Cryptography(CSL7480) Project

A Detailed Analysis on

The Complexity of Memory Checking with Covert Security

Soumen Kumar
B22ES006
b22es006@iitj.ac.in

Tharakadatta D Hedge
B22CS102
b22cs102@iitj.ac.in

April 18, 2025

Contents

1 Introduction

The increasing reliance on remote storage and computation, epitomized by cloud services and distributed systems, raises fundamental questions about data integrity. How can a user trust that data stored on a potentially unreliable or even malicious remote server remains correct and untampered with? Memory checking, introduced by Blum et al. [1991], provides a formal cryptographic framework to address this challenge [Boyle et al., 2025]. A memory checker acts as an intermediary, a trusted proxy, between a user and an untrusted remote server storing a large dataset. Using only a small amount of secure local memory, the checker processes the user's read and write requests, translating them into physical queries to the remote server. The crucial guarantee is **soundness**: the checker ensures that for any user request, it either returns the demonstrably correct result or explicitly signals a failure (by outputting '$\perp$') if tampering is detected. It should never return an incorrect value [Boy].

The efficiency of a memory checker is typically measured by two key parameters: its local space complexity p (the size in bits of the trusted local storage maintained between operations) and its query complexity q (the maximum number of physical queries made to the remote server for each logical user operation) [Boyle et al., 2025]. While early work showed that **information-theoretically** secure checkers (secure against unbounded servers) require a prohibitive trade-off ($p \cdot q = \Omega(n)$, where n is the logical memory size) [Blum et al., 1991], the situation is much better in the **computational** setting, assuming the server is computationally bounded (runs in probabilistic polynomial time, PPT). Here, constructions based on cryptographic primitives like Merkle trees [Merkle, 1989] can achieve remarkably low overhead, notably $q = O(\log n / \log \log n)$ query complexity with small (e.g., polylogarithmic or constant) local storage p [Blum et al., 1991, Papamanthou and Tamassia, 2011, Boyle et al., 2024].

A longstanding open question was whether this logarithmic query overhead was fundamentally necessary. While Dwork et al. [2009] established a similar lower bound for restricted types of checkers (deterministic and non-adaptive), a general lower bound was only recently achieved by Boyle et al. [2024]. They proved that any computational memory checker, regardless of its adaptivity, use of randomness, or underlying cryptographic assumptions, must satisfy the trade-off $p \geq n/(\log n)^{O(q)}$. This implies that if the local storage p is sub-polynomial in n (e.g., $p \leq n^{1-\epsilon}$), then the query complexity must be $q = \Omega(\log n / \log \log n)$, matching the known constructions up to constant factors in the exponent [Boyle et al., 2024, 2025].

However, the powerful result of Boyle et al. [2024] critically relies on the memory checker providing a very strong security guarantee: **negligible soundness error**, meaning the probability of the checker returning an incorrect value is inverse-polynomial in n (e.g., $1/n^{1+o(1)}$). Standard techniques for amplifying soundness error (e.g., via repetition) are either ineffective or too costly in the context of online memory checkers, degrading the lower bound significantly [Boyle et al., 2025]. This left a crucial question unanswered: Can we achieve better efficiency, perhaps even constant query complexity $q = O(1)$, if we relax the security guarantee?

This report delves into the paper "The Complexity of Memory Checking with Covert Security" by Boyle et al. [2025], Boy, which directly addresses this question. The authors investigate memory checking under the **covert security**model, introduced by Afek and Lindell [2010]. In this model, a malicious server attempting to cheat (i.e., cause the checker to return an incorrect value) is only guaranteed to be caught with a constant probability ϵ (e.g., $\epsilon = 1/3$). The server *can* cheat successfully with probability $1 - \epsilon$, but risks detection [Boyle et al., 2025]. This model is practically motivated by scenarios where the potential consequences of being caught (reputation loss, financial penalties, slashed stake in blockchain systems) provide sufficient deterrence [Boy]. Since relaxing from malicious to covert security has led to significant asymptotic efficiency gains in other areas of secure computation [Afek and Lindell, 2010], it was plausible that a similar benefit might apply to memory checking.

The main achievement of Boyle et al. [2025] is to show that, perhaps surprisingly, this is not the case. They prove that the $\Omega(\log n / \log \log n)$ lower bound on query complexity essentially persists even in the covert security setting (constant soundness), provided the checker satisfies a natural property called "read-only reads". This property requires that logical read operations do not modify the remote database or the checker's local state [Boyle et al., 2025]. Their result effectively separates memory checking from other functionalities where covert security offers substantial advantages and underscores the inherent complexity of verifying outsourced memory online.

This report aims to provide a detailed description and analysis of the work by Boyle et al. [2025]. We will cover:

- The importance of the memory checking problem (Section 2).
- Essential background concepts, including formal definitions of memory checkers, security notions (completeness, soundness, covert security), and prior lower bounds (Section 3 , 4).
- The main results achieved in the paper, including the formal lower bound theorem and its implications (Section 5).
- An intuitive explanation of the proof techniques, highlighting the challenges posed by constant soundness and the key ideas used to overcome them (Section 6).
- A discussion of the open challenges mentioned in the paper and potential avenues for future research (Section 7).
- A concluding summary (Section 8).

2 The Importance of Memory Integrity Verification

Ensuring the integrity of data, especially when it resides outside of a user's direct control, is a cornerstone of computer security. The problem that memory checking addresses—verifying the correctness of data stored on an untrusted remote server—is deeply relevant in numerous modern computing paradigms.

Ubiquity of Outsourced Storage: Cloud storage is pervasive. Individuals and organizations entrust vast amounts of data to third-party providers. While providers offer convenience and scalability, the user fundamentally loses direct control over the physical storage medium. A misconfiguration, a software bug, or a malicious act (by an external attacker or a rogue insider at the provider) could lead to data corruption or unauthorized modification. Memory checkers provide a mechanism for users to regain assurance about their data's integrity without needing to download and verify the entire dataset frequently [Boyle et al., 2025].

Foundation for Verifiable Computation: Memory checking is closely related to, and sometimes a component of, broader verifiable computation systems [Boy]. These systems allow a computationally weak client to outsource complex computations to a powerful server and receive a result along with a succinct proof of correctness. Verifying the integrity of intermediate memory states during the computation is often a critical sub-problem. Lower bounds on memory checking, therefore, have direct implications for the efficiency limits of certain verifiable computation schemes [Boyle et al., 2025].

Blockchain and Distributed Systems: In decentralized systems like blockchains, participants often rely on data stored and processed collectively. Ensuring the integrity of this shared state is paramount. Applications like maintainable vector commitments (MVCs), which allow for efficient updates and proofs of membership/non-membership in large datasets, are conceptually linked to memory checking. Lower bounds on memory checkers translate to limitations on MVCs, impacting the scalability and efficiency of potential blockchain applications that rely on them [Boyle et al., 2025, Boy]. The deterrent effect of covert security (risking slashed stake) is particularly relevant here [Boyle et al., 2025].

Mitigating Software Vulnerabilities: While memory checking primarily addresses threats from the *server*, the underlying principles relate to broader issues of memory safety within software. A significant fraction of critical security vulnerabilities in complex software systems stem from memory safety errors (e.g., buffer overflows, use-after-free) [chr, acm]. While memory checkers don't directly fix these application-level bugs, they operate in an environment where such bugs might exist on the server side, potentially leading to corruption. The difficulty of managing sensitive data securely in volatile memory, even in managed languages [sec], further highlights the need for robust mechanisms to verify data integrity when direct memory control is complex or delegated. Memory checkers provide a formal way to reason about and guarantee integrity despite the potential untrustworthiness or fallibility of the storage layer.

In essence, memory checking tackles a fundamental trust issue in distributed computation and storage. Understanding its inherent complexity limits—how efficiently we can perform this verification—is crucial for designing practical, secure systems and for knowing where the performance bottlenecks lie. The question addressed by Boyle et al. [2025]—whether relaxing the security guarantee offers an escape from known efficiency limits—is therefore highly significant, as a positive answer could have opened doors to much faster verification methods, while a negative answer (as they found) reinforces our understanding of the fundamental costs involved.

3 Background Papers

3.1 Memory Checkers: Architecture and Definitions

A memory checker emulates a RAM-like interface for a user while outsourcing actual storage to an untrusted remote server. The user interacts with a logical address space $[n]$ using words of size w_l, but the physical memory, managed by the remote server, has a potentially larger address space $[m]$ with words of size w [Boyle et al., 2025].

The checker maintains a small, trusted local state st of p bits between operations. Each logical operation—such as 'read(addr)' or 'write(addr, data)'—is mapped to a (possibly randomized and adaptive) sequence of physical memory accesses. These accesses are modeled by a protocol called 'MemCheckerOp':

$$\langle(\text{data}', st'), DB'\rangle \leftarrow \langle \text{MemCheckerOp}(st, \text{op}, \text{addr}, \text{data}) \Leftrightarrow DB\rangle$$

This interaction takes the current local state and user's operation, queries the physical memory, and returns either a result 'data'' (or $\perp$ on failure) and updated state/database [Boyle et al., 2025, Sec 3.1].

The protocol is judged by the following properties:

- **Completeness:** If the server behaves honestly, then the checker returns the correct result with high probability—typically $\geq 2/3$ or $1 - \text{negl}(n)$.
- **Soundness:** Against a cheating (PPT) adversary, the checker should detect incorrect behavior with high probability. That is, the probability of returning an incorrect (non-$\perp$) value on a read is bounded—typically by $\leq 1/3$ (covert) or $\leq \text{negl}(n)$ (standard).

The performance of a memory checker is measured via:

- **Local Storage Size** $p = |st|$
- **Query Complexity** $q = \max(q_r, q_w)$, where q_r and q_w denote the number of physical memory queries per logical read/write, respectively [Boyle et al., 2025, Def 2].

This report focuses on the *computational setting*, where the adversary is bounded by polynomial time, aligning with modern cryptographic assumptions [Boyle et al., 2025, Sec 1].

3.2 Prior Lower Bounds on Memory Checkers

Research on memory checking dates back to the foundational work of Blum et al. [Blum et al., 1991], which established a lower bound in the information-theoretic setting: $p \cdot q = \Omega(n)$. That is, one cannot achieve both small local state and low query complexity if adversaries are computationally unbounded.

In the computational setting, where the adversary is restricted to PPT algorithms, the following key results shaped the landscape before Boyle et al. [2025]:

- **Damgård et al. (2009)** [Dwork et al., 2009]: Showed that for deterministic, non-adaptive checkers (i.e., where the physical memory locations accessed are fixed per logical operation), the lower bound becomes $p \geq n/(\log n)^{O(q)}$. This result was significant, though its applicability was limited by its restriction to non-adaptive protocols [Boyle et al., 2025, Fig. 1].
- **Boyle, Komargodski, and Vafa (2024)** [Boyle et al., 2024]: Extended the above to *general* checkers—randomized, adaptive, and based on any cryptographic assumption—proving the same lower bound $p \geq n/(\log n)^{O(q)}$. However, their result applied only in the strong setting of negligible soundness error (e.g., error $\leq 1/n^{1+o(1)}$).

This left an important open question: *Can we reduce the query overhead if we settle for weaker (e.g., constant) soundness guarantees?*

3.3 Covert Security: A Relaxed Soundness Model

The covert security model, originally introduced by Afek and Lindell [2010] in the context of secure computation, offers a practical relaxation to standard malicious security. Rather than aiming to prevent all adversarial misbehavior, covert security ensures that any such misbehavior is detected with some constant probability ϵ (typically 1/3). Formally, an adversary may cheat with probability $1 - \epsilon$, but risks detection with probability ϵ [Boyle et al., 2025, Sec 1].

Covert security is motivated by real-world scenarios where deterrence is often enough to prevent cheating. Examples include:

- Cloud services, where detection could harm the provider's reputation or user trust.
- Blockchain applications, where penalties (e.g., slashing of staked tokens) can deter misbehavior.

In the setting of memory checkers, this corresponds to relaxing the soundness requirement from negligible (standard) to constant. This has two main consequences:

1. It opens the door to potentially more efficient constructions (lower q, smaller p).
2. It may allow circumvention of existing lower bounds which require negligible soundness (e.g., those of Boyle et al. [2024]).

This motivates the central question of Boyle et al. [2025]: *Does relaxing soundness to the covert level allow one to bypass the* $\Omega(\log n/\log\log n)$ *query lower bound for efficient memory checkers?* Their work provides a definitive negative answer, proving that even under covert security, a logarithmic query overhead remains necessary in the computational setting.

4 Background and Definitions

To understand the contributions of Boyle et al. [2025], we first need to define the core concepts precisely, following their formalisms [Boy, Sec 3.1].

4.1 Memory Checkers Formalized

A memory checker simulates a Random Access Machine (RAM) for a user, operating on a *logical* memory space $[n]$ (where n is the number of logical addresses) with words of size w_l bits. The actual data resides in a *physical* memory managed by the untrusted server, with address space $[m]$ and word size w bits. The checker itself maintains a trusted *local state* st of size p bits between user operations.

We assume the checker operates in the *computational setting*, where the server (adversary) is modeled as a Probabilistic Polynomial-Time (PPT) machine [Boyle et al., 2025]. We also typically assume $w_l = 1$ (single-bit logical words) for simplicity in lower bounds, and that $m \leq \text{poly}(n)$ and $w \leq \text{polylog}(n)$ [Boy, Rem 1, Cor 1].

4.2 Security Properties: Completeness and Soundness

The correctness and security of a memory checker are defined by two properties:

Definition 4.1 (Completeness [Boy], Def 4)**.** *A memory checker has $c(n)$-completeness if, for any sequence of $l = poly(n)$ logical operations $I = \{(op_j, addr_j, data_j)\}_{j=1}^{l}$, when the checker interacts with an *honest* physical memory (correctly executing the protocol), the probability that all read operations return the correct value according to the ideal logical memory snapshot is at least $c(n)$. The probability is over the checker's internal randomness.*

Typically, completeness is required to be high, e.g., $c(n) = 1 - \text{negl}(n)$ or a constant close to 1 like $2/3$ or $99/100$ [Boyle et al., 2025]. Completeness can often be amplified via repetition [Boy, Rem 2].

Definition 4.2 (Soundness [Boy], Def 5)**.** *A memory checker has soundness error $s(n)$ if, for any sequence of $l = poly(n)$ logical operations I, and for any stateful PPT adversary $\mathcal{A}$ controlling the physical memory, the probability that the checker outputs an *incorrect* value (i.e., $data' \neq \perp$ and $data'$ is not the correct value according to the ideal logical memory snapshot for that read operation) for *any* read operation during the execution is at most $s(n)$. The probability is over the checker's and the adversary's randomness.*

The distinction crucial to Boyle et al. [2025] lies in the magnitude of $s(n)$:

- **Standard (Malicious) Soundness:** Requires negligible error, e.g., $s(n) \leq 1/\text{poly}(n)$ or often $s(n) \leq 1/n^{1+o(1)}$ as needed by Boyle et al. [2024]. This essentially forbids successful cheating.
- **Covert Security / Constant Soundness:** Requires only constant error, e.g., $s(n) \leq 1/3$. This allows the adversary to cheat successfully with probability $1 - s(n)$ but guarantees detection with probability $s(n)$ [Boyle et al., 2025].

4.3 Prior Lower Bounds

Understanding the context of Boyle et al. [2025] requires knowing the previous state-of-the-art lower bounds in the computational setting (summarized in [Boy, Fig 1]):

- **Information-Theoretic Bound [Blum et al., 1991]** For unbounded adversaries, $p{\cdot}q = \Omega(n)$, rendering efficient checkers impossible.
- **Dwork et al. [2009] Bound:** For computational adversaries, they showed $p \geq n/(\log n)^{O(q)}$ even for *constant* soundness. However, this bound only applies to highly restricted checkers that are *deterministic* (no internal randomness) and *non-adaptive* (physical query locations depend only on the logical operation type, not on prior responses or state). Most practical constructions use randomness and adaptivity [Boyle et al., 2025].
- **Boyle et al. [2024] Bound:** This was the breakthrough result, proving $p \geq n/(\log n)^{O(q)}$ for *any* computational memory checker (randomized, adaptive, crypto-based). This implies $q = \Omega(\log n/\log\log n)$ for $p \leq n^{1-\epsilon}$. The crucial limitation was the requirement of *negligible* soundness error $(s(n) \leq 1/n^{1+o(1)})$ [Boyle et al., 2024, 2025].

The gap was clear: Does the Boyle et al. [2024] bound hold if we only require the constant soundness of Dwork et al. [2009], but allow general (randomized, adaptive) checkers?

4.4 Covert Security

The covert security model [Afek and Lindell, 2010] provides a formal way to capture the notion of "security with deterrence." Instead of demanding that the adversary cannot cheat at all (as in standard malicious security with negligible soundness), it requires that any cheating attempt is detected with a noticeable, constant probability ϵ. In the memory checking context, this corresponds precisely to requiring constant soundness $s(n) \leq 1 - \epsilon$ (or often just $s(n) \leq \epsilon$, e.g., $s(n) \leq 1/3$) [Boyle et al., 2025].

The motivation is practical: in many scenarios, the risk of being caught is sufficient disincentive. Cloud providers fear reputational damage, and participants in blockchain systems might lose staked assets [Boy]. Furthermore, in other areas of secure computation, relaxing security to the covert model has yielded significant asymptotic efficiency improvements [Afek and Lindell, 2010]. Therefore, it was highly plausible that covert security might allow memory checkers to break the $\Omega(\log n/\log\log n)$ query barrier established by Boyle et al. [2024] for negligible soundness [Boyle et al., 2025].

5 Main Results of the Paper

The core achievement of Boyle et al. [2025] is to demonstrate that, contrary to expectations from other domains, relaxing to covert security does *not* enable asymptotically more efficient memory checkers in terms of query complexity, under a mild structural assumption on the checker.

5.1 The "Read-Only Reads" Assumption

The authors introduce a condition they call "read-only reads," which they argue is natural and satisfied by existing standard constructions like those based on Merkle trees [Blum et al., 1991, Papamanthou and Tamassia, 2011, Boyle et al., 2025].

Assumption 5.1 (Read-Only Reads [Boy], Def 1)**.** *A memory checker satisfies the "read-only reads" property if, during the execution of any logical read instruction from the user:*

(i) It only performs physical read *operations on the remote database (no physical writes are issued).*

(ii) It does not modify its own local storage state ($st' = st$).

The intuition is that a read operation should only involve fetching and verifying data, not altering the stored data or the checker's persistent state [Boy]. While one could devise contrived schemes violating this, it's unclear if doing so offers any benefit [Boyle et al., 2025].

5.2 Main Lower Bound Theorems

Under this assumption, Boyle et al. [2025] prove a lower bound that essentially matches the Boyle et al. [2024] bound, but holds even for constant soundness.

Theorem 5.2 (Informal Main Result [Boy], Thm 1)**.** *Every computational memory checker satisfying the read-only reads property (Assumption 5.1) for a logical memory of size n, with query complexity q, local storage p, completeness $2/3$, and constant soundness $1/3$, must satisfy:*

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

This immediately implies that low query complexity requires large local storage, and vice-versa. They also provide a more fine-grained theorem separating read and write complexities:

Theorem 5.3 (Formal Main Result [Boy], Thm 3)**.** *There exists a universal constant $C < 10^3$ such that the following holds. Consider a memory checker with read-only reads for a logical memory of size n (with $w_l = 1$). Let it have physical database size m, physical word size w, read query complexity $q_r < n/C$, write query complexity q_w, and local storage size p. If completeness is $99/100$ and soundness error is $1/3$, then for $n \geq C$:*

$$p \geq \frac{n}{(Cq_r q_w(w + \log m))^{C \cdot q_r}} - \log(1/q_r) - C$$

(Note: Completeness can be amplified from $2/3$ to $99/100$ via standard repetition with constant overhead [Boy, Proof of Cor 1]).

5.3 Implications and Corollaries

These theorems lead to direct consequences for the achievable efficiency of covertly secure memory checkers (assuming standard parameters $m \leq \text{poly}(n)$, $w \leq \text{polylog}(n)$):

Corollary 5.4 ([Boy], Cor 1)**.** *Under the conditions of Theorem 5.2 (with standard m, w), if the local storage is sub-polynomial ($p < n^{1-\epsilon}$ for some $\epsilon > 0$), then the overall query complexity must be $q = \Omega(\log n / \log \log n)$.*

This key corollary shows that the logarithmic barrier persists even for constant soundness, matching the bound previously known only for negligible soundness [Boyle et al., 2024].

Corollary 5.5 ([Boy], Cor 2)**.** *Under the conditions of Corollary 5.4, there is a sharp trade-off between read and write complexity:*

- *If read complexity is sub-logarithmic, $q_r = o(\log n / \log \log n)$, then write complexity must be super-logarithmic, $q_w = (\log n)^{\omega(1)}$.*

- *If read complexity is constant, $q_r = O(1)$, then write complexity must be polynomial, $q_w = n^{\Omega(1)}$.*

This highlights that achieving very fast reads (e.g., $O(1)$ queries) necessitates extremely slow writes, even with only covert security.

These results effectively place covertly secure memory checkers (with read-only reads) in the same complexity class as standard maliciously secure ones, distinguishing them from other secure computation tasks where covert security offers significant gains [Boyle et al., 2025]. The lower bound also applies to related primitives like PRAM checkers [?], locality bounds [?], and maintainable vector commitments [??] when adapted to require only constant soundness [Boy]. The landscape of computational lower bounds is summarized in Figure 1.

Figure 1: Summary of Computational Memory Checker Lower Bounds [Boy, Adapted from Fig 1]

Reference	Space/Query Tradeoff (p vs q)	Soundness	Limitation
Dwork et al. [2009]	$p \geq n/(\log n)^{O(q)}$	Constant	Deterministic & Non-adaptive
Boyle et al. [2024]	$p \geq n/(\log n)^{O(q)}$	Negligible ($1/n^{1+o(1)}$)	None
Boyle et al. [2025]	$p \geq n/(\log n)^{O(q)}$	Constant (1/3)	**Read-only reads**

6 Proof Overview and Key Ideas

The proof of the main lower bound (Theorem 5.3) builds significantly on the compression argument framework established by Boyle et al. [2024], but requires substantial modifications to function under the weaker assumption of constant soundness error [Boy, Sec 2].

6.1 The BKV Compression Framework (Recap)

The high-level strategy is to show that the checker's local state st must encode a significant amount of information about the logical memory contents. This is done via a communication complexity game [Boy, Sec 2]:

1. **Setup:** Alice wants to communicate a random string x (of fixed Hamming weight k) to Bob. They start with an initialized, empty memory checker (st_0, DB_0). Alice then writes '1's according to x, resulting in state st_1 and database DB_1.

2. **Communication:** Alice sends st_1 to Bob. In the Boyle et al. [2024] framework, she also sends information about a "heavy set" H of physical locations and the contents $DB_1|_H$.

3. **Decoding:** Bob constructs a *hybrid* database $\tilde{DB}$ using DB_1 on H and DB_0 elsewhere. Using the received state st_1 and $\tilde{DB}$, Bob performs logical reads on all n indices.

4. **Analysis using Completeness:** The construction of H and $\tilde{DB}$ is designed such that, by the checker's completeness property, Bob correctly recovers the value for *most* (e.g., $> 4n/5$) of the logical indices, even when using the hybrid $\tilde{DB}$ instead of the true DB_1 [Boy, Sec 2].

5. **Analysis using Soundness:** The checker's soundness property is invoked to argue that Bob receives *no incorrect* binary values during his decoding reads (only correct values or $\perp$).

6. **Lower Bound:** If Bob successfully recovers most of the memory content without errors, he learns significant information about x. Standard information-theoretic arguments (like Lemma

6.1, adapted from [Boy, Lem 1]) then imply a lower bound on the size of Alice's message. Since st_1 dominates the message size (if H is chosen carefully), this yields a lower bound on $p = |st_1|$.

Lemma 6.1 (Compression Lemma, e.g., [**?**], [Boy])**.** *If Alice sends a message of length L bits to Bob (using public coins), allowing Bob to recover a random input x chosen uniformly from a set S with probability δ, then $L \geq \log_2(|S|) - \log_2(1/\delta)$.*

6.2 The Challenge of Constant Soundness

The critical step where the Boyle et al. [2024] proof fails for constant soundness is step 5, invoking soundness [Boy, Sec 2]. For the soundness guarantee to apply, Bob's decoding procedure (reading all n indices using st_1 and $\tilde{DB}$) must correspond to an efficient (PPT) attack algorithm $\mathcal{A}$ against the checker.

However, the "optimal" heavy set H and a related parameter k (the Hamming weight Alice encodes) used in Bob's decoding depend on the exact probability distribution of physical accesses induced by st_1 and DB_1. This distribution might depend on the checker's internal randomness and the secret state st_1, which are unknown to the adversary $\mathcal{A}$ constructing the attack [Boy, Sec 2].

Boyle et al. [2024] handled this by having the adversary $\mathcal{A}$ essentially *guess* the correct k (out of $O(q)$ possibilities) and guess whether each physical location accessed during a read belongs to the true H (effectively guessing the relevant parts of $\tilde{DB}$). This guessing introduces a multiplicative factor of roughly $O(q_r \cdot 2^{q_r})$ into the soundness error probability achieved by the reduction [Boy, Sec 2].

- If the original checker had negligible soundness $s(n) = 1/n^{1+o(1)}$, this error blowup was acceptable, as $s'(n) = O(q_r 2^{q_r}) \cdot s(n)$ could still be negligible for $q_r = O(\log n/\log\log n)$.
- However, if the original checker only has constant soundness $s(n) = 1/3$, the resulting error $s'(n) = O(q_r 2^{q_r}) \cdot (1/3)$ becomes much larger than 1 for any non-trivial q_r, rendering the soundness guarantee useless in the reduction [Boy, Sec 2].

6.3 The Key Idea: Learning an Approximate Partition

The central innovation of Boyle et al. [2025] is to replace the adversary's *guessing* of the partition with *learning* an *approximate* partition [Boy, Sec 2.1]. They show two crucial things:

1. An approximate partition is sufficient for the compression argument to work. They define a notion of approximation that allows some physical locations to be slightly misclassified between Heavy, Medium, and Light sets, as long as a sufficient multiplicative gap between Heavy and Light probabilities is maintained [Boy, Thm 2].
2. Such an approximate partition (H, j, and derived k) can be learned efficiently by a PPT adversary.

The learning process, performed by the adversary $\mathcal{A}$ in the soundness reduction (see [Boy, Fig 6]), works as follows:

- After the initial writes encoding x are performed (resulting in DB_1, st_1), the adversary issues a large number, $t = \text{poly}(n)$, of additional logical reads to random indices $i \in [n]$.

- For each logical read, the adversary honestly interacts with the checker (using DB_1, st_1) and records one uniformly random physical location from the set $R(i, DB_1, st_1; r)$ accessed by the checker. This provides t samples from the "next-read" distribution D_{DB_1,st_1}.

- The adversary then runs an efficient algorithm called ‘Partition‘ [Boy, Sec 4, Thm 2]. This algorithm uses the collected samples to estimate the access probability $\hat{p}_j$ for each physical location $j \in [m]$. It then constructs "cushioned" histogram buckets $\{\hat{C}_i\}$ based on these estimated probabilities. Using Chernoff bounds, they show that these efficiently computable buckets reliably contain the "true" buckets $\{B_i\}$ defined by the underlying (unknown) probabilities p_j. By finding a cushioned bucket $\hat{C}_{i^*}$ with low total estimated probability, the algorithm derives the approximate partition H, M, L and the index j^* (which determines $k = k_{j^*}$).

- This efficiently computed H and k are then used by the adversary to construct the hybrid database $\tilde{DB}$ and perform the final n logical reads for the soundness attack.

Crucially, this learning process avoids the exponential 2^{q_r} blowup associated with guessing, allowing the reduction to work even when the original soundness $s(n)$ is constant [Boyle et al., 2025]. The adversary described in Figure 6 of the paper implements precisely this learning strategy before performing the attack reads [Boy].

6.4 The Role of the "Read-Only Reads" Assumption

The "read-only reads" assumption (Assumption 5.1) is technically required to ensure the validity of Bob's decoding process (step 3) and the corresponding soundness analysis (step 5) [Boy, Sec 2].

Consider Bob performing his n sequential logical reads using state st_1 and the hybrid database $\tilde{DB}$.

- If reads could modify the local state st_1 or write to the physical database, the checker's state and the database content relevant for *subsequent* reads could change during the decoding process.

- A read to index i might rely on data fetched during a previous read to index i'. If the read to i' used the hybrid $\tilde{DB}$ (which differs from the "correct" DB_1), it might have returned $\perp$ or incorrect data, potentially corrupting st_1 or writing bad data to $\tilde{DB}$ if reads were not read-only.

- This corruption could then cause the subsequent read to index i to fail completeness, even if it wouldn't have failed using the original st_1 and DB_1.

The Boyle et al. [2024] proof avoided this by having Bob effectively "rewind" the state between reads (a non-efficient process suitable only for the analysis), which contributed to the soundness loss factor [Boy, Sec 2].

The "read-only reads" assumption allows Bob (and the soundness adversary) to perform all n logical reads *sequentially* using the *same* initial state st_1 and the *same* hybrid database $\tilde{DB}$. Since reads don't change st_1 or $\tilde{DB}$, the conditions for applying completeness and soundness remain stable throughout the decoding process. Specifically, Claim 9 in the proof relies on this property to relate Bob's outcomes z'_i to the ideal outcomes z_i via the completeness guarantee [Boy, Proof of Thm 3, Claim 9]. This allows the simpler sequential decoding strategy ("route (b)" in their terminology) to work for the analysis, avoiding the pitfalls of rewinding ("route (a)") that break constant soundness [Boy, Sec 2].

6.5 Computing a Partition

We now present an effective version of the partition lemma of Boyle et al. [2024], which is computationally efficient when one can generate samples from random logical reads to the database. This algorithm, called Partition, will be used in the lower bound proof to argue that an efficient adversary can compute a partition and perform a replay attack accordingly.

6.5.1 Theorem Statement and Discussion

We briefly discuss the differences between the Theorem 2 and Boyle et al. [2024]'s Lemma 3. Notably, Item 3 here shows a probability bound of $4/(c-1)$ for hitting M instead of $1/c$, and Item 4 has $2c$ in the exponent in the denominator instead of c; these constant factor differences are not essential for our application.

Theorem 6.2 (Effective version of Lemma 3 of Boyle et al. [2024])**.** *There is a polynomial-time algorithm* $\textsc{Partition}_D(1^\gamma, 1^c, 1^n, m)$ *with sample access to a distribution* D *with the following guarantee. Suppose* $\gamma, c, n \in \mathbb{N}$ *with* $\gamma, c \geq 2$*, and suppose* D *is a distribution over* $[m]$ *where* $m \leq 2^{\text{poly}(n)}$*. Then, with probability at least* $1 - 1/n$*,* Partition *outputs some* $H \subseteq [m]$ *and* $i \in [c]$ *such that there exist* $L, M \subseteq [m]$ *with:*

1. $[m] = H \sqcup M \sqcup L$ *(disjoint union);*
2. *For all* $\ell \in L$, $\Pr_{X\sim D}[X = \ell] \leq \frac{(2\gamma)^{2i-2}}{n}$*;*
3. $\Pr_{X\sim D}[X \in M] \leq \frac{4}{c-1}$*;*
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* D *and is deterministic (ignoring the randomness in the samples from* D*).*

6.5.2 Algorithm and Intuition

The algorithm proceeds by estimating the probabilities $p_j = \Pr_{X\sim D}[X = j]$ for each $j \in [m]$ via sampling, then grouping indices into "buckets" according to these estimated probabilities, and finally constructing "cushioned" buckets that overlap but are guaranteed to cover the true buckets with high probability.

6.5.3 Proof of Theorem 6.2

We now prove the correctness of the algorithm by establishing a series of claims.

Claim 6.1. *For the cushioned buckets* C_i *defined above, with probability at least* $1 - 1/n$ *over the samples, for all* $i \in [c]$*, the true bucket* $B_i := A_{2i-1} \cup A_{2i}$ *satisfies* $B_i \subseteq C_i$*.*

Proof. We consider three cases: $i \in \{2, \ldots, c-1\}$, $i = c$, and $i = 1$.

Case 1: $i \in \{2, \ldots, c-1\}$**.** Here,

$$C_i = A_{2i-2} \cup A_{2i-1} \cup A_{2i} \cup A_{2i+1} = \left\{ j \in [m] : p_j \in \left(\frac{(2\gamma)^{2i-3}}{n}, \frac{(2\gamma)^{2i+1}}{n} \right] \right\}$$

Algorithm Partition$_D(1^\gamma, 1^c, 1^n, m)$**:**

1. Let $t := 100n^2 \ln(m+n)$. Generate t samples $a_1, \ldots, a_t \sim D$.
2. For $j \in [m]$, let $p_j := \frac{|\{k \in [t] : a_k = j\}|}{t}$. Note that $\sum_{j \in [m]} p_j = 1$.
3. Define the buckets:
$$\begin{aligned} A_1 &:= \left\{ j \in [m] : p_j \in \left[0, \frac{2\gamma}{n}\right) \right\}, \\ A_i &:= \left\{ j \in [m] : p_j \in \left(\frac{(2\gamma)^{i-1}}{n}, \frac{(2\gamma)^i}{n} \right] \right\} \quad \text{for } i \in \{2, \ldots, 2c-1\}, \\ A_{2c} &:= \left\{ j \in [m] : p_j \in \left(\frac{(2\gamma)^{2c-1}}{n}, \infty \right) \right\}. \end{aligned}$$
These buckets partition $[m]$.
4. Define the "cushioned" buckets:
$$\begin{aligned} C_1 &:= A_1 \cup A_2 \cup A_3, \\ C_i &:= A_{2i-2} \cup A_{2i-1} \cup A_{2i} \cup A_{2i+1} \quad \text{for } i \in \{2, \ldots, c-1\}, \\ C_c &:= A_{2c-2} \cup A_{2c-1} \cup A_{2c}. \end{aligned}$$
All elements in A_1 and A_{2c} are covered once, all others are covered at most twice.
5. Fix some $i^* \in \arg\min_{i \in \{2, \ldots, c\}} \sum_{j \in C_i} p_j$.
6. Set $H := \bigcup_{i \geq i^*+1} C_i$.
7. Output (H, i^*).

Figure 2: The Partition algorithm as in Theorem 6.2.

and
$$B_i = A_{2i-1} \cup A_{2i} = \left\{ j \in [m] : p_j \in \left(\frac{(2\gamma)^{2i-2}}{n}, \frac{(2\gamma)^{2i}}{n} \right] \right\}.$$
By the multiplicative Chernoff bound, for all $j \in B_i$ (where $p_j \geq (2\gamma)^{2i-2}/n \geq 1/n$), we have
$$\Pr\left[p_j \notin \left(\frac{p_j}{2}, \frac{3p_j}{2} \right) \right] \leq 2e^{-p_j t/12} \leq 2e^{-t/(12n)}.$$
Since $t \geq 100n \ln(m+n)$, union bounding over all $j \in [m]$ gives total failure probability at most $1/(3n)$. Thus, with high probability, $B_i \subseteq C_i$ for all $i \in \{2, \ldots, c-1\}$.

Case 2: $i = c$**.** Here,
$$C_c = A_{2c-2} \cup A_{2c-1} \cup A_{2c} = \left\{ j \in [m] : p_j \in \left(\frac{(2\gamma)^{2c-3}}{n}, \infty \right) \right\},$$
and
$$B_c = A_{2c-1} \cup A_{2c} = \left\{ j \in [m] : p_j \in \left(\frac{(2\gamma)^{2c-2}}{n}, \infty \right) \right\}.$$

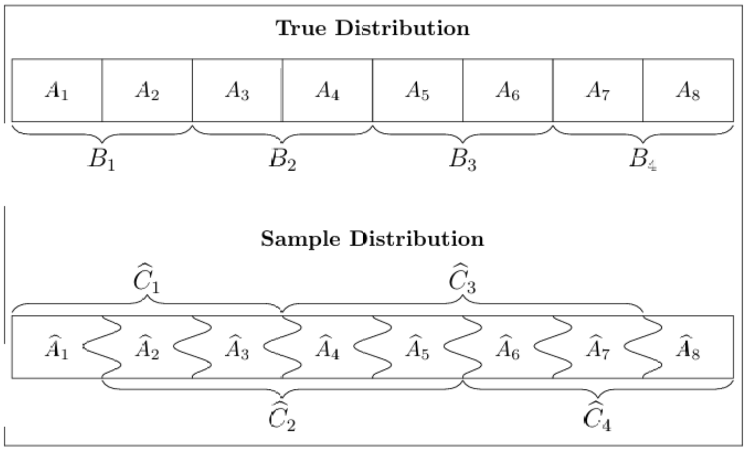

Figure 3: Graphical sketch of "cushioned buckets" C_i in Theorem 6.2. Each B_i (true bucket) is contained in C_i with high probability. Each A_i is contained in at most two C_i, so $\{C_i\}$ form at most a double cover.

The same Chernoff bound argument applies, so $B_c \subseteq C_c$ with high probability.

Case 3: $i = 1$**.** Here,

$$C_1 = A_1 \cup A_2 \cup A_3 = \left\{ j \in [m] : p_j \in \left[0, \frac{(2\gamma)^3}{n}\right) \right\},$$

and

$$B_1 = A_1 \cup A_2 = \left\{ j \in [m] : p_j \in \left[0, \frac{(2\gamma)^2}{n}\right) \right\}.$$

For small p_j, we use the additive Chernoff bound:

$$\Pr[p_j \geq p_j + 1/n] \leq e^{-2t/n^2}.$$

Since $t \geq 2n^2 \ln(m+n)$, union bounding over all $j \in [m]$ gives failure probability at most $1/(3n)$. Thus, with high probability, $B_1 \subseteq C_1$.

Combining all cases, the total failure probability is at most $1/n$. ☐

Claim 6.2. $\sum_{j \in C_{i^*}} p_j \leq \frac{2}{c-1}$.

Proof. The C_i form at most a double cover of $[m]$. Let $\alpha_i := \sum_{j \in C_i} p_j$. By construction,

$$\sum_{i=2}^{c} \alpha_i \leq \sum_{i=1}^{c} \alpha_i = \sum_{i=1}^{c} \sum_{j \in C_i} p_j \leq 2 \sum_{j \in [m]} p_j = 2.$$

By averaging, there exists $i^* \in \{2, \ldots, c\}$ with $\alpha_{i^*} \leq 2/(c-1)$, as desired. ☐

Now, define $M := B_{i^*} \setminus H$ and $L := \bigcup_{i < i^*-1} B_i$.

Claim 6.3 (Item 1)**.** *H, M, and L partition* $[m]$.

Proof. By construction, B_i partition $[m]$. M and H are disjoint by definition. L and M are disjoint since B_i are disjoint. L and H are disjoint because L is a union of B_i for $i < i^* - 1$ and H is a union for $i \geq i^* + 1$. Their union covers $[m]$. $\square$

Claim 6.4 (Item 2)**.** *For all $\ell \in L$, $\Pr_{X\sim D}[X = \ell] \leq (2\gamma)^{2i^*-2}/n$.*

Proof. By definition, $L = \bigcup_{i<i^*-1} B_i = \bigcup_{i<2i^*-2} A_i = \{j \in [m] : p_j \leq (2\gamma)^{2i^*-2}/n\}$. $\square$

Claim 6.5 (Item 3)**.** $\Pr_{X\sim D}[X \in M] \leq 4/(c-1)$.

Proof. By Claim 6.1, $M \subseteq C_{i^*}$, so

$$\Pr_{X\sim D}[X \in M] \leq \sum_{j\in C_{i^*}} p_j \leq \frac{2}{c-1}.$$

Accounting for estimation error (doubling), we get $4/(c-1)$. $\square$

Claim 6.6 (Item 4)**.** *The set H satisfies*

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

Proof. For all $h \in H$, $p_h \geq (2\gamma)^{2i^*-1}/n$. Thus,

$$|H| \cdot \frac{(2\gamma)^{2i^*-1}}{n} \leq \sum_{h\in H} p_h \leq 1,$$

so $|H| \leq n/(2\gamma)^{2i^*-1}$. Therefore,

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

as desired. $\square$

This completes the proof of Theorem 6.2.

6.6 Proof of Theorem 3

We now prove the main theorem of this paper, Theorem 3. The proof combines the compression argument with the efficient partitioning technique from Theorem 2, enabling us to invoke memory checking soundness against an efficient adversary.

Notation and Setup. Let $DB \in (\{0,1\}^w \cup \{\perp\})^m$ be the public database, and $st \in \{0,1\}^p$ the (secret) local storage. For $i \in [n]$, let $R(i, DB, st; r) \subseteq [m]$ denote the set (of at most q_r) physical locations queried by the checker when performing a logical read to index i, using DB, st, and internal randomness r.

Define the *next-read distribution* $D_{DB,st}$ as follows:

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

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

Communication Protocol. We construct a public-coin protocol for Alice and Bob to send a string x chosen uniformly at random from $\{0,1\}^{9n/10+k_{j^*}}$ with Hamming weight k_{j^*}. This follows Boyle et al. [2024], with differences in the soundness argument.

Protocol Steps:

1. Alice and Bob share randomness. Both initialize the memory checker and write 0 to all logical indices $i \in [n]$.
2. Sample $j^* \sim [c]$ uniformly at random.
3. Sample $y \in \{0,1\}^n$ with Hamming weight $n/10 - k_{j^*}$, and write 1 to all indices i with $y_i = 1$ in random order.
4. Let DB_0, st_0 be the resulting public database and local storage.
5. Alice and Bob agree on a bijection $\pi : [9n/10 + k_{j^*}] \to [n]$ mapping to the indices with $y_i = 0$. For $k := k_{j^*}$, define $\pi(x) := \{\pi(i) : i \in [9n/10 + k], x_i = 1\} \subseteq [n]$.

Alice's Encoding. Alice encodes $x_{(-j^*)} = (x_1, \ldots, x_{j^*-1}, x_{j^*+1}, \ldots, x_c)$. For $x := x_{j^*}$, she writes 1 to all indices in $\pi(x)$ using DB_0 and st_0 in random order, resulting in DB_1, st_1. She runs $\textsc{Partition}_{D_{DB_1,st_1}}$ to get (H, j). Alice sends st_1, H, $DB_1|_H$, and auxiliary info aux (see Figure 4 in the original proof) to Bob.

The message length is bounded by:

$$\log\left(\prod_{j\in[c],j\neq j^*}\binom{9n/10+k_j}{k_j}\right) + |st_1| + |H|\lceil\log m\rceil + |H|\lceil\log(2w+1)\rceil + |\text{aux}|$$

Bob's Decoding. Bob decodes $x_{(-j^*)}$, reconstructs the hybrid database DB (using DB_1 on H and DB_0 elsewhere), and performs logical reads to all $i \in [n]$ with st_1 and DB. Let $\tilde{z}_i \in \{0,1,\perp\}$ be the result. Bob fills in $\perp$ values using aux.

Analysis. Let $z \in \{0,1\}^n$ be the logical memory after writing y and x; $z_i = y_i \vee 1[i \in \pi(x)]$ (so $\|z\|_0 = n/10$). Let ρ be the randomness for the order of writes.

Claim 6.7. *The distribution of j^* is independent of z and ρ. In particular,* $\Pr[j = j^*] = 1/c$.

Completeness. Assume the call to $\textsc{Partition}$ succeeds (probability at least $1 - 1/n$). Let $D = D_{DB_1,st_1}$ and $(H \sqcup M \sqcup L, j)$ be the output of $\textsc{Partition}$. Then:

1. For all $\ell \in L$, $\Pr_{X\sim D}[X = \ell] \leq \delta_j$.
2. $\Pr_{X\sim D}[X \in M] \leq 4/(c-1) = 1/(25q_r)$.
3. $\frac{1}{\delta_j} - |H|\gamma > n/(2\gamma)^{2c}$.

Let W be the set of physical locations written by Alice (at most kq_w). Let $\text{BAD} := W \cap (L \cup M)$; DB_1 and DB differ only at BAD.

Claim 6.8. *Assuming $j = j^*$,*

$$\Pr_{i,r}[R(i, DB_1, st_1; r) \cap BAD = \emptyset] \geq \frac{19}{20}.$$

Proof. We decompose $\mathrm{BAD} = (W \cap L) \cup (W \cap M)$.

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

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

Thus,

$$\Pr_{i,r}[R(i, DB_1, st_1; r) \cap (W \cap L) \neq \emptyset] \leq q_r \cdot k_{j^*} q_w \delta_{j^*} \leq \frac{1}{100}$$

by our choice of k_{j^*}.

For $W \cap M$, $\Pr_{X \sim D}[X \in M] \leq 1/(25q_r)$, so

$$\Pr_{i,r}[R(i, DB_1, st_1; r) \cap (W \cap M) \neq \emptyset] \leq q_r \cdot \frac{1}{25q_r} = \frac{1}{25}.$$

By union bound, the total is at most $1/100 + 1/25 = 1/20$. □

Claim 6.9.

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

Proof. By completeness, for each i,

$$\Pr_r[\tilde{z}_i = z_i \mid R(i, DB_1, st_1; r) \cap \mathrm{BAD} = \emptyset] \geq \frac{99}{100}.$$

By the previous claim, the event $R(i, DB_1, st_1; r) \cap \mathrm{BAD} = \emptyset$ holds with probability at least 19/20. Thus,

$$\mathbb{E}_{i,r}\left[\mathbf{1}_{\tilde{z}_i = z_i}\right] \geq \frac{99}{100} \cdot \frac{19}{20} > \frac{47}{50}.$$

By Markov's inequality, with probability at least 7/10, at least $4n/5$ of the $\tilde{z}_i$ equal z_i. □

Soundness. We now describe the efficient adversary for soundness: after the writes, the adversary samples $t = \Theta(n^2 \log(m+n))$ random logical reads, records a random physical location from each, runs Partition to compute (H, j), computes $k := k_j$, reconstructs DB as above, and performs logical reads to all $i \in [n]$. By the read-only reads property, Bob's and the adversary's views coincide. By soundness,

$$\Pr[\exists i \in [n], \tilde{z}_i \notin \{z_i, \perp\} \mid j = j^*] \leq \frac{1}{3},$$

so

$$\Pr[\forall i \in [n], \tilde{z}_i \in \{z_i, \perp\} \mid j = j^*] \geq \frac{2}{3}.$$

Combining with the previous claim and the fact that $\Pr[j = j^*] = 1/c$, we have (by union bound):

$$\Pr\left(\forall i \in [n], \tilde{z}_i \in \{z_i, \perp\} \wedge |\{i : \tilde{z}_i = z_i\}| \geq \frac{4n}{5} \wedge j = j^*\right) \geq \frac{1}{3c} \geq \frac{1}{303q_r}$$

(assuming $c = 100q_r + 1$).

Compression Bound. If all the above conditions hold, Bob successfully recovers Alice's input. The auxiliary string aux encodes a subset of the $n/5$ remaining indices of size at most k ($k \leq n/100$), so $|\text{aux}| \leq \log \binom{n/5}{k} + 1$.

Thus, by the compression lemma, the total message length satisfies

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

where $\alpha \geq 1/(800q_r)$ is the protocol's success probability.

Applying standard binomial coefficient bounds and simplifying,

$$p \geq \frac{n}{(600q_r q_w(w + \log m))^{406q_r}} - \log(q_r) - 13,$$

as desired.

7 Open Challenges and Future Directions

While Boyle et al. [2025] significantly advance our understanding of memory checking complexity under relaxed security, their work also highlights important open questions.

7.1 Removing the "Read-Only Reads" Assumption

The most prominent open challenge explicitly mentioned by the authors is to prove the lower bound (Theorem 5.2 / 5.3) *without* relying on the "read-only reads" assumption [Boy, Sec 1]. While they argue the assumption is natural and holds for known constructions, removing it would make the lower bound fully general for all conceivable memory checkers, matching the generality of the Boyle et al. [2024] result (albeit for constant soundness).

Difficulty: As discussed in Section 6, the assumption is currently crucial for ensuring that Bob's sequential decoding process doesn't invalidate the completeness/soundness conditions for later reads due to state or database modifications made by earlier reads using the hybrid database. Circumventing this seems non-trivial.

Potential Ideas(Speculative):

- **Alternative Compression/Communication Game:** Could a different communication scenario be devised where the interaction structure naturally avoids the issue of state corruption during decoding? Perhaps a game not based on sequential reads of the entire memory?
- **More Sophisticated Hybrid Database/State Construction:** Is it possible for Alice to send slightly more information, or for Bob to construct a more complex hybrid state/DB, that allows simulating the reads correctly even if the checker modifies state/DB during reads? This seems difficult without significantly increasing communication.
- **Stronger Analysis of State Changes:** Perhaps a very careful analysis could bound the impact of state/DB changes during reads, showing they don't fundamentally break the information flow needed for the compression argument, even if they complicate the analysis? This might involve tracking potential corruptions and showing they affect only a small fraction of reads.
- **Different Lower Bound Technique:** Are there entirely different techniques (beyond compression arguments based on this specific communication game) that could yield the same lower bound without being sensitive to the read-only reads property?

Addressing this challenge would likely require new technical insights into how information is encoded and retrieved in memory checkers that actively modify state during read operations.

7.2 Soundness Amplification for Memory Checkers

The paper also touches upon the general difficulty of amplifying soundness for **online** memory checkers [Boyle et al., 2025, Sec. 1]. Standard parallel repetition techniques, which work well for offline tasks, run into issues in the online, interactive setting: simply running multiple instances in parallel and taking a majority vote doesn't necessarily reduce the soundness error significantly against adaptive adversaries. Furthermore, even if amplification were possible, achieving negligible soundness from constant soundness would likely require $O(\log n)$ repetitions, which would increase both p and q by a logarithmic factor, potentially weakening the resulting lower bounds derived from the negligible soundness setting [Boyle et al., 2025].

Developing efficient and provably secure soundness amplification techniques specifically tailored for online memory checkers remains an interesting challenge. Success here could potentially bridge the gap between constant-soundness and negligible-soundness regimes in different ways, possibly impacting upper bounds (constructions) more than lower bounds.

Some Ideas:

Selective Sequential Repetition with Consistency Checks. Run k sequential executions of the checker, but intersperse *cross-instance consistency proofs* between them. After each execution, the checker publishes a small commitment to its updated local state (e.g., a Merkle-tree root of st), and the server must present a succinct proof (via a Merkle-proof or lightweight SNARK) that it responded consistently with that committed state. An adversary that cheats in one repetition risks detection both within that instance and in the consistency check with the next. By choosing $k = O(\log \log n)$, one can drive the soundness error down to $1/(n)$ while incurring only a doubly-logarithmic overhead in p and q.

Fingerprinting with Randomized Challenges. Adapt Freivalds' fingerprinting idea: between logical operations, the checker issues a *fresh* random linear combination of all prior read–write transcripts and challenges the server to prove consistency under that combination. Since the adversary cannot anticipate the random coefficients, any deviation is caught with high probability. Batching $\Theta(\log \log n)$ such fingerprint checks into a single round can amplify soundness to negligible with only an $O(\log \log n)$ factor increase in communication and storage.

Hybrid SNARK-Based Proofs. Leverage succinct non-interactive arguments (SNARKs) by requiring the server to produce a *single* proof attesting to the correctness of all t logical operations so far. Recent "incremental" SNARK constructions compress proofs to $O(1)$ group elements per operation and allow verification in $O(\log t)$ time [**?**]. Integrating such proofs into the memory-checker protocol can reduce soundness error to negligible while limiting the p, q overhead to $O(\log n)$ or better.

Each of these approaches trades added cryptographic machinery against communication and storage cost. Designing a provably secure amplification method that keeps $p, q = o(\log n)$ remains an open problem of both theoretical and practical importance.

7.3 Exploring Other Relaxations

Covert security is one natural relaxation of standard malicious security. Are there other meaningful security models for memory checking (perhaps related to amortized guarantees, different adversarial capabilities, or specific application contexts) where the logarithmic query complexity barrier might be broken? Exploring the efficiency landscape under various security definitions continues to be a relevant direction.

Some Ideas:

Amortized Soundness. Instead of bounding the failure probability per operation, allow a *total* error budget over a sequence of T operations. In applications where occasional undetected errors are tolerable (e.g., streaming data), targeting an aggregate failure probability ϵ_{total} could reduce per-operation p and q by a factor of T.

Honest-but-Curious (Semi-Honest) Adversaries. Assuming the server follows the protocol but may inspect transcripts, one can drop soundness requirements and focus on *privacy*. Classical ORAM constructions then achieve $O(\log n)$ overhead with only statistical security [**?**], suggesting that purely privacy-oriented checkers could be significantly more efficient.

Bounded Adversarial Capabilities. Restricting the adversary's computational power (e.g., to small space or streaming algorithms) may allow checkers based on weaker hardness assumptions. For instance, if the server cannot invert certain hash functions within bounded space, an ORAM-style construction could achieve sublogarithmic q with only constant local state.

Hardware-Assisted Models. Introducing a small secure enclave (e.g., Intel SGX) or a tamper-resistant module on the server side can simplify memory checking dramatically. A trusted hardware root can perform integrity checks in $O(1)$ time per operation, reducing both p and q to practical constants. The trade-off is reliance on physical assumptions and potential side-channel vulnerabilities.

Exploring these alternative models may reveal new trade-offs between security guarantees and efficiency, and could even break the logarithmic-query barrier under suitably constrained adversarial or failure models.“‘

8 Conclusion

The work of Boyle et al. [2025], Boy provides a compelling answer to the question of whether relaxing security guarantees can lead to more efficient online memory checking. By adapting the lower bound techniques of Boyle et al. [2024], they demonstrate that the $\Omega(\log n / \log\log n)$ query complexity lower bound for computationally secure memory checkers with sub-polynomial local storage is surprisingly robust. It holds not only for the standard setting of negligible soundness error but also persists even when the guarantee is weakened to constant-probability covert security, provided the checker satisfies the natural "read-only reads" property.

Their key technical contribution lies in showing how an adversary in the soundness reduction can efficiently *learn* an approximate partition of the physical memory by sampling the checker's read behavior, thereby circumventing the exponential soundness loss incurred by previous guessing-based approaches which failed under constant soundness.

This result has significant implications. It suggests that the logarithmic overhead associated with verifying outsourced memory online is largely inherent, regardless of whether perfect prevention of cheating or merely probabilistic deterrence is required. This distinguishes memory checking from other cryptographic tasks where covert security enables substantial asymptotic efficiency gains. The findings reinforce the trade-offs established by Boyle et al. [2024] and provide a nearly complete picture of the complexity landscape for this important primitive, solidifying the optimality of known constructions (up to constant factors).

The primary remaining open question is the necessity of the "read-only reads" assumption. Proving the lower bound without this condition would fully generalize the result to all possible memory checker designs. Nonetheless, the work of Boyle et al. [2025] represents a significant step forward in understanding the fundamental limits of secure outsourced storage verification.

References

Yehuda Afek and Yehuda Lindell. Covert security with public verifiability: Faster, leaner, and simpler. In *Advances in Cryptology – EUROCRYPT 2010*, pages 1–20, 2010. Introduced covert security model. As cited in [Boyle et al., 2025].

Manuel Blum, William Evans, Peter Gemmell, Sampath Kannan, and Moni Naor. Checking the correctness of memories. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science (FOCS '91)*, pages 90–99, 1991. Introduced memory checking. As cited in [Boyle et al., 2025].

Elette Boyle, Ilan Komargodski, and Neekon Vafa. Lower bounds on memory checkers. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing (STOC)*, pages 107:1–107:14, 2024. As cited in [Boyle et al., 2025]. Established lower bound for negligible soundness.

Elette Boyle, Ilan Komargodski, and Neekon Vafa. The complexity of memory checking with covert security, 2025.

Cynthia Dwork, Moni Naor, Omer Reingold, Guy N. Rothblum, and Salil Vadhan. On the complexity of verifiable delegation. In *Theory of Cryptography Conference (TCC)*, pages 1–18, 2009. Lower bound for deterministic/non-adaptive checkers. As cited in [Boyle et al., 2025].

Ralph C. Merkle. A certified digital signature. In *Advances in Cryptology – CRYPTO '89 Proceedings*, pages 218–238, 1989. Merkle Trees. As cited in [Boyle et al., 2025].

Charalampos Papamanthou and Roberto Tamassia. Optimal and parallel online memory checking. In *Cryptology ePrint Archive, Report 2011/620*, 2011. Refined Merkle-tree based construction. As cited in [Boyle et al., 2025].