrievability via oblivious RAM. *J. Cryptol.*, 30(1):22–57, 2017. 4

[CSG+05] Dwaine E. Clarke, G. Edward Suh, Blaise Gassend, Ajay Sudan, Marten van Dijk, and Srinivas Devadas. Towards constant bandwidth overhead integrity checking of untrusted data. In *IEEE S&P*, pages 139–153, 2005. 2

[DNRV09] Cynthia Dwork, Moni Naor, Guy N Rothblum, and Vinod Vaikuntanathan. How efficient can memory checking be? In *TCC*, pages 503–520, 2009. 2, 3, 5, 9, 12, 13, 20

[DTT10] Anindya De, Luca Trevisan, and Madhur Tulsiani. Time space tradeoffs for attacks against one-way functions and PRGs. In *CRYPTO*, pages 649–665, 2010. 9

[GMS08] Vipul Goyal, Payman Mohassel, and Adam D. Smith. Efficient two party and multi party computation against covert adversaries. In Nigel P. Smart, editor, *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, volume 4965 of *Lecture Notes in Computer Science*, pages 289–306. Springer, 2008. 3

[Gol23] Oded Goldreich. On the lower bound on the length of relaxed locally decodable codes. *Electron. Colloquium Comput. Complex.*, TR23-064, 2023. 8

[HJ05] William Eric Hall and Charanjit S. Jutla. Parallelizable authentication trees. In *Selected Areas in Cryptography*, pages 95–109, 2005. 2

[HL10] Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. *J. Cryptol.*, 23(3):422–456, 2010. 3

[JJ07] Ari Juels and Burton S. Kaliski Jr. PORs: proofs of retrievability for large files. In *CCS*, pages 584–597. ACM, 2007. 4

[Mat23] Surya Mathialagan. Memory checking for parallel rams. In *TCC (2)*, volume 14370 of *Lecture Notes in Computer Science*, pages 436–464. Springer, 2023. 4

[Mer89] Ralph C. Merkle. A certified digital signature. In *Advances in Cryptology - CRYPTO*, pages 218–238, 1989. 4, 13

[NR09] Moni Naor and Guy N Rothblum. The complexity of online memory checking. *Journal of the ACM*, 56(1):1–46, 2009. 2, 9, 12, 13

[NW93] Noam Nisan and Avi Wigderson. Rounds in communication complexity revisited. *SIAM J. Comput.*, 22(1):211–219, 1993. 5

[OR07] Alina Oprea and Michael K. Reiter. Integrity checking in cryptographic file systems with constant trusted storage. In *USENIX*, 2007. 2, 4

[OWWB20] Alex Ozdemir, Riad S. Wahby, Barry Whitehat, and Dan Boneh. Scaling verifiable computation using efficient set accumulators. In *USENIX*, pages 2075–2092, 2020. 4

[PT11] Charalampos Papamanthou and Roberto Tamassia. Optimal and parallel online memory checking. *Cryptology ePrint Archive*, 2011. 2, 4, 19

[SCP+22] Shravan Srinivasan, Alexander Chepurnoy, Charalampos Papamanthou, Alin Tomescu, and Yupeng Zhang. Hyperproofs: Aggregating and maintaining proofs in vector commitments. In *USENIX Security Symposium*, pages 3001–3018, 2022. 4

[Set20] Srinath T. V. Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In *CRYPTO*, pages 704–737, 2020. 4

[ST25] Srinath T. V. Setty and Justin Thaler. Twist and shout: Faster memory checking arguments via one-hot addressing and increments. *IACR Cryptol. ePrint Arch.*, page 105, 2025. 4

[STW24] Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Unlocking the lookup singularity with lasso. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part VI*, volume 14656 of *Lecture Notes in Computer Science*, pages 180–209. Springer, 2024. 4

[SW13] Hovav Shacham and Brent Waters. Compact proofs of retrievability. *J. Cryptol.*, 26(3):442–483, 2013. 4

[WLPZ23] Weijie Wang, Yujie Lu, Charalampos Papamanthou, and Fan Zhang. The locality of memory checking. In *ACM CCS*, pages 1820–1834, 2023. 4

[WTS+18] Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *IEEE S&P*, pages 926–943, 2018. 4

[WUP23] Weijie Wang, Annie Ulichney, and Charalampos Papamanthou. Balanceproofs: Maintainable vector commitments with fast aggregation. In *USENIX Security Symposium*, pages 4409–4426, 2023. 4

[XZZ+19] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In *CRYPTO*, pages 733–764, 2019. 4

[ZXZS20] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *IEEE S&P*, pages 859–876, 2020. 4