Limits of Breach-Resistant and Snapshot-Oblivious RAMs

Giuseppe Persiano[*] Kevin Yeo[†]

Abstract

Oblivious RAMs (ORAMs) are an important cryptographic primitive that enable outsourcing data to a potentially untrusted server while hiding patterns of access to the data. ORAMs provide strong guarantees even in the face of a *persistent adversary* that views the transcripts of all operations and resulting memory contents. Unfortunately, the strong guarantees against persistent adversaries comes at the cost of efficiency as ORAMs are known to require $\Omega(\log n)$ overhead.

In an attempt to obtain faster constructions, prior works considered security against *snapshot adversaries* that only have limited access to operational transcripts and memory. We consider (s, ℓ)-snapshot adversaries that perform s data breaches and views the transcripts of ℓ total queries. Promisingly, Du, Genkin and Grubbs [Crypto'22] presented an ORAM construction with $O(\log \ell)$ overhead protecting against $(1, \ell)$-snapshot adversaries with the transcript of ℓ consecutive operations from a single breach. For small values of ℓ, this outperforms standard ORAMs.

In this work, we tackle whether it is possible to further push this construction beyond a single breach. Unfortunately, we show that protecting against even slightly stronger snapshot adversaries becomes difficult. As our main result, we present a $\Omega(\log n)$ lower bound for any ORAM protecting against a $(3, 1)$-snapshot adversary that performs three breaches and sees the transcript of only one query. In other words, our lower bound holds even if an adversary observes only memory contents during two breaches while managing to view the transcript of only one query in the other breach. Therefore, we surprisingly show that protecting against a snapshot adversary with three data breaches is as difficult as protecting against a persistent adversary.

[*]Università di Salerno and Google, `giuper@gmail.com`
[†]Google and Columbia University, `kwlyeo@google.com`

Contents

1 Introduction

In this work, we study the setting where a client wishes to outsource the storage of some data to a third-party service provider that we refer to as the server. For example, this is a natural problem that arises when users rely on cloud computing or cloud storage services. In terms of privacy, the client desires to keep all the data private even after outsourcing the data to the server. A straightforward attempt to secure the data is to require the client to encrypt all data before uploading to the server. This guarantees that the plaintext data may never be observed by anyone except the client. However, an adversary may still be able to observe patterns of access to the encrypted data. Prior works [IKK12, CGPR15, ZKP16, KKNO16, LMP18, GLMP19, BKM20, KPT21] have shown that access patterns may be utilized to reveal parts of the plaintext data in certain settings. Therefore, it is also important to also hide access patterns to the encrypted data.

Persistent Adversaries. Oblivious RAMs (ORAMs), introduced by Goldreich and Ostrovsky [GO96], are one cryptographic primitive that may be used to hide access patterns. At a high level, ORAMs guarantee that any adversary cannot distinguish between the access patterns to encrypted data incurred by any two equal-length operational sequences. These obliviousness guarantees hold even if the adversary has persistently compromised the server (for example, this may be the case if the adversary is the server operator). In this case, the *persistent adversary* observes server memory contents and all access to server memory (i.e., the operational transcript) during the entire execution of the ORAM.

A long line of work (such as [GMOT12, KLO12, SvS$^+$13, RFK$^+$15, ZWR$^+$16, BCP16, CLT16, PPRY18]) has studied the best efficiency that is achievable by an ORAM leading to the best construction obtaining $O(\log n)$ overhead [AKL$^+$20]. Recently, $\Omega(\log n)$ lower bounds [LN18] were proven showing the best ORAM constructions are optimal in the presence of persistent adversaries. Unfortunately, recent works have also shown that $\Omega(\log n)$ overhead is required to protect against persistent adversaries even with weaker privacy guarantees including differential privacy [PY19, PY23], searchable encryption leakage [PPY20], multiple non-colluding servers [LSY20] and hidden operational boundaries [HKKS19]. The same lower bounds have also been extended to other settings including small blocks [KL21].

Snapshot Adversaries. In an attempt to obtain faster constructions, prior works have considered weaker adversaries that model real-world scenarios. In most cases, users trust the service provider (server) to store data securely. However, attackers may try to breach these services to temporarily gain access to the system. During this time, adversaries may quickly download all data stored on the server or potentially even view transcripts of operations while being performed by the server. Eventually, the server operators will detect the system breach and remedy the situation to revoke access from the adversary. Recent reports [dbi] show that the above is a common theme in most real system attacks and remediation typically occurs within a couple of days after the attack starts.

To model the above, we introduce the notion of (s, ℓ)-snapshot adversaries where the adversary may breach the system at most s times. During each of these s breaches, the adversary receives a snapshot of the server memory and sees the transcript for a total of at most ℓ operations. When protecting against weaker snapshot adversaries (with small values of s and ℓ), we can hope to obtain sub-logarithmic efficiency for ORAMs. Prior works have studied constructions against such adversaries. Amjad, Kamara and Moataz [AKM19] considered breach-resistant data structures that were secure against $(s, 0)$-snapshot adversaries; these are adversary that only observed server memory contents and no operational transcripts. Unfortunately, their constructions still required

$O(\log n)$ overhead in the worst case (see Section 6.2 for more details).

A recent work of Du, Genkin and Grubbs [DGG22] presented an ORAM construction secure against $(1, \ell)$-snapshot adversaries that required overhead of only $O(\log \ell)$. For small values of ℓ, this is significantly faster than the $O(\log n)$ overhead ORAMs that protect against persistent adversaries. It was also shown that $\Omega(\log \ell)$ overhead is necessary for ORAMs protecting against $(1, \ell)$-snapshot adversaries as such constructions can be used to construct ORAMs against persistent adversaries by setting ℓ to be an upper bound on the length of the operational sequence such as $\ell = n^{O(1)}$.

However, this construction only protects against a single breach of the system ($s = 1$) leading to the following natural question that was also posed as an open problem in [DGG22]:

Is it possible to build sub-logarithmic overhead ORAMs that are secure against snapshot adversaries that may perform multiple breaches?

This is an important question as many real-world systems must run for long periods of time and may be compromised multiple times due to different attacks. For example, at least six large companies have been compromised by at least three data breaches in the past two decades (see [bre]). In this paper, we tackle this open problem to answer whether it is possible to construct efficient ORAMs to protect against multiple data breaches.

1.1 Our Contributions

As our main result, we answer the open problem in the negative. In particular, we show that there is a $\Omega(\log n)$ lower bound for ORAMs protecting against $(3, 1)$-snapshot adversaries that perform only three breaches and view the transcript for one operation. Afterwards, we explore various settings where we can circumvent the lower bound and present sub-logarithmic constructions for some weaker primitives.

Lower Bounds for $(3, 1)$-Snapshot Oblivious RAMs. We begin by presenting our lower bound for ORAMs that protect against $(3, 1)$-snapshot adversaries. Recall that a $(3, 1)$-snapshot adversary is able to gain access to the system three times and view the transcripts for only a single query. That means, the $(3, 1)$-snapshot adversary only views the memory contents for two of the three breaches while observing the memory contents and the access patterns to server memory for only a single operation in the other breach. We present the following logarithmic lower bound for any oblivious RAMs that provide protection against $(3, 1)$-snapshot adversaries:

Theorem 1 (Informal). *Any $(3, 1)$-snapshot oblivious RAM for n b-bit entries with client storage c must have overhead*

$$\Omega(\log(nb/c)).$$

Quite surprisingly, our main result proves that protecting against $(3, 1)$-snapshot adversaries with relatively weak access to the system is as challenging as protecting powerful persistent adversaries with unlimited access to the system. We note that our above lower bound asymptotically matches prior lower bounds for ORAMs with respect to persistent adversaries [LN18, PY19, HKKS19, KL21]. Our work shows that for any applications where protection of up to three breaches is required, one may use the logarithmic ORAM constructions [AKL+20] to obtain optimal overhead.

Our work is the first to prove a super-constant lower bound even when the adversary observes the transcript of only a constant number of operations. Previous lower bounds [LN18, PY19, HKKS19, LMWY20, PPY20, KL21, PY23] with persistent adversaries proved $\Omega(\log n)$ lower bounds, but required the adversary to observe the transcript of $\Theta(n)$ operations. Du, Genkin and Grubbs [DGG22] proved a $\Omega(\log \ell)$ lower bound for protecting against $(1, \ell)$-snapshot adversaries. However, the lower bound becomes trivial when the adversary views only $\ell = O(1)$ operational transcripts. Our lower bound shows that $\Omega(\log n)$ overhead is necessary even when $\ell = 1$ operational transcripts are observed. This is also the first lower bound that does not depend on the number of operational transcripts, ℓ, viewed by the adversary.

While our lower bound considers $(3, 1)$-snapshot adversaries, we note it also applies to any stronger adversaries. Formally, any ORAM requires logarithmic lower bound when consider security against a (s, ℓ)-snapshot adversary with $s \geq 3$ breaches and viewing the transcript of $\ell \geq 1$ operations. We note that our lower bound also applies to privacy guarantees that are weaker than obliviousness. Prior works have considered weaker notions including differential privacy [PY19], leakage functions commonly used in searchable encryption [PPY20] as well as read-only obliviousness. In our work, we show that our logarithmic lower bound may be extended to all three weaker privacy notions.

Sub-Logarithmic Constructions. Next, we explore various ways to circumvent the above logarithmic lower bound with respect to snapshot adversaries. In particular, we are trying to find constructions with sub-logarithmic lower bounds that still provide protections against $(3, 1)$-snapshot or stronger adversaries. As we know that this is impossible for RAMs (i.e., arrays), we explore functionalities that are weaker than RAMs that admit sub-logarithmic overhead.

- *No-Write Oblivious RAMs*[1]: Our lower bound for oblivious RAMs with respect to $(3, 1)$-snapshot adversaries critically leverages the ability to overwrite RAM entries. A natural question to study is a no-write ORAM that only enables reading entries. In Section 6.1, we show there exists a very simple construction for no-write ORAMs with $O(1)$ overhead that is secure against $(s, 1)$-snapshot adversaries for every $s \geq 1$. We note this is not surprising as no-write ORAMs never need to modify server memory to overwrite entries. However, no such separation is known for persistent adversaries as designing a sub-logarithmic no-write ORAM remains an open problem. Furthermore, prior work showed significant barriers towards non-trivial lower bounds for no-write ORAMs even against persistent adversaries [WW18].

- *Oblivious Stacks and Queues*: We also consider stacks and queues that have weaker retrieval functionalities than a RAM data structure. We show that there exist simple constructions with $O(1)$ overhead secure against $(s, 1)$-snapshot adversaries for every $s \geq 1$. The constant overhead implementation of stacks and queues provides a separation with respect to snapshot adversaries. In contrast, it is known that oblivious RAMs and stacks/queues both require logarithmic overhead against persistent adversaries [JLN19].

[1]Prior works have also referred to this primitive as read-only ORAMs. However, similar names have also been used for read-only obliviousness where security is only provided against read operations, but write operations may not be private. To differentiate, we chose to rename this primitive as a no-write ORAM. Throughout our work, we will refer to read-only and write-only ORAMs as those that provide obliviousness for only read or write operations respectively.

1.2 Related Works

Lower Bounds and Barriers. The first lower bounds for oblivious RAMs were proven in the *balls-and-bins* model that makes a non-encoding assumption on each entry. Goldreich and Ostrovsky [GO96] proved the first logarithmic ORAM lower bounds in this model. Boyle and Naor [BN16] were the first to explicitly point out this non-encoding assumption and defined the balls-and-bins model. Further works prove lower bounds in this model such as for differentially private [PPY19], searchable encryption [BF19] and one-round ORAMs [CDH20]. PIR lower bounds were studied in models nearly identical to the balls-and-bins model including for public preprocessing [BIM04, PY22] and private preprocessing [CK20, CHK22, Yeo23]. To our knowledge, all balls-and-bins lower bounds were proven against persistent adversaries.

More general logarithmic lower bounds for ORAMs were proven by Larsen and Nielsen [LN18] in the cell probe model without any non-encoding assumptions. Follow-up works have considered lower bounds for other data structures [JLN19] and weaker security guarantees including differential privacy [PY19], no operational boundary knowledge [HKKS19], searchable encryption leakage functions [PPY20] and multiple non-colluding servers [LSY20]. The highest lower bounds for oblivious data structures remains $\Omega(\log^2 n)$ for near-neighbor search [LMWY20]. Lower bounds were also proven for the setting of small block sizes in [KL21]. A recent work presented a framework for proving lower bounds for a wide range of data structures and security notions [PY23]. To our knowledge, all cell probe lower bounds were also proven with respect to persistent adversaries.

Finally, several works have presented barriers for proving lower bounds. Boyle and Naor [BN16] showed that proving non-trivial lower bounds for *offline* ORAMs that receive all operations at once would imply unknown circuit lower bounds. A subsequent work by Weiss and Wichs [WW18] showed non-trivial lower bounds for no-write ORAMs would also imply unknown lower bounds for either circuits or locally decodable circuits (both of which are long-standing open problems).

Oblivious RAMs. Oblivious RAMs were first introduced by Goldreich and Ostrovsky [GO96] and have been studied extensively for decades thereafter. For examples, see [GO96, GMOT12, KLO12, SvS$^+$13, RFK$^+$15, ZWR$^+$16, BCP16, CLT16, PPRY18] and references therein. The best constructions obtain logarithmic overhead [AKL$^+$20, AKLS21]. Many ORAM constructions have been considered for various settings such as differential privacy [WCM18, PPY19], multi-party computation [WHC$^+$14, ZWR$^+$16, Ds17], write-only obliviousness [LD13, BMNO14, RACM17] and parallel RAM access [BCP16, CLT16, AKL$^+$22] to list some examples. Of these results, write-only ORAMs are the closest to our notions. For completeness, we discuss their relation to the notion of snapshot security in Section 6.2.

Encrypted Search and Structured Encryption. Searching over encrypted data was first introduced by Song, Wagner and Perrig [SWP00]. Structured encryption was introduced by Chase and Kamara [CK10] to generalize the notion beyond encrypted indexes. In both primitives, the goal is to obtain efficient constructions while ensuring reasonable privacy upper bounded by a "sensible" leakage function. Follow-up works have considered adaptive adversaries [CGKO06], dynamic variants [KPR12, CJJ$^+$14, HK14], forward and backward privacy [Bos16, BMO17], public-key settings [BDOP04, ACD$^+$22], cache-efficiency [CT14, BBF$^+$21, MR22], Boolean queries [CJJ$^+$13, KM17, PPSY21], SQL queries [KM18], leakage suppression [KMO18, DPPS20, GKM21], frequency smoothing [GKL$^+$20], and volume-hiding [KM19, PPYY19, APP$^+$23]. Another line of work has studied the implications of various leakage profiles through abuse attacks (see [IKK12, CGPR15, ZKP16, KKNO16, LMP18, GLMP19, BKM20, KPT21] and references therein).

2 Technical Overview

Reviewing Prior Lower Bounds. We start by reviewing the logarithmic lower bounds for ORAMs protecting against persistent adversaries, the first of which was proven by Larsen and Nielsen [LN18]. There are two main proof techniques, both originating from the data structure community: information transfer [PD06] and chronogram [FS89]. Prior works on lower bound for data structures guaranteeing different levels of privacy constructed similar persistent adversaries independent of the employed technique. These persistent adversaries consist of two main parts. First, the adversary observes the transcripts of a sequence of write ORAM operations where the transcript consists of all memory probes and whether the corresponding memory cell is overwritten. The adversary records a list of all memory cells that are updated and the operation responsible for the modification. Afterwards, the adversary views the transcript for one or more read ORAM operations and attempts to correlate probed memory cells and the operation that last modified the probed cell. At a high level, the adversary is attempting to estimate which ORAM write operation is responsible for the information returned by the ORAM read operation. As a result, this forces the ORAM to make additional memory probes to hide the real ORAM write operation responsible for the relevant information that needs to be returned.

The above adversary structure has been used by prior works regardless of whether they rely on the information transfer or chronogram technique. The main difference in the two techniques is the arrangement of counting disjoint memory probes that are forced by the adversary. The information transfer technique (used in [LN18, JLN19, HKKS19, PPY20, LSY20, KL21]) arranges a binary tree over a sequence of $\Theta(n)$ ORAM operations. Each ORAM operation is uniquely assigned to a leaf of a tree and every memory probe is uniquely assigned to a node in the tree by the adversary as follows. For any probe to a memory cell during operation i, the persistent adversary finds the last operation j responsible for modifying the contents of the memory cell. This probe is then assigned to the lowest common ancestor of leaf nodes associated to operations i and j. By a combination of correctness and privacy properties, it can be shown that, for sufficiently large m, each internal node with m leaf nodes must have $\Omega(m)$ assigned probes. In total, the entire tree has $\Omega(n \log n)$ assigned probes and, thus, an $\Omega(\log n)$ lower bound may be obtained.

The chronogram technique (used in [PY19, LMWY20, PY23]) takes a different approach by arranging $\Theta(n)$ ORAM write operations into partitions of geometrically decreasing size, for some constant factor $r > 1$. The leftmost partition contains $\Theta(n)$ operations, the second left most partition contains $\Theta(n/r)$ operations and so forth for a total of $p = \Theta(\log n)$ partitions. At last, a final ORAM read operation is performed after all $\Theta(n)$ ORAM write operations. If the last operation was chosen to read one random entry that was overwritten in the i-th partition, correctness implies that $\Omega(1)$ memory cells that were last overwritten by an operation during the i-th partition must be probed. On the other hand, privacy requires that this must be the case for all other p partitions thus obtaining an $\Omega(p) = \Omega(\log n)$ lower bound.

In either case, prior lower bounds heavily relied on the persistent nature of the adversary to view the transcripts for all operations. A weaker (s, ℓ)-snapshot adversary viewing transcripts for $\ell = O(1)$ operations will not be able to accumulate a log of all memory cell updates as done in the past.

New Techniques for Snapshot Adversaries. In our work, we develop new snapshot adversarial strategies for very weak compromise settings with only a constant $s = 3$ number of breaches and observing the transcript of just $\ell = 1$ operation. We first note that all prior persistent adversary strategies used the transcripts from all $\Theta(n)$ operations. In particular, they never utilized the contents of server memory that are also in the adversary's view. This is not surprising as a persistent adversary with transcripts from all operations can easily compute the exact contents of server memory at any point in time. However, one cannot compute the exact server memory when only observing the transcripts of a small subset of transcripts as is the case for snapshot adversaries with $\ell = 1$.

Instead, our new snapshot adversarial strategy will heavily utilize the contents of server memory that may be viewed during the $s = 3$ data breaches. Consider any pair of breaches where the adversary is able to view the contents of server memory. We observe that the adversary is able to compute the memory cells that have changed between the two snapshots of server memory. Therefore, a snapshot adversary is able to record memory cell updates between each of the s breaches that occur. The snapshot adversary is able to observe a list of all memory cell updates with coarser-grained update times where updates are only known to have occurred during operations between any pair of breaches. For $s = 3$, the snapshot adversary is able to categorize memory cell updates into three groups: cells updated before the first breach, cells updated between the first and second breach and cells updated between the second and third breach.

Next, we adapt the chronogram technique and show that it may still be used when a snapshot adversary is only able to sample and test a single adversarial event. Recall that the chronogram technique splits $\Theta(n)$ ORAM write operations into $p = \Theta(\log n)$ partitions. The goal is to prove that the final ORAM read operation must probe $\Omega(1)$ memory cells that were last overwritten from each of the p partitions. A persistent adversary is able to check whether all p events are satisfied (as done in prior works). In our proof, we instead show that it suffices for a snapshot adversary to randomly sample and test only one event. Suppose in fact that we have an ORAM construction that does not satisfy the conditions for one of the $p = \Theta(\log n)$ events with non-negligible probability q. Then, the snapshot adversary will sample this event and detect the violation with probability $q/p = \Omega(q/\log n)$ that is also non-negligible and thus privacy is violated. Therefore, a snapshot adversary only needs to test one random event. In our full proof, we extend this idea to constant adversarial advantage using randomization to hide the partitioning structure from the adversary.

Finally, we show that our seemingly weak snapshot adversarial strategy with $s = 3$ and $\ell = 1$ is sufficient to randomly sample and test one of the adversarial events. To do this, the snapshot adversary first samples one of the p partitions uniformly at random. Afterwards, the snapshot adversary chooses to perform breaches directly before and after all ORAM write operations of the sampled partition. In these two breaches, the snapshot adversary only sees server memory contents and no operational transcripts. Next, the adversary performs the third breach for the final ORAM read operation and views the transcript for this operation. Using the server memory from all three breaches, the adversary can determine the set of memory cells W that were updated by ORAM write operations during the sampled partition, but were not modified until the final ORAM read operation. As the last step, the adversary can check whether the intersection of the memory cells probed by the final ORAM read operation and W contains at least $\Omega(1)$ memory cells that is equivalent to testing the adversarial event corresponding to the sampled partition. Therefore, we show that an ORAM requires $\Omega(\log n)$ overhead against $(3, 1)$-snapshot adversaries.

As a side note, we chose to adapt the chronogram technique as it enables smaller values of $\ell = 1$. Our same techniques could be applied to the information transfer proof where the number of probes assigned to each internal node could be the adversarial event. Unfortunately, testing events associated to nodes with many leaf nodes would require the adversary to view the transcripts of

$\ell = \Omega(n)$ operations. Thus the information tree methodology would use a stronger adversary and, thus, yield a weaker lower bound than what we obtain.

Sub-Logarithmic Constructions. We overview the simple constructions of our no-write ORAM secure against $(s, 1)$-snapshot adversaries for any $s \geq 1$. First, we note that security against an $(s, 0)$-snapshot attack is trivial for no-write ORAMs that never need to update server memory. For example, the server can store an encrypted array and perform reads by simply reading the i-th entry. To protect against $(s, 1)$-snapshot attacks, we can simply have the server store a random permutation π of the encrypted array. For reads to entry i, the client simply downloads the $\pi(i)$-th entry of the encrypted array. This provides a separation for no-write and read-write ORAMs against $(3, 1)$-snapshot adversaries that is not known with respect to persistent adversaries.

Next, we show that we can obtain sub-logarithmic overhead for oblivious stacks and queues with weaker functionality than ORAMs. As stacks and queues may only push or pop entries to a fixed location, we can ensure that locations of memory updates only depend on the length of the operational sequence. This immediately obtains an $(s, 0)$-snapshot oblivious stack/queue for any $s \geq 1$. Using the same trick as done for no-write ORAMs, we can obtain $(s, 1)$-snapshot security using a random permutation. Therefore, we give a $O(1)$ overhead implementation of oblivious stacks/queues providing a separation from ORAMs. This separation is not possible for persistent adversaries where $\Omega(\log n)$ lower bounds are known for oblivious stacks/queues [JLN19].

3 Definitions

3.1 Snapshot Security

We start by defining (s, ℓ, ϵ)-*snapshot security* that guarantees that any PPT (s, ℓ)-snapshot adversary has advantage at most ϵ. Roughly speaking an (s, ℓ)-*snapshot* adversary is an adversary that perform a *snapshot attack* that consists of s *snapshot windows* for a total span of ℓ operations. When ϵ is clear from the context, we may drop ϵ and use (s, ℓ)-snapshot security.

More formally, we model an (s, ℓ)-snapshot adversary $\mathcal{A}$ as a two-part adversary $(\mathcal{A}_0, \mathcal{A}_1)$. Algorithm $\mathcal{A}_0(1^n)$ outputs two equal-length operational sequences, O_1 and O_2, as well as a sequence of *snapshot windows* denoted by

$$S = \big((t_1, \ell_1), (t_2, \ell_2), \ldots, (t_s, \ell_{|S|})\big)$$

describing the snapshot attack the adversary intends to mount. Specifically, a pair (t_i, ℓ_i) specifies a snapshot window of length ℓ_i starting with operation t_i. As a result of this snapshot window, the adversary will get the memory content before the operation with index t_i is executed and the server memory accesses for the following ℓ_i operations indexed $t_i, \ldots, t_i + \ell_i - 1$. Note that if $\ell_i = 0$ then the adversary only gets a snapshot of the memory content before operation t_i is performed and no memory access. Without loss of generality, we will also suppose that the snapshot windows are given in increasing order of the t_i and that they do not overlap.

Adversary $\mathcal{A}_1$ receives the leakage obtained from the snapshot attack specified by $\mathcal{A}_0$ when one of the two sequences of operations, O_1 and O_2, is executed. $\mathcal{A}_1$ outputs its guess as to which one has actually been used to produce the leakage. An adversary $\mathcal{A}$ is an (s, ℓ)-*snapshot adversary* if the following two conditions are satisfied: S contains at most s snapshot windows, that is $|S| \leq s$ and the total snapshot time $\ell_1 + \ldots + \ell_{|S|}$ is at most ℓ.

$\mathsf{Read}(x)$	$\mathsf{Write}(x,y)$	$\mathsf{LRead}(x)$	$\mathsf{LWrite}(x,y)$
1. Return $M[x]$.	1. Set $M[x] \leftarrow y$. 2. Return $\perp$.	1. Set $\mathcal{L} \leftarrow \mathcal{L} \mid\mid (\mathsf{read}, x)$. 2. Return $M[x]$.	1. Set $\mathcal{L} \leftarrow \mathcal{L} \mid\mid (\mathsf{write}, x, y)$. 2. Set $M[x] \leftarrow y$. 3. Return $\perp$.

Figure 1: (Leaky) Memory Read and Write Oracles.

We next define an experiment that is parameterized by the data structure DS and by a snapshot adversary $\mathcal{A}$ and a bit $\beta \in \{0,1\}$. The experiment will execute the data structure DS on the sequence of operation O_β output by $\mathcal{A}_0$ and records the leakage of DS from the snapshot attacks specified by $\mathcal{A}_0$.

For any data structure DS, we will assume that DS has access to an underlying physical memory M that is indexed uniquely by integers from $[|M|]$. Access will be exclusively performed by means of two oracles: one for reading memory cells and one for writing to memory cells. In the experiment, DS will be equipped with standard oracles (Read and Write, see Figure 1) except for the snapshot windows specified by the adversary during which the access to memory will be performed by means of leaky oracles (LRead and LWrite, see Figure 1) that record the read/write operation performed for the adversary.

We present the formal definition of our experiment defining snapshot security in Figure 2. Finally, we present the formal definition of (s, ℓ, ϵ)-snapshot security for data structures using the above defined experiment.

Definition 1 ((s, ℓ, ϵ)-Snapshot Security)**.** *Let $s \geq 1$ and $\ell \geq 0$ be fixed integers and let $0 \leq \epsilon < 1$. A data structure DS is (s, ℓ, ϵ)-snapshot secure if for any PPT (s, ℓ)-snapshot adversary $\mathcal{A}$, the following holds:*

$$\left|\Pr[\mathsf{Expt}_{\mathsf{DS},\mathcal{A}}(n,0) = 1] - \Pr[\mathsf{Expt}_{\mathsf{DS},\mathcal{A}}(n,1) = 1]\right| \leq \epsilon,$$

for all sufficiently large n.

In our work, we will focus on RAM data structures that maintain an array of length n where the two supported operations are retrieving and update an entry. Throughout our work, we will denote a RAM data structure DS satisfying as (s, ℓ)-snapshot security as an (s, ℓ)-snapshot oblivious RAM or an oblvious RAM secure against (s, ℓ)-snapshot adversaries.

Discussion about Adaptive Adversary. In our above definition, we define the adversary to be non-adaptive. In particular, the adversary must pick the t snapshot windows at the beginning of the experiment. A stronger definition would be to enable the adversary to adaptively choose snapshot times and lengths that may depend on the leakage of previously seen snapshot leakage. As we are proving lower bounds, a weaker security definition implies a stronger lower bound. In other words, our lower bounds for non-adaptive adversaries would immediately apply to settings with more realistic adaptive adversaries.

Discussion about $\ell < s$. In our definition, we enable values of $\ell < s$. That is, the total snapshot time may be smaller than the number of snapshots. This is allowed because we allow snapshots of length 0 that is formally denoted by $\ell_i = 0$. In this case, note that the experiment only adds the current contents of memory cells M to the snapshot leakage $\mathcal{L}$ but not the memory access patterns for any operations performed by DS. This formalizes the setting where an adversary may perform a snapshot attack but no operations occur during the attack.

```
$\mathsf{Expt}_{\mathsf{DS},\mathcal{A}}(n, \beta)$:

  1. Execute $(O_0, O_1, S, \mathtt{st}) \leftarrow \mathcal{A}_0(1^n)$.
  2. Parse $S = \{(t_1, \ell_1), \ldots, (t_{|S|}, \ell_{|S|})\}$.
  3. Set leakage $\mathcal{L} \leftarrow \emptyset$ and initialize $\mathsf{DS}$ with $n$ blocks each consisting of randomly select
     blocks of $b$ bits.
  4. Set $i \leftarrow 1$.
  5. While $i \leq |O_\beta|$:
     (a) If $i = t_j$ for some $j \in [|S|]$:
         i. Set $\mathcal{L} \leftarrow \mathcal{L} \mid\mid (\mathsf{memory}, M)$.
         ii. For $k = 1, \ldots, \ell_j$:
             A. Execute $\mathsf{DS}^{\mathsf{LRead},\mathsf{LWrite}}(O_b[i])$.
             B. Set $i \leftarrow i + 1$.
     (b) Else:
         i. Execute $\mathsf{DS}^{\mathsf{Read},\mathsf{Write}}(O_b[i])$.
         ii. Set $i \leftarrow i + 1$.
  6. Execute $\beta' \leftarrow \mathcal{A}_1(\mathtt{st}, \mathcal{L})$.
  7. Return $\beta'$.
```

Figure 2: Experiment for (s, ℓ)-snapshot security.

3.2 Cell Probe Model

In this work, we prove our lower bounds in the oblivious cell probe model introduced by Larsen and Nielsen [LN18] that adapts the cell probe model of Yao [Yao81]. In this model, there exists a client and a server with separate storage. The client has c bits of storage. The server's memory consists of memory cells that consist of $w \geq 1$ bits each. The only operation that is charged any cost is probing a cell in server memory to retrieve or update its contents. All other operations are free of cost including computation or accessing client storage. Additionally, we will assume there exists a long, but finite random string $\mathbf{R}$ that is accessible to both the client and the server without any cost. One can view $\mathbf{R}$ as a random oracle. As we consider a weak cost model, our lower bounds will apply for any reasonable model of computation.

4 Lower Bound

In this section, we prove a logarithmic lower bound for any RAM data structure DS that is $(3, 1)$-snapshot private. The adversary is able to employ three snapshot attacks at various time points. However, the three snapshot attacks are able to observe the memory access pattern of exactly one operation performed by the DS during one snapshot attack. In the other two snapshot attacks, the adversary may only view the current memory contents of DS. Even for such a weak snapshot adversary, we are able to prove the following lower bound:

Theorem 2. *For any $0 \leq \epsilon \leq 1/16$, let DS be a $(3, 1, \epsilon)$-snapshot private RAM data structure for n entries each of $b \geq 1$ bits implemented over $w = \Omega(\log n)$ bits using client storage of $c \geq 1$ bits in the cell probe model. If DS has amortized write time t_w and expected amortized read time t_r with failure probability at most $1/3$, then*

$$t_r + t_w = \Omega\left(b/w \cdot \log(nb/c)\right).$$

To do this, we actually prove the following lemma that assumes that the block size b is a sufficiently large constant ($b > 60$ is sufficient). Although, we show that this still implies the desired above theorem for all values of $b \geq 1$.

Lemma 1. *For any $0 \leq \epsilon \leq 1/16$, let DS be a $(3, 1, \epsilon)$-snapshot private RAM data structure for n entries each of $b > 60$ bits implemented over cells of size $w = \Omega(\log n)$ bits using client storage of $c \geq 1$ bits in the cell probe model. If DS has amortized write time t_w and expected amortized read time t_r with failure probability at most $1/3$, then*

$$t_r + t_w = \Omega\left(b/w \cdot \log(nb/c)\right).$$

We next show that the lemma above implies the main theorem.

Proof of Theorem 2. Assuming Lemma 1, we know that Theorem 2 holds for any $b > 60$. Towards a contradiction, suppose that Theorem 2 is false for some $1 \leq b \leq 60$. That is, there exists a DS with overhead that contradicts the lower bound $\Omega(b/w \cdot \log(nb/c)) = \Omega(\log(n/b)/w)$ as $b = \Theta(1)$. Note, we can construct DS' for blocks of size 60 by simply keeping $60/b$ copies of DS. The resulting read and write overheads of DS' are only a constant multiplicative factor higher than DS that would contradict Lemma 1. Therefore, Lemma 1 implies Theorem 2 for all values of $b \geq 1$. $\square$

The remainder of this section is dedicated to proving Lemma 1. We prove the lower bound in the three following steps:

1. First, we will present a snapshot adversarial strategy that uses only three snapshots of which only one contains the memory access pattern of the operation. The adversarial algorithm will pick a hard sequence of operations as well as the locations and lengths of snapshots non-adaptively.

2. Next, we will analyze the adversarial method by relating the snapshot adversary's advantage to the efficiency of a one-way communication protocol that encapsulates the correctness guarantees of DS.

3. Finally, we show that we can derive our desired lower bounds using the analysis of the advantage of the snapshot adversarial strategy.

Discussion about Other Models. We note that prior works have considered lower bounds in other models including small blocks when w is small [KL21, PY23] and when operational boundaries are hidden from the adversary [HKKS19]. We believe that adapting prior techniques would enable extending our lower bound to these models, but we leave it as future work to not distract the readers from our main goal of studying snapshot security.

We do extend our lower bounds for weaker notions of snapshot security that may be easily adapted to our proof techniques. These weaker snapshot security notions include read-only obliviousness, differential private guarantees and encrypted search leakage (see Section 5 for more details).

In our lower bound, if we restrict the adversary to only choose snapshots within a consecutive subset of Z operations, then our lower bound would be $\Omega(b/w \cdot \log(Zb/c))$ matching the construction in [DGG22].

4.1 Constructing a Snapshot Adversary

We start by presenting our $(3,1)$-snapshot adversary that will be used to prove our lower bound. Recall that our snapshot adversary must submit two operational sequences O_0 and O_1 of equal length as well as a description of the times and lengths of the three snapshot attacks $S = \{(t_1,\ell_1),(t_2,\ell_2),(t_3,\ell_3)\}$ such that $\ell_1+\ell_2+\ell_3 = 1$ representing that only one snapshot can view the access patterns to memory for exactly one operation. Afterwards, the adversary receives the leakage from the s snapshot attacks and must output a bit b' that aims to distinguish whether O_0 or O_1 was executed.

Our adversary is inspired by the chronogram lower bound proof technique introduced by Fredman and Saks [FS89] and it outputs two sequences each consisting of m writes, for some $n/2 < m \leq n$, followed by a single read, for a total of $m+1$ operations. To describe the snapshot windows output by the adversary, we introduce the notion of a *partition* of the sequence of the m write operations The partitions are disjoint consecutive subsequences, whose sizes are geometrically increasing by some ratio $r > 1$ to be fixed. The write operations are naturally indexed $1,\ldots,m$ with 1 being the index of the first write to be executed and m the index of the last operations. Partitions are instead indexed in reverse order; that is, partition 0 is the rightmost partition and consists of the write operation of index m that is executed immediately before the read operation. The second partition consists of the r write operations that occur exactly before the last write operation; that is, it consists of the write operations indexed $m-r,\ldots,m-1$. Generally, the i-th partition will consist of r^i consecutive write operations and there will be a total of $p = O(\log n/\log r)$ partitions. The $(p-1)$-th partition will be the leftmost and largest partition and the 0-th partition will be the rightmost and smallest partition. Finally, we will denote the index of the operation at the start of the i-th partition by p_i and we can see that $p_i + r^i - 1$ is the last operation of the i-th partition.

Now, we are ready to define the adversary's algorithm for generating the sequence of operations O_0, O_1 and the snapshot windows in S. The adversary $\mathcal{A}^{r,i} = (\mathcal{A}_0^{r,i}, \mathcal{A}_1^{r,i})$ is parameterized by rate $r > 1$ and an index i for a partition and proceeds as follows. $\mathcal{A}_0^{r,i}$ picks a random number m uniformly at random integer from the interval $[n/2+1, n]$. As already described above, O_0 and O_1 will consist of the same m write operations followed by a read operation that will differ between the two sequences. The first m operations will be write operations to indices $1,2,\ldots,m$ of a uniformly random b-bit string. For O_0, the final operation will be a read to the index 1. For O_1, a read operation will be performed on a uniformly random index that was overwritten in the i-th partition. This can be done by picking an uniformly random index from the interval $[p_i, p_i + r^i - 1]$ to be used as input to the read. Finally, the snapshot windows outputs by the adversary are $(p_i, 0)$, $(p_i + r^i, 0)$ and $(m+1, 1)$. In other words, the adversary elects to see the memory before partition i starts, just after it is completed, and before the read operation. In addition the adversary also sees the memory access performed by the read operation. A formal description of $\mathcal{A}_0^{r,i}$ is provided below:

$\mathcal{A}_0^{r,i}(1^n)$:

1. Pick integer m uniformly at random from interval $[n/2, n]$.
2. Generate m uniformly random b-bit strings $\mathbf{B}_1, \ldots, \mathbf{B}_m$.
3. Set $O_0 = (\mathsf{write}(1, \mathbf{B}_1), \ldots, \mathsf{write}(m, \mathbf{B}_m), \mathsf{read}(1))$.
4. Pick a uniformly index $j \in [p_i, p_i + r^i - 1]$, where p_i is the index of the first operation in the i-th partition.
5. Construct $O_1 = (\mathsf{write}(1, \mathbf{B}_1), \ldots, \mathsf{write}(m, \mathbf{B}_m), \mathsf{read}(j))$.
6. Set $S = ((p_i, 0), (p_i + r^i, 0), (m + 1, 1))$.
7. Return (O_0, O_1, S).

The leakage $\mathcal{L}$ associated with the snapshot attack output by $\mathcal{A}_0^{r,i}(1^n)$ consists of $((\mathsf{memory}, M_{p_i}), (\mathsf{memory}, M_{p_i+r^i}), (\mathsf{memory}, M_m), \mathcal{T}_{m+1})$ where (memory, M_x) denotes the contents of all memory cells before the x-th operation in O_β and $\mathcal{T}_x \subseteq [|M|]$ is the access pattern to memory cells performed by the x-th operation.

The leakage $\mathcal{L}$ is passed as input to $\mathcal{A}_1^{r,i}$ that, roughly speaking, proceeds as follows. First it computes the set of memory locations that were last overwritten in the i-th partition and never overwritten until the final read operation. To do this, $\mathcal{A}_1^{r,i}$ computes the set of memory cell locations whose contents were changed in between the first and second snapshots, $U = \{j \in [|M|] \mid M_{p_i}[j] \neq M_{p_i+r^i}[j]\}$. Afterwards, the adversary computes a similar set of locations that changed between the second and third snapshot, $V = \{j \in [|M|] \mid M_{p_i+r^i}[j] \neq M_{m+1}[j]\}$. Lastly, the adversary computes the set difference $U \setminus V$ to obtain the desired set of memory cell locations that were last overwritten in the i-th partition. Finally, the adversary checks whether there exists any memory location in the intersection of $U \setminus V$ and $\mathcal{T}_{m+1}$. If the intersection is strictly less than $\rho \cdot b/w$ for some constant $\rho > 0$ that we will choose later, $\mathcal{A}_1^{r,i}$ outputs 0. Otherwise, $\mathcal{A}_1^{r,i}(\mathcal{L})$ returns 1 when the intersection is contains at least $\rho \cdot b/w$ locations. We formalize this adversarial algorithm below:

```
$\mathcal{A}_1^{r,i}(\mathcal{L})$:
    1. Parse $\mathcal{L} = \{(\mathsf{memory}, M_{p_i}), (\mathsf{memory}, M_{p_i+r^i}), (\mathsf{memory}, M_{m+1}), \mathcal{T}_{m+1}\}$.
    2. Compute $U_i = \{j \in [|M|] \mid M_{p_i}[j] \neq M_{p_i+r^i}[j]\}$.
    3. Compute $V_i = \{j \in [|M|] \mid M_{p_i+r^i}[j] \neq M_{m+1}[j]\}$.
    4. Compute $W_i = U_i \setminus V_i$.
    5. If $|W_i \cap \mathcal{T}_{m+1}| < \rho \cdot b/w$, return 0.
    6. Else if $|W_i \cap \mathcal{T}_{m+1}| \geq \rho \cdot b/w$, return 1.
```

Extension to Differential Privacy. We note that our adversary only submits two operational sequences that differ in exactly one operation (that is, the final read operation). Therefore, this adversary could also be applied to snapshot private data structures that are differentially private (following prior works such as [PY19]) where the adversarial advantage is a function of the number of differing operations in the two adversarially chosen sequences. As a result, our adversarial algorithm may be used to prove lower bounds snapshot private RAMs that are only differentially private. We point readers to Section 5.1 for more details.

Extension to Read-Only Obliviousness. Another property that is enjoyed by our adversarial algorithm is that the adversary only views memory access patterns for read operations. Therefore, we could also extend use the same adversary to extend our lower bound for snapshot private RAMs that only provide privacy for read operations while write operations may be completely public. Read-only obliviousness may be natural in certain settings where the underlying data is public while the sensitive information is the portion of the information that is of interest for the querier. See Section 5.2 for more information.

Extension to Structured Encryption Leakage. Most structured encryption schemes have some leakage functions known as key-equality revealing whether two operations are for the same index. Extending prior work [PPY20], we show that any ORAMs providing even slightly more security beyond key-equality also require $\Omega(\log n)$ overhead. See definitions and proofs in Section 5.3.

Barriers for $(2,1)$-Snapshot Security. Finally, we quickly discuss barriers towards extending our proof towards $(2,1)$-snapshot security. In the above strategy, the adversary divides server memory updates into $p = \Theta(\log n)$ disjoint partitions. Afterwards, the adversary samples one partition and checks whether the final ORAM read operation accesses any memory in the sampled partition. To our knowledge, the most intelligent adversary requires three data breaches to perform this action: two data breaches for the beginning and end of the partition and one for the final ORAM read operation. Furthermore, all three data breaches seem necessary to perform this check.

4.2 Analyzing Adversarial Strategy

Next, we analyze the snapshot attack described in the previous section. In particular, we will use this specific snapshot adversary to prove certain properties about any RAM data structure that is $(3,1)$-snapshot private. To do this, we will specifically analyze the set of operational sequences that the snapshot adversary will output as O_1. Recall that these are the sequences that picks a random index j to read from the set of r^i indices that are overwritten in the i-th partition. We show that for these sequences, the snapshot adversary is likely to output 1 with at least constant probability for many choices of partitions.

Formally, we define P to be the set of all partitions containing at least $\max(100c/b, 200\log n/b)$ write operations. By our choice of $m = \Theta(n)$, we get

$$|P| = \log_r m - \max(\log_r(100c/b), \log_r(200\log n/b)) = \Theta(\log(nb/c))$$

since $r \geq 2$ is a constant. First, we show that (99/100)-fraction of the partitions in P satisfy a certain property of distributions of memory accesses performed during the writes of the i-th partition. To do this, we define the following sets of memory cell accesses with respect to the i-th partition. Recall that we denote M_x to be the contents of memory before the x-th operation and $\mathcal{T}_x$ to be the set of memory locations that are accessed (either read or overwritten) during the x-th operation. Then, we define the following sets that are computed when executing the experiment $\mathsf{Expt}_{\mathsf{DS},\mathcal{A}^{r,i},3,1}(n,1)$. Recall that is executing DS on the random operational sequence O_1 produced by the adversary $\mathcal{A}^{r,i}$.

- Denote U_i to be the set of memory locations that are overwritten by write operations in the i-th partition as defined in the description of $\mathcal{A}_1^{r,i}$. Formally,
$$U_i = \{j \in [|M|] \mid M_{p_i}[j] \neq M_{p_i+r^i}[j]\}.$$

- Denote Y_i to be the set of memory locations that are overwritten in the i-th partition and accessed by write operations after the i-th partition. Formally,
$$Y_i = U_i \cap (\mathcal{T}_{p_i+r^i} \cup \mathcal{T}_{p_i+r^i+1} \cup \ldots \cup \mathcal{T}_m).$$

Now, we show that for a large number of partitions $i \in P$, the size of Y_i cannot be too large assuming that the write overhead of the data structure DS beats the lower bound. We call these partitions the *critical* partitions.

Lemma 2. *Let* DS *be a* $(3,1)$*-snapshot private RAM data structure for n b-bit entries run over a memory of w-bit cells using c bits of client storage and suppose that the amortized write time is* $t_w = o(b/w \cdot \log(nb/c))$*. For any* $r \geq 2$ *and sufficiently large n, there exists a set of* critical *partitions that contains at least* $(99/100)$*-fraction of the partitions* $i \in P$ *such that*

$$\mathbb{E}[|Y_i|] \leq r^{i-1} \cdot b/w.$$

Proof. We start by bounding the probability (taken over the choices of m) that a fix memory access belongs to Y_i, for a fixed partition i. We denote by β the index of the operation that performs the memory access and by $\alpha < \beta$ the index of the operation that last overwrote the cell before it is accessed by operation β. Observe that for the memory access to belong to Y_i it must be the case that operation α falls into partition i and that operation β falls into partition $j < i$. Let y denote the integer such that $z_{y-1} \leq \beta - \alpha < z_y$, where $z_y = 1 + r + \ldots + r^y$ is the number of operation in partitions 0 to y. Note that such a y is uniquely determined since z_y increases with y. Let us now distinguish two cases.

In the first case it holds that $i \leq y-1$. Suppose that β is the last write operation (that is, it is executed just before the read operation). Since $\beta - \alpha \geq z_{y-1}$, operation α is executed before partition $y-1$ is started and, since $i \leq y-1$, before partition i is started. Therefore α cannot be one of the operation of partition i. If β is not the last operation then α will be executed even earlier and then will not be part of partition i for the same reason.

Let us now consider the case $i \geq y$. Observe that if β belongs to partition $j \leq i-1$ then it must occur between the start of partition $i-1$ and the last write operation. Therefore epoch $i-1$ cannot start before operation $\beta - z_{i-1} + 1$. On the other hand, if α is in partition i then partition $i-1$ cannot start after β for otherwise α and β are both in partition i. Therefore there at most z_{i-1} good choices for the start of epoch $i-1$ each corresponding to a different choice of m. Moreover, epoch $i-1$ must start between α and β and thus there are at most $\beta - \alpha < x_y$ good choices. As a result, the probe contributes to Y_i with probability at most $\min\{z_{i-1}, z_y\} \cdot 2/n$ for any $i \geq y$, as there are $n/2$ possible choices of m and each completely defines the partitioning.

By the two cases above, for every partition i, the contribution of a memory access to $|Y_i|/r^i$ is at most

$$\sum_{y \leq i} \frac{2}{r^i n} \cdot \min\{z_{i-1}, z_y\} \leq \sum_{y \leq i} \frac{4}{r^i n} \cdot \min\{r^{i-1}, r^y\} \leq \frac{4}{n}\left[\frac{1}{r} + \sum_{y \leq i-1} r^{y-i}\right] \leq \frac{12}{rn}.$$

since $z_{i-1} \leq 2r^{i-1}$ assuming $r \geq 2$. Each memory access can contribute to a single Y_i and thus, for a random partition i, the expected contribution of a memory access to $|Y_i|/r^i$ is at most $12/(rn|P|)$. Since there are at most $n \cdot t_w$ memory accesses, we can conclude that for a random partition i

$$\mathbb{E}[|Y_i|/r^i] \leq \frac{12}{r} \cdot \frac{t_w}{|P|} \leq \frac{1}{100 \cdot r} \cdot \frac{b}{w}$$

since, for some $\delta > 0$ and sufficiently large n, $|P| \geq \delta \log(nb/c)$ and $t_w \leq \delta/1200 \cdot b/w \cdot \log(nb/c)$. The lemma follows by Markov's inequality. □

Using the above lemma, we can now derive properties about the advantage achieved by our snapshot adversary. We say that, for any partition i such that the above lemma applies, this partition i is *critical*. For any critical partition i, we show that $\mathcal{A}^{r,i}$ outputs 1 with constant probability when run for $\beta = 1$ (that is, it receives the leakage obtained from executing the sequence of operations O_1). The proof uses the correctness property of the data structure DS and shows that it is possible to derive a too-good-to-be-true, prefix-free compression scheme that will contradict Shannon's source coding theorem if the adversary does not output 1 with high enough probability. In other words, the DS provides an impossible method of storing and retrieving randomness without even requiring looking up the randomness in memory.

Lemma 3. *Let $b > 60$ and let DS be a $(3,1)$-snapshot private RAM data structure for n b-bit entries run over a memory of w-bit cells using c bits of client storage. Suppose that the amortized write time of DS is $t_w = o(b/w \cdot \log(nb/c))$. For $r \geq 32$, $w = \Omega(\log n)$, and a sufficiently small but constant $\rho > 0$, the following holds for every critical partition $i \in P$,*

$$\Pr[\mathsf{Expt}_{\mathsf{DS},\mathcal{A}^{r,i}}(n,1) = 1] \geq 1/8.$$

Proof. Towards a contradiction, suppose that the above probability statement is false and let us unpack the implications of this assumption. Note, that $\mathcal{A}^{r,i}$ outputs 1 if and only if the final read operation accesses at least $\rho \cdot b/w$ memory locations whose contents were last changed by one of the r^i write operations in the i-th partition. The above assumption essentially means that the underlying DS will most likely only access a small number of memory locations that were changed by write operations in the i-th partition. Using this property, we will construct an impossibly efficient compression algorithm for random coin tosses. We show that we can use the i-th partition to compress random coin tosses in an efficient manner that will contradict Shannon's source coding theorem. At a high level, the idea is to execute DS where the encoding algorithm will embed the random coin tosses as blocks into the write operations of the i-th partition. Afterwards, the encoding algorithm will send enough information for the decoder to execute DS in an identical manner to retrieve the written random coin tosses.

Formally, both the encoder and the decoder will receive shared random public coins denoting the random blocks that will be inputs to write operations outside of the i-th partition. Only the encoding algorithm will receive the random blocks of write operations in the i-th partition. We denote $\mathbf{B}^{-i} = (\mathbf{B}_1, \ldots, \mathbf{B}_{p_i-1}, \mathbf{B}_{p_i+r^i}, \ldots, \mathbf{B}_m)$ as the set of blocks that in write operations outside of the i-th partition. Furthermore, let $\mathbf{B}^i = (\mathbf{B}_{p_i}, \mathbf{B}_{p_i+1}, \ldots, \mathbf{B}_{p_i+r^i-1})$ be the blocks in write operations inside the i-th partition. Finally, both algorithms will receive random coin tosses $\mathbf{R}$ that will be used as the internal randomness when executing DS. We will assume that $\mathbf{R}$ contains enough randomness to execute $m+1$ operations on DS.

Encoding Algorithm: Receives $\mathbf{B}^{-i}, \mathbf{B}^i$ and $\mathbf{R}$ as input.

1. Set encoding $X = \emptyset$.

2. Execute the operations $\mathsf{write}(1, \mathbf{B}_1), \ldots, \mathsf{write}(p_i - 1, \mathbf{B}_{p_i-1})$ using DS and randomness $\mathbf{R}$. That is, execute all write operations occurring before the i-th partition.

3. Record the memory contents M_{p_i} before the first operation in the i-th partition.

4. Execute the operational sequence $\mathsf{write}(p_i, \mathbf{B}_{p_i}), \ldots, \mathsf{write}(p_i + r^i - 1, \mathbf{B}_{p_i+r^i-1})$ using DS and unused randomness from $\mathbf{R}$. That is, execute all write operations in the i-th partition.

5. Record the memory contents $M_{p_i+r^i}$ after the last operation in the i-th partition.

6. Append the c-bit content of client memory after the last operation in the i-th partition to encoding X.

7. Compute the set $U_i = \{j \in [|M|] \mid M_{p_i}[j] \neq M_{p_i+r^i}[j]\}$ that are the memory cells whose contents were changed during the i-th partition.

8. Execute the operations $\mathsf{write}(p_i + r^i, \mathbf{B}_{p_i+r^i}), \ldots, \mathsf{write}(m, \mathbf{B}_m)$ using DS and unused randomness from $\mathbf{R}$. That is, execute all write operations after the i-th partition. During the execution, record the transcripts $\mathcal{T}_{p_i+r^i}, \ldots, \mathcal{T}_m$ as the memory locations that are accessed during each of the write operations.

9. Record the memory contents M_{m+1} before the $(m+1)$-st and final read operation.

10. Compute the set $V_i = \{j \in [|M|] \mid M_{p_i+r^i}[j] \neq M_{m+1}[j]\}$ that are the memory cells whose contents were changed after the i-th partition.

11. Compute the set $W_i = U_i \setminus V_i$ that are the memory cells whose contents were last changed during the i-th partition and not by any operations after the i-th partition.

12. Compute $Y_i = U_i \cap (\mathcal{T}_{p_i+r^i} \cap \ldots \cap \mathcal{T}_m)$ that are the set of memory cells that were overwritten in the i-th partition and accessed by a write operation after the i-th partition. We will assume Y_i consists of both the memory cell locations and contents when they were accessed.

13. Append $|Y_i|$ and Y_i to encoding X.

14. Set $Q = \emptyset$,

15. For each $j = p_i, \ldots, p_i + r^i - 1$:

    (a) Execute $\mathsf{read}(j)$ using DS and unused randomness from $\mathbf{R}$ and keep track in $\mathcal{T}^j_{m+1}$ of the memory cells accessed during the read operation along with their content.

    (b) Compute $Q^j_i = \mathcal{T}^j_{m+1} \cap W_i$ consisting of the locations and contents of memory cells that were last overwritten in the i-th partition and accessed by $\mathsf{read}(j)$.

    (c) If $|Q^j_i| \leq \rho \cdot b/w$ and the output of operation $\mathsf{read}(j)$ is correct, append $(0, Q^j_i)$ to X and add j to Q. We pad Q^j_i with dummy values up to exactly $\rho \cdot b/w$ memory cells.

    (d) Otherwise when $|Q^j_i| > \rho \cdot b/w$ or the answer to $\mathsf{read}(j)$ is incorrect (that is, the read returns something different from $\mathbf{B}_j$), append $(1, \mathbf{B}_j)$ to X.

    (e) Rewind DS to its state before executing the $\mathsf{read}(j)$.

16. If $|Q| \geq r^i/4$, return $0 \mid\mid X$.

17. Otherwise when $|Q| < r^i/4$, return $1 \mid\mid \mathbf{B}^i$ where $\mathbf{B}^i = (\mathbf{B}_{p_i}, \ldots, \mathbf{B}_{p_i+r^i-1})$.

Decoding Algorithm: Receives $\mathbf{B}^{-i}, \mathbf{R}$ and the encoding X as input.

1. If the first bit of the encoding X is 1, then simply decode and return the next $r^i \cdot b$ bits as $\mathbf{B}^i = (\mathbf{B}_{p_i}, \ldots, \mathbf{B}_{p_i+r^i-1})$.

2. Otherwise, parse $X = 0 \mid\mid M^c \mid\mid |Y_i| \mid\mid Y_i \mid\mid (b_{p_i}, X_{p_i}) \mid\mid \ldots \mid\mid (b_{p_i+r^i-1}, X_{p_i+r^i-1})$.

3. Repeat Steps 15c-15d of the encoding algorithm using shared randomness $\mathbf{R}$ and inputs $\mathbf{B}^{-i}$.

4. Skip the operations in the i-th partition and update the client storage of DS to be M^c.

5. Execute the operational sequence $\mathsf{write}(p_i + r^i, \mathbf{B}_{p_i+r^i}), \ldots, \mathsf{write}(m, \mathbf{B}_m)$ using DS and randomness $\mathbf{R}$. That is, execute all write operations after the i-th partition. When executing operations, if any memory location encoded in Y_i is accessed, then use the contents of Y_i to continue execution. For all memory accesses outside of Y_i use the current memory contents of DS to continue execution.

6. For each $j = p_i, \ldots, p_i + r^i - 1$:

    (a) If $b_j = 1$, then parse $\mathbf{B}_j$ as X_j and continue to the next iteration of the loop.

    (b) If $b_j = 0$, then parse the next X_j as the locations and contents of $\rho \cdot b/w$ memory cells in Q^j_i.

(c) Execute $\mathsf{read}(j)$ using DS and randomness $\mathbf{R}$. When executing the read operation, if any memory location encoded in Q_i^j is accessed, then use the encoded contents in Q_i^j. Otherwise, if any other memory location is accessed, use the current memory contents of DS.

(d) Parse $\mathbf{B}_j$ as the answer of $\mathsf{read}(j)$.

(e) Rewind DS to its state before executing the $\mathsf{read}(j)$.

7. Return $(\mathbf{B}_{p_i}, \dots, \mathbf{B}_{p_i+r^i-1})$.

Correctness. To show that encoding and decoding of $\mathbf{B}^i$ is successful, it suffices to show that both algorithms execute the operations for DS identically. Note, the execution of all write operations before the i-th partition are identical as both algorithms execute the same operations with the same randomness. Note that the encoder executes the write operations in the i-th partition while the decoder skips this subsequence of write operations. Both the encoder and decoder execute all operations after the i-th partition. We show that the decoder is able to execute identically to the encoder even though it is missing the write operations in the i-th partition. Consider the execution of any operation after the i-th partition by the decoder and any specific memory access. If the accessed memory location was not updated by any write operation in the i-th partition, then the decoder may simply use the current memory contents of DS to continue execution. If the memory location was modified during the i-th partition, we note that the contents are encoded by the encoder and used by the decoder. As a result, we can see that all operations after the i-th partition are executed identically by both the encoder and decoder. Therefore, we can see that the decoder always outputs the correct answer.

Length of Encoding. For the case that the encoding starts with a 1, we know that the encoding will always have bit length $1 + r^i \cdot b$. It remains us to upper bound the probability that the encoding starts with a 1 and the expected length of the encoding conditioned on it starting with a 0.

In this case, an encoding starts with c bits for client storage, $2\log n$ bits for $|Y_i|$ and $2|Y_i| \cdot w$ bits for Y_i. Note that $|Y_i| \le t_w \cdot n$ and thus at most $2\log n$ are needed to represent $|Y_i|$. Moreover, since i is a critical partition, $\mathbb{E}[|Y_i|] \le r^{i-1} \cdot b/w$ meaning the encoding of Y_i has expected size $2 \cdot r^{i-1} \cdot b$ as each memory cell location and contents can be encoded using $2w$ bits.

The length of the part of the encoding produced at the for loop of Step 15, conditioned on the encoding starting with 0, is upper bounded by $r^i + (r^i \cdot 2\rho \cdot b) + \frac{3r^i \cdot b}{4}$. To see this, note that we use 1 bit to distinguish encoding output at Step 15c from those output at Step 15d. Moreover at most $(3/4)$-fraction of operations will encode the b-bit block and the remainder will encode $\rho \cdot (b/w)$ memory locations using $2w$ bits each. By choosing $\rho < 1/32$ and $b \ge 16$ we have

$$r^i + (r^i \cdot 2\rho \cdot b) + \frac{3r^i \cdot b}{4} \le \frac{7r^i \cdot b}{8}.$$

Altogether, the expected length of encoding is upper bounded by

$$c + 2\log n + 2r^{i-1} \cdot b + \frac{7r^i \cdot b}{8} \le \frac{19r^i \cdot b}{20}$$

by choosing $r \ge 32$ and noticing critical partition i belongs to P and therefore $r^i \ge \max(100c/b, 200\log n/b)$ meaning that $c \le (1/100) \cdot r^i \cdot b$ and $2\log n \le (1/100) \cdot r^i \cdot b$.

Finally, we lower bound the probability that the encoding starts with a 0. By our assumption towards a contradiction, we know that $\mathcal{A}^{r,i}$ outputs 1 with probability at most 1/8 meaning that Q_i^j contains $\rho \cdot b/w$ locations with probability at least 7/8. As the error probability of DS is at most 1/3, both conditions are satisfied with probability at least $7/8 - 1/3 > 1/2$. By Markov's inequality, get that at least 1/4 of these queries will encode $\rho \cdot b/w$ locations with probability at least $z \geq 1/3$. Using this bound, we get the expected encoding length is at most

$$1 + (1-z) \cdot r^i \cdot b + z \cdot \frac{19 r^i \cdot b}{20} \leq 1 + \frac{59 r^i \cdot b}{60} < r^i \cdot b$$

assuming $b > 60$.

Applying Shannon's Source Coding Theorem. Finally, we can use the expected length of the encoding to derive our contradiction. From the above, we already know that the expected length of the encoding for the i-th partition is strictly smaller than $r^i \cdot b$. Note that the protocol enables successful encoding and decoding of $\mathbf{B}^i$ that is independent of all other shared randomness. Therefore,

$$H(\mathbf{B}^i \mid \mathbf{B}^{-i}, \mathbf{R}) = H(\mathbf{B}^i) = r^i \cdot b$$

as $\mathbf{B}^i$ consists of r^i uniformly random b-bit strings. This contradicts Shannon's source coding theorem stating that the expected length of any encoding scheme must be at least $r^i \cdot b$ completing the proof. □

Comparison with Prior Works. We note that our compression protocol differs from prior lower bound works [LN18, PY19, PY23, PPY20, KL21, LSY20] to handle the weaker nature of a snapshot adversary. In our scheme, we have to handle the case that the snapshot adversary is unable to observe the majority of memory accesses. Instead, the snapshot adversary can only view the differences in memory contents between snapshot attacks and the memory accesses of the final read operation. To accommodate this restriction, our communication protocol only sends locations and contents of memory cells whose contents have changed. In contrast, prior works would send the locations and contents of memory cells that are accessed even if their contents were unchanged, which was wasteful.

4.3 Completing the proof

To complete the proof, we will utilize Lemma 3 from the prior section that says that $\mathcal{A}^{r,i}$ is likely to output 1 with high probability when running experiment $\mathsf{Expt}_{\mathsf{DS},\mathcal{A}^{r,i}}(n,1)$ for at least (99/100)-fraction of partitions with at least $100c/b$ operations. By the snapshot privacy of DS, this immediately implies that $\mathcal{A}^{r,i}$ should also output 1 when running $\mathsf{Expt}_{\mathsf{DS},\mathcal{A}^{r,i}}(n,0)$. Note that for all partitions i, the operational sequence O_0 that is executed is identical. As a result, we show that this operational sequence is the hard distribution for which we can prove our desired lower bound. We formalize this argument to complete the proof of our lower bound below.

Proof of Lemma 1. To prove the condition of the theorem, we will first assume that $t_r = o(b/w \cdot \log(nb/c))$ and show that $t_w = \Omega(b/w \cdot \log(nb/c))$.

First, we utilize Lemma 3 that states that our adversary outputs 1 with high probability when running the experiment with bit $\beta = 1$, we know that

$$\Pr[\mathsf{Expt}_{\mathsf{DS},\mathcal{A}^{r,i},3,1}(n,1) = 1] \geq 1/8$$

for the critical partitions that constitute a (99/100)-fraction of the partitions from the set P of all partitions with at least $100c/b$ operations. For convenience, denote $I \subseteq P$ to be the set of critical partitions.

Next, we apply the fact that DS is $(3, 1, \epsilon)$-snapshot private with $\epsilon \leq 1/16$. Note, this immediately implies that $\mathcal{A}^{r,i}$ must also output 1 when running the experiment with bit $\beta = 0$. Formally,

$$\Pr[\mathsf{Expt}_{\mathsf{DS},\mathcal{A}^{r,i}}(n, 0) = 1] \geq 1/8 - \epsilon \geq 1/16 \tag{1}$$

for all critical partitions i.

Finally, we translate the above probabilities into the overhead of the final read operation. Note that $\mathsf{Expt}_{\mathsf{DS},\mathcal{A}^{r,i}}(n, 0) = 1$ means that $\mathcal{A}^{r,i}$ output 1. This only occurs when the final read operation of O_0 accesses at least $\rho \cdot b/w$ memory locations that were last overwritten in the i-th partition. Furthermore, note that the randomized operational sequence O_0 produced by $\mathcal{A}^{r,i}$ is the same regardless of the choice of the partition i. Formally, recall that $\mathcal{A}^{r,i}$ defines the following:

- $U_i = \{j \in [|M|] \mid M_{p_i}[j] \neq M_{p_i+r^i}[j]\}$.
- $V_i = \{j \in [|M|] \mid M_{p_i+r^i}[j] \neq M_{m+1}[j]\}$.
- $W_i = U_i \setminus V_i$.

where W_i ends up being the set of memory locations that were last updated by write operations in the i-th partition. $\mathcal{A}^{r,i}$ only outputs 1 if the intersection of memory accesses by the final read operation $\mathcal{T}_m$ and W_i is at least $\rho \cdot b/w$. That is, $|\mathcal{T}_{m+1} \cap W_i| \geq \rho \cdot b/w$. Note that one way to compute the overhead of the final read operation is to lower bound the expected size of the set

$$\bigcup_{i \in P} (\mathcal{T}_{m+1} \cap W_i) \,.$$

The main observation is that each of W_i are disjoints. Therefore, we can lower bound the number of memory accesses by the final read operation as

$$\begin{aligned}
t_r &\geq \mathbb{E}\left[\left|\bigcup_{i \in P} (\mathcal{T}_{m+1} \cap W_i)\right|\right] \\
&\geq \sum_{i \in I} \mathbb{E}\left[|\mathcal{T}_{m+1} \cap W_i|\right] \\
&\geq \sum_{i \in I} \mathbb{E}\left[|\mathcal{T}_{m+1} \cap W_i| \mid \mathsf{Expt}_{\mathsf{DS},\mathcal{A}^{r,i}}(n, 0) = 1\right] \\
&\geq \frac{99}{100} \cdot |P| \cdot \Pr[\mathsf{Expt}_{\mathsf{DS},\mathcal{A}^{r,i}}(n, 0) = 1] \cdot \rho \cdot b/w \\
&= \Omega(b/w \cdot \log(nb/c))
\end{aligned}$$

as we know that $|I| \geq (99/100) \cdot |P|$, $|P| = \Theta(\log(nb/c))$ and $\rho = \Theta(1)$. □

5 Extensions of Our Lower Bound

In this section, we strengthen our lower bound for various weaker notions of snapshot security. We present definitions of these weaker notions and the modifications to our proof necessary for proving the lower bound.

5.1 Differential Privacy

We start by defining differential privacy with respect to (s, ℓ)-snapshot adversaries $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$. For this notion, we force the adversary to output challenge operational sequences that differ in only one position.

Definition 2 (Differentially Private $(s, \ell, \epsilon, \delta)$-Snapshot Security)**.** *Let $s \geq 1$ and $\ell \geq 0$ be fixed integers and let $0 \leq \epsilon, \delta < 1$. A data structure DS is $(s, \ell, \epsilon, \delta)$-snapshot secure if for any PPT (s, ℓ)-snapshot adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ such that $\mathcal{A}_0$ outputs two challenge sequences that differ in only one operation, the following holds for all $b \in \{0, 1\}$:*

$$\Pr[\mathsf{Expt}_{\mathsf{DS},\mathcal{A}}(n, b) = 1] \leq e^{\epsilon} \Pr[\mathsf{Expt}_{\mathsf{DS},\mathcal{A}}(n, 1 - b) = 1] + \delta,$$

for all sufficiently large n.

The above is a weaker notion of the snapshot security that was defined in Definition 1. For example, if we set $\epsilon = 0$ and $\delta = \mathsf{negl}(n)$, then we obtain a notion that is equivalent to standard snapshot security with negligible adversarial advantage. For any constant $\epsilon > 0$ and $\delta > 0$, the above definition seems weaker. However, we show that ORAMs satisfying this definition still requires logarithmic overhead.

Theorem 3. *For any $0 \leq \epsilon \leq 1$ and any $0 \leq \delta \leq 1/16$, let DS be a differentially private $(3, 1, \epsilon, \delta)$-snapshot secure RAM data structure for n entries each of $b \geq 1$ bits implemented over $w = \Omega(\log n)$ bits using client storage of $c \geq 1$ bits in the cell probe model. If DS has amortized write time t_w and expected amortized read time t_r with failure probability at most $1/3$, then*

$$t_r + t_w = \Omega\left(b/w \cdot \log(nb/c)\right).$$

Proof. The proof proceeds identically to Theorem 1 with the only difference being the snapshot security guarantees. The only modification occurs when when we lower bound the advantage of the adversary in Equation 1. We can replace this with the following series of inequalities:

$$\begin{aligned} e^{\epsilon} \cdot \Pr[\mathsf{Expt}_{\mathsf{DS},\mathcal{A}^{r,i}}(n, 0) = 1] + \delta &\geq \Pr[\mathsf{Expt}_{\mathsf{DS},\mathcal{A}^{r,i}}(n, 1) = 1] \\ \Pr[\mathsf{Expt}_{\mathsf{DS},\mathcal{A}^{r,i}}(n, 0) = 1] &\geq (1/8 - \delta)/e^{\epsilon} \\ \Pr[\mathsf{Expt}_{\mathsf{DS},\mathcal{A}^{r,i}}(n, 0) = 1] &\geq 1/(16e). \end{aligned}$$

In other words, we can also lower bound the adversarial advantage by some constant factor that is sufficient for completing the proof. □

5.2 Read-Only Obliviousness

Next, we consider a weaker privacy notion where obliviousness is only to be provided for ORAM read operations. In particular, the adversary is not allowed to see the transcripts for any ORAM write operations. This may make sense when write operations are public information, but read operations are sensitive (e.g., a medical research database).

To define read-only ORAMs, we simply modify the experiment to only return memory access leakage for ORAM write operations. We present the experiment in Figure 3.

```
$\mathsf{ROExpt}_{\mathsf{DS},\mathcal{A}}(n,\beta)$:

  1. Execute $(O_0, O_1, S, \mathtt{st}) \leftarrow \mathcal{A}_0(1^n)$.
  2. Parse $S = \{(t_1, \ell_1), \ldots, (t_{|S|}, \ell_{|S|})\}$.
  3. Set leakage $\mathcal{L} \leftarrow \emptyset$ and initialize $\mathsf{DS}$ with $n$ blocks each consisting of randomly select
     blocks of $b$ bits.
  4. Set $i \leftarrow 1$.
  5. While $i \le |O_\beta|$:
     (a) If $i = t_j$ for some $j \in [|S|]$:
         i. Set $\mathcal{L} \leftarrow \mathcal{L} \,||\, (\mathsf{memory}, M)$.
         ii. For $k = 1, \ldots, \ell_j$:
             A. **If $O_b[i]$ is a read operation, execute $\mathsf{DS}^{\mathsf{LRead},\mathsf{LWrite}}(O_b[i])$.**
             B. **If $O_b[i]$ is a write operation, execute $\mathsf{DS}^{\mathsf{Read},\mathsf{Write}}(O_b[i])$.**
             C. Set $i \leftarrow i + 1$.
     (b) Else:
         i. Execute $\mathsf{DS}^{\mathsf{Read},\mathsf{Write}}(O_b[i])$.
         ii. Set $i \leftarrow i + 1$.
  6. Execute $\beta' \leftarrow \mathcal{A}_1(\mathtt{st}, \mathcal{L})$.
  7. Return $\beta'$.
```

Figure 3: Experiment for read-only (s, ℓ)-snapshot security. Bolded parts denote differences between prior snapshot security.

Definition 3 (Read-Only (s, ℓ, ϵ)-Snapshot Security)**.** *Let $s \ge 1$ and $\ell \ge 0$ be fixed integers and let $0 \le \epsilon < 1$. A data structure* DS *is (s, ℓ, ϵ)-snapshot private if for any PPT (s, ℓ)-snapshot adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ such that $\mathcal{A}_0$ outputs two challenge sequences with the same number of read operations and the following holds:*

$$\left|\Pr[\mathsf{ROExpt}_{\mathsf{DS},\mathcal{A}}(n, 0) = 1] - \Pr[\mathsf{ROExpt}_{\mathsf{DS},\mathcal{A}}(n, 1) = 1]\right| \le \epsilon,$$

for all sufficiently large n.

We show that $\Omega(\log n)$ overhead is required ORAMs that are persistent even against $(3, 1)$-snapshot adversaries with read-only operational transcripts.

Theorem 4. *For any $0 \le \epsilon \le 1/16$, let* DS *be a read-only $(3, 1, \epsilon)$-snapshot secure RAM data structure for n entries each of $b \ge 1$ bits implemented over $w = \Omega(\log n)$ bits using client storage of $c \ge 1$ bits in the cell probe model. If* DS *has amortized write time t_w and expected amortized read time t_r with failure probability at most $1/3$, then*

$$t_r + t_w = \Omega\left(b/w \cdot \log(nb/c)\right).$$

Proof. The only difference is that we must apply read-only snapshot security as opposed to standard snapshot security from Definition 1. Note that our choice of snapshot adversary from Section 4.1 outputs two operational sequences with exactly one ORAM read operation. Furthermore, the transcript viewed by the snapshot adversary is always the single ORAM read operation. Therefore, our snapshot adversary from Section 4.1 is also a read-only snapshot adversary and the lower bound holds without modification. □

5.3 Structured Encryption Leakage

Finally, we show the last extension using leakage functions that are common in structured encryption and encrypted search. First, we define snapshot security with respect to a leakage function $\mathcal{L}$.

Definition 4 ($\mathcal{L}$-Leakage (s,ℓ,ϵ)-Snapshot Security)**.** *Let $s \geq 1$ and $\ell \geq 0$ be fixed integers and let $0 \leq \epsilon < 1$. A data structure DS is $\mathcal{L}$-leakage $(s,\ell,\epsilon,\mathcal{L})$-snapshot secure if for any PPT (s,ℓ)-snapshot adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ such that $\mathcal{A}_0$ outputs two challenge sequences O_0 and O_1 satisfying $\mathcal{L}(O_0) = \mathcal{L}(O_1)$ and the following holds:*

$$\left|\Pr[\mathsf{Expt}_{\mathsf{DS},\mathcal{A}}(n,0) = 1] - \Pr[\mathsf{Expt}_{\mathsf{DS},\mathcal{A}}(n,1) = 1]\right| \leq \epsilon,$$

for all sufficiently large n.

In other words, standard snapshot security applies but only for pairs of operational sequences O_0 and O_1 with the same leakage according to $\mathcal{L}$, that is, $\mathcal{L}(O_0) = \mathcal{L}(O_1)$. By setting the leakage to be the length of the operational sequence $\mathcal{L}(O) = |O|$, we obtain standard snapshot security from Definition 1.

Next, we present two leakage functions: key-equality and decoupled key-equality. At a high level, key-equality returns whether any pair of operations in a sequence are operating on the same index of the ORAM. On the other hand, decoupled key-equality leaks whether any pair of operation of the same type (i.e., read or write operations) are operating on the same index of the ORAM. We define both below:

Definition 5 (Key-Equality Leakage)**.** *Key-equality leakage $\mathcal{L}_{KE}$ for any operational sequence O outputs a $|O| \times |O|$ matrix M with the following properties:*

$$M[i][j] = \begin{cases} 0, & \textit{if the } i\textit{-th and } j\textit{-th operation are for different keys;} \\ 1, & \textit{if the } i\textit{-th and } j\textit{-th operation are for the same key.} \end{cases}$$

There are straightforward $O(1)$ constructions that $\mathcal{L}$-leakage security against even persistent adversaries that may be adapted from structured encryption and encrypted search literature (such as [CGKO06, CJJ$^+$14]). We can also consider a slightly smaller leakage function denoted as decoupled-key equality introduced in [PPY20].

Definition 6 (Decoupled Key-Equality Leakage [PPY20])**.** *Decoupled key-equality leakage $\mathcal{L}_{DKE}$ for any operational sequence O outputs two $|O| \times |O|$ matrices M_R and M_W with the following properties:*

$$M_R[i][j] = \begin{cases} \perp, & \textit{if one of the } i\textit{-th and } j\textit{-th operation is an ORAM write;} \\ 0, & \textit{if the } i\textit{-th and } j\textit{-th operation are ORAM reads for different keys;} \\ 1, & \textit{if the } i\textit{-th and } j\textit{-th operation are ORAM reads for the same key;} \end{cases}$$

$$M_W[i][j] = \begin{cases} \bot, & \textit{if one of the } i\textit{-th and } j\textit{-th operation is an ORAM read;} \\ 0, & \textit{if the } i\textit{-th and } j\textit{-th operation are writes for different keys;} \\ 1, & \textit{if the } i\textit{-th and } j\textit{-th operation are writes for the same key;} \end{cases}$$

We show that satisfying decoupled key-equality leakage against even $(3, 1)$-snapshot adversaries requires $\Omega(\log n)$ overhead for ORAMs.

Theorem 5. *For any $0 \leq \epsilon \leq 1/16$, let DS be a $\mathcal{L}_{DKE}$-leakage $(3, 1, \epsilon)$-snapshot secure RAM data structure for n entries each of $b \geq 1$ bits implemented over $w = \Omega(\log n)$ bits using client storage of $c \geq 1$ bits in the cell probe model. If DS has amortized write time t_w and expected amortized read time t_r with failure probability at most $1/3$, then*

$$t_r + t_w = \Omega\left(b/w \cdot \log(nb/c)\right).$$

Proof. Similar to the read-only proof, we will show that the snapshot adversary from Section 4.1 always outputs pairs of operational sequences whose leakage is identical with respect to the decoupled key-equality leakage function $\mathcal{L}_{DKE}$. Recall that the first $\Theta(n)$ ORAM write operations are the same for both sequences and only the last ORAM read operation differs. As there is only ORAM read operation, the decoupled key-equality leakage for read operations is trivial. For the write operations, we note that there is no repeated indices. Therefore, the entire matrix for write operations will consist of 0. As a result, we see that the snapshot adversary from Section 4.1 always outputs operations with identical decoupled key-equality leakage. As a result, we can proceed with the proof identically to Theorem 1. □

6 Snapshot Oblivious RAMs

In this section, we study no-write snapshot ORAMs that only enable clients to read entries. We present a very simple $O(1)$ construction secure against $(s, 1)$-snapshot adversaries for any $s \geq 1$ providing a separation with our lower bound for read-write ORAMs. Such a separation is not known between no-write and read-write ORAMs with respect to persistent adversaries. We also survey prior works with slightly different security notions and show that they also provide security against $(s, 0)$-snapshot adversaries.

6.1 No-Write Snapshot ORAMs

In a no-write ORAM, the client can only access the data but cannot overwrite any entries. Constructing a no-write ORAM secure against $(s, 0)$-snapshot attacks for every $s \geq 1$ is straightforward. The client uploads all n entries in encrypted form to the server with an encryption key stored by the client. To query for the i-th entry, the client simply downloads the i-th encrypted entry and decrypts to retrieve the i-th entry. Since no block is overwritten, the snapshot adversary receives the same server memory for any sequence of operations. This construction can be upgraded to security against $(s, 1)$-snapshot adversary, for every $s \geq 1$. To do this, the client simply samples a random key for a pseudorandom permutation (PRP) and uploads the encrypted n entries in a permuted manner. Retrieving the i-th entry becomes retrieving the i-th entry according to the PRP. Even though the adversary observes the transcript of a single ORAM read operation, we note that the transcript simply contains a memory read to a random entry.

Theorem 6. *Assuming the existence of one-way functions, there exists a no-write ORAM with $O(1)$ overhead that is secure against $(s, 1)$-snapshot attacks for any $s \geq 1$.*

The theorem above must be contrasted with the our current understanding of the overhead needed for no-write ORAMs secure against persistent adversaries. Specifically, the current best construction for no-write ORAMs is the generic $O(\log n)$ overhead construction for ORAMs supporting both read and write operations. On the lower bound front, Weiss and Wichs [WW18] proved that showing lower bounds for no-write ORAMs even against persistent adversaries will give corresponding lower bounds on sorting circuit size or on the query complexity and size of locally decodable codes. Both are long-standing, important open problems in the area of computational complexity.

6.2 Snapshot ORAMs from Prior Works

Next, we show that prior works have studied slightly different privacy notions that may be reinterpreted as snapshot security.

Breach-Resistant Structured Encryption. Amjad, Kamara, and Moataz [AKM19] introduced the notion of a breach-resistant structured encryption. A structured encryption scheme (STE) encrypts a data structures so that it can be privately queried. Special cases of STE include graphs, dictionaries, multi-maps and RAMs that are the primary focus of this paper. They consider adversaries that only perform breaches to see the contents of server memory. Their goal is to design breach *breach-resistant* multi-maps where only the size of the underlying data is revealed to the adversary. If we restrict the multi-map to be an array, we can view their work as constructing ORAMs that are secure against $(s, 0)$-snapshot adversaries for some $s \geq 1$. However, the snapshot adversary considered in [AKM19] is stronger[2] than ours as it is allowed to adaptively decide on the next batch of operations after having seen the leakage from the previous batch. More precisely, the snapshot adversary considered in [AKM19] works in rounds. At the start of each round, the adversary receives a snapshot of the memory and decides for the next round of operations to be executed. The following result is implicit in [AKM19]:

Theorem 7 ([AKM19])**.** *If one-way functions exist, then there exists an ORAM with $O(\log n)$ overhead that is secure against $(s, 0)$-snapshot adversary for any $s \geq 0$.*

For completeness, we will present the construction of [AKM19] adapted to our terminology in Appendix A.

Write-Only ORAMs. The concept of a write-only ORAM (see [LD13, BMNO14, RACM17]) is closely related to that of snapshot security. A write-only ORAM relaxes the security notion of an ORAM by requiring only the write operations to be oblivious with respect to a persistent adversary. In other words, the adversary receives the access pattern to server memory for all write operations. This is somewhat similar to security with respect to $(s, 0)$-snapshot adversary. In write-only ORAMs, the view of the adversary is filtered based upon the type of the logical ORAM operation type whereas in $(s, 0)$-snapshot attack the filtering occurs at the physical level (whether a server memory location is updated or not).

[2]As the main goal of our paper was the lower bound, we only considered weaker non-adaptive snapshot adversaries since weaker adveraries imply stronger lower bounds. Although, we note that all the constructions presented in our paper can also be proven secure against adaptive snapshot adversaries.

The work of Amjad, Kamara and Moataz [AKM19] proved that breach-resistant arrays implied write-only obliviousness. In this work, we show that write-only obliviousness under certain restrictions also implies $(s, 0)$-snapshot security for any $s \geq 1$. Specifically, we assume that ORAM read operations perform no physical writes to server memory. We view this as a mild assumption as the most efficient write-only ORAMs satisfy this property (see [RACM17]).

Theorem 8. *Any write-only ORAM that implements logical ORAM read operations with no physical writes to server memory during read operations is secure against $(s, 0)$-snapshot adversaries for any $s \geq 1$.*

Proof. As any logical ORAM read operations do not modify server memory, the leakage viewed by a $(s, 0)$-snapshot adversary after ORAM read operations is trivial. Furthermore, by definition, write-only ORAMs are oblivious for any logical write operations. Therefore, any write-only ORAM with the condition that ORAM read operations perform no server memory overwrites is also $(s, 0)$-snapshot secure. □

Using the above, we can see that prior write-only ORAM constructions are also secure against $(s, 0)$-snapshot adversaries. To our knowledge, the DetWoORAM construction [RACM17] satisfies the conditions of the above theorem and is the most efficient write-only ORAM to date. Therefore, we get that:

Theorem 9 ([RACM17])**.** *If one-way functions exist, then there exists an ORAM with $O(\log n)$ overhead that is secure against $(s, 0)$-snapshot adversary for any $s \geq 0$.*

State-of-the-Art for $(s, 0)$-Snapshot Security. To our knowledge, the above constructions for breach-resistance and write-only ORAMs achieve the best concrete overhead for $(s, 0)$-snapshot ORAMs. One could also use standard ORAMs secure against persistent adversaries [AKL+20] to also obtain $O(\log n)$ overhead. As our lower bounds do not apply for $(s, 0)$-snapshot security, we leave it as an open to resolve the optimal overhead for $(s, 0)$-snapshot security.

7 Snapshot Oblivious Stacks and Queues

In this section, we show that we can also construct stacks and queues with constant overhead that is secure against $(s, 1)$-snapshot adversaries for any $s \geq 1$. This provides a separation between snapshot oblivious RAMs and stacks/queues. In contrast, oblivious RAMs and stacks/queues require $\Omega(\log n)$ overhead against persistent adversaries [LN18, JLN19]. Throughout, we will describe constructions only for stacks, but it is trivial to derive queue (or even deque) constructions.

First, we will show a simple construction that obtains $(s, 0)$-snapshot security for any $s \geq 1$. Afterwards, we show that a simple trick using permutations would enable $(s, 1)$-snapshot security. We chose to present our construction this way to showcase that it seems easy to upgrade $(s, 0)$-snapshot security to $(s, 1)$-snapshot security. This is a similar approach that was taken for no-write ORAMs in Section 6.1 as well.

7.1 $(s, 0)$-Snapshot Secure Oblivious Stack

We start by describing an implementation of the stack data structure that is secure with respect to an $(s, 0)$-snapshot adversary for any $s \geq 1$. That is, an adversary that receives an initial snapshot of the server memory and the snapshot of the memory after each operation. The server memory consists of n locations indexed $0, \ldots, n-1$. All memory is initialized to contain the all 0-bit string. The client memory will consists of two components: an IND-CPA encryption key, `cnt` to track the number of operations and `top` to track the current server index of the top of the stack. Next, we formally describe the stack algorithms for initialization `Init`, pushing an item `Push` and popping an item `Pop`.

- $\texttt{Init}(1^\lambda)$: Initialization sets the two variables `cnt` and `top` to 0 and -1 respectively and also randomly generates a λ-bit encryption key $K \leftarrow \{0,1\}^\lambda$ for an IND-CPA encryption scheme. All three are stored in client memory.

- $\texttt{Push}(v)$: To push the value v, the clients encrypts $\mathsf{Enc}(K, (v, \texttt{top}))$ and uploads it to the location with index `cnt`. This pair consists of the value v and a pointer to the previous top of the stack. Finally, `top` is set equal to `cnt` and `cnt` is incremented.

- $\texttt{Pop}()$: To pop the top of the stack, the client downloads and decrypts the pair at the server location `top` to obtain $(v, \texttt{pTop})$ consisting of the value at the top of the stack and a pointer to the previous top of the stack. Next, a dummy pair is encrypted $\mathsf{Enc}(K, (\perp, \perp))$ and uploaded to server memory at location `cnt`. Finally, `top` is set equal to `pTop` and `cnt` is incremented. The value v is returned.

We show that the above construction is secure against any $(s, 0)$-snapshot adversary for any choice of $s \geq 1$.

Theorem 10. *Assuming one-way functions exist, there exists an oblivious stack with $O(1)$ overhead that is secure against $(s, 0)$-snapshot adversaries for any $s \geq 1$.*

Proof. The above stack construction clearly has $O(1)$ overhead as it only downloads and updates $O(1)$ entries.

To argue security, we simply need to focus on the server locations that are updated as the $(s, 0)$-snapshot adversary does not observe operational transcripts. For each operation (regardless of `Push` or `Pop`), the stack will always encrypt and upload a pair to the server memory location indexed by `cnt`. By IND-CPA security, we know that the encryption is indistinguishable from random. Recall that `cnt` is the number of operations that have been executed. Therefore, all operational sequences of the same length will result in the same view for a $(s, 0)$-snapshot adversary for any $s \geq 1$.

As a final remark, we observe that the `Pop` operation downloads the encrypted pair stored at index `top` whereas `Push` do not. However, the location is not re-written and this different behavior between operations goes undetected to an $(s, 0)$-snapshot adversary that does not observe operational transcripts. $\square$

7.2 Upgrading to $(s, 1)$-Snapshot Security

Next, we upgrade the construction from the previous section to $(s, 1)$-snapshot security that includes adversaries that also obtain the transcript of one operation of their choice. The previous construction is not $(s, 1)$-snapshot secure because the memory accesses in the `Pop` operation reveals the size of the real stack (and, thus, reveals the number of `Push` and `Pop` operations total).

To remedy this, we simply permute memory contents according to a pseudorandom permutation F generated using a randomly selected λ-bit seed K_π. We modify the stack so that it accesses locations $F(K_\pi, \texttt{top})$ and $F(K_\pi, \texttt{cnt})$ instead of `top` and `cnt`. Even if an adversary views the memory access of a `Pop` operation, the resulting read memory is essentially uniformly random enabling $(s, 1)$-snapshot security. More precisely, we have the following modifications:

- $\texttt{Init}(1^\lambda)$ proceeds identically as before, but also randomly selects a λ-bit seed $K_\pi \leftarrow \{0,1\}^\lambda$ for a PRP family of functions F. The key K_π is stored in client memory.
- $\texttt{Push}(v)$ must be modified in two ways. First, the client must also read a memory location to be indistinguishable from `Pop` operations. To do this, the client will download the encrypted pair currently stored at $F(K_\pi, \texttt{top})$. The downloaded result is discarded immediately. Secondly, the encrypted $\mathsf{Enc}(K, (v, \texttt{top})$ is stored at server location $F(K_\pi, \texttt{cnt})$ as opposed to `cnt`.
- $\texttt{Pop}()$ must be modified in two ways as well. First, the top of the stack is downloaded from server location $F(K_\pi, \texttt{top})$ as opposed to just `top`. Additionally, the encrypted dummy pair $\mathsf{Enc}(K, (\perp, \perp))$ is stored at location $F(K_\pi, \texttt{cnt})$ as opposed to `cnt`.

Theorem 11. *Assuming one-way functions exist, there exists an oblivious stack with $O(1)$ overhead that is secure against $(s, 1)$-snapshot adversaries for any $s \geq 1$.*

Proof. The proof follows from Theorem 10 for $(s, 0)$-snapshot security. To see security against $(s, 1)$-snapshot security, we observe that the operational transcript for any `Push` or `Pop` operation involves reading a random location and updating a random location with a random entry due to the security of the underlying PRP and IND-CPA encryption schemes. □

Remark about $(2, 2)$-snapshot attacks. We note that the construction above ceases to be snapshot-secure if the adversary is allowed to see the transcript of two operations. The attack relies on the fact that both `Push` and `Pop` download the current top of the stack. More precisely, consider sequences $O_1 = (\texttt{Push}, \texttt{Push}, \texttt{Pop}, \texttt{Push})$ and $O_2 = (\texttt{Push}, \texttt{Push}, \texttt{Push}, \texttt{Push})$. Note that in O_1, but not in O_2, the top of the stack is at the same location before the second operation and before the fourth operation. This is so because the third operation in O_1 is a `Pop` and "undoes" the previous push. The third operation in O_2 instead is a push and the top does not go back. In other words, even though for any single operation the accessed memory location is random, the memory locations accessed by two operations might not be independent.[3]

8 Conclusions and Open Problems

In this work, we present a negative answer to the open problem posed in [DGG22] of whether it is possible to build sub-logarithmic ORAM constructions secure against multiple breaches. We present a $\Omega(\log n)$ lower bound for ORAMs secure against $(3, 1)$-snapshot adversaries. In other words, we show that protecting against three breaches is as challenging as protecting against a persistent adversary. Furthermore, we prove some separations by presenting $O(1)$ overhead constructions for no-write ORAMs and oblivious stacks/queues secure against $(s, 1)$-snapshot adversaries for any $s \geq 1$. We leave the following open problems:

[3]A prior version discussed potential obstacles of obtaining $(s, 1)$-snapshot security and contained informal claims about compiling a $(s, 0)$-snapshot ORAMs into a $(s, 1)$-snapshot ORAMs under some conditions. To the best of our knowledge, none of the $(s, 0)$-snapshot ORAMs from the literature satisfy these conditions. We thank Wei-Kai Lin for raising this point.

- Our work rules out sub-logarithmic overhead against $(3,1)$-snapshot adversaries with three data breaches. However, it remains open to resolve the correct overhead for snapshot adversaries with two data breaches. Is it possible to construct a $o(\log n)$ overhead ORAM construction to protect against $(2,\ell)$-snapshot adversaries for any $\ell \geq 0$?
- The lower bound presented in our work shows that snapshot adversaries are as powerful as persistent adversaries if they can see the transcript of one operation. It is also natural to consider adversaries that never see the transcript of any operation. Unfortunately, current constructions from prior works [RACM17, AKM19] still require $O(\log n)$ overhead to protect against $(s,0)$-snapshot adversaries. Is it possible to construct a $o(\log n)$ ORAM that is secure against $(s,0)$-snapshot adversaries for any $s \geq 2$?
- Are there any other meaningful weakenings of security for ORAMs that admit sub-logarithmic constructions while still enabling wide applicability in practical applications?

Acknowledgements. The authors would like to thank Yang Du and Paul Grubbs for posing the question of multiple data breaches and Wei-Kai Lin and Daniel Wichs for initial discussions about the problem.

References

[ACD+22] Erik Aronesty, David Cash, Yevgeniy Dodis, Daniel H Gallancy, Christopher Higley, Harish Karthikeyan, and Oren Tysor. Encapsulated search index: Public-key, sub-linear, distributed, and delegatable. In *Public-Key Cryptography–PKC 2022: 25th IACR International Conference on Practice and Theory of Public-Key Cryptography, Virtual Event, March 8–11, 2022, Proceedings, Part II*, pages 256–285. Springer, 2022.

[AKL+20] Gilad Asharov, Ilan Komargodski, Wei-Kai Lin, Kartik Nayak, Enoch Peserico, and Elaine Shi. OptORAMa: Optimal oblivious RAM. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 403–432. Springer, Heidelberg, May 2020.

[AKL+22] Gilad Asharov, Ilan Komargodski, Wei-Kai Lin, Enoch Peserico, and Elaine Shi. Optimal oblivious parallel ram. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2459–2521. SIAM, 2022.

[AKLS21] Gilad Asharov, Ilan Komargodski, Wei-Kai Lin, and Elaine Shi. Oblivious RAM with worst-case logarithmic overhead. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 610–640, Virtual Event, August 2021. Springer, Heidelberg.

[AKM19] Ghous Amjad, Seny Kamara, and Tarik Moataz. Breach-resistant structured encryption. *PoPETs*, 2019(1):245–265, January 2019.

[APP+23] Ghous Amjad, Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. Dynamic volume-hiding encrypted multi-maps with applications to searchable encryption. *PoPETs (to appear)*, 2023.

[BBF+21] Angèle Bossuat, Raphael Bost, Pierre-Alain Fouque, Brice Minaud, and Michael Reichle. SSE and SSD: Page-efficient searchable symmetric encryption. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 157–184, Virtual Event, August 2021. Springer, Heidelberg.

[BCP16] Elette Boyle, Kai-Min Chung, and Rafael Pass. Oblivious parallel RAM and applications. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part II*, volume 9563 of *LNCS*, pages 175–204. Springer, Heidelberg, January 2016.

[BDOP04] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 506–522. Springer, Heidelberg, May 2004.

[BF19] Raphael Bost and Pierre-Alain Fouque. Security-efficiency tradeoffs in searchable encryption. *PoPETs*, 2019(4):132–151, October 2019.

[BIM04] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers' computation in private information retrieval: PIR with preprocessing. *Journal of Cryptology*, 17(2):125–151, March 2004.

[BKM20] Laura Blackstone, Seny Kamara, and Tarik Moataz. Revisiting leakage abuse attacks. In *NDSS 2020*. The Internet Society, February 2020.

[BMNO14] Erik-Oliver Blass, Travis Mayberry, Guevara Noubir, and Kaan Onarlioglu. Toward robust hidden volumes using write-only oblivious RAM. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014*, pages 203–214. ACM Press, November 2014.

[BMO17] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1465–1482. ACM Press, October / November 2017.

[BN16] Elette Boyle and Moni Naor. Is there an oblivious RAM lower bound? In Madhu Sudan, editor, *ITCS 2016*, pages 357–368. ACM, January 2016.

[Bos16] Raphael Bost. $\Sigma o\phi o\varsigma$: Forward secure searchable encryption. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1143–1154. ACM Press, October 2016.

[bre] List of data breaches. https://en.wikipedia.org/wiki/List_of_data_breaches.

[CDH20] David Cash, Andrew Drucker, and Alexander Hoover. A lower bound for one-round oblivious RAM. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 457–485. Springer, Heidelberg, November 2020.

[CGKO06] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 79–88. ACM Press, October / November 2006.

[CGPR15] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 668–679. ACM Press, October 2015.

[CHK22] Henry Corrigan-Gibbs, Alexandra Henzinger, and Dmitry Kogan. Single-server private information retrieval with sublinear amortized time. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 3–33. Springer, Heidelberg, May / June 2022.

[CJJ$^+$13] David Cash, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for Boolean queries. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 353–373. Springer, Heidelberg, August 2013.

[CJJ+14] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *NDSS 2014*. The Internet Society, February 2014.

[CK10] Melissa Chase and Seny Kamara. Structured encryption and controlled disclosure. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 577–594. Springer, Heidelberg, December 2010.

[CK20] Henry Corrigan-Gibbs and Dmitry Kogan. Private information retrieval with sublinear online time. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 44–75. Springer, Heidelberg, May 2020.

[CLT16] Binyi Chen, Huijia Lin, and Stefano Tessaro. Oblivious parallel RAM: Improved efficiency and generic constructions. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part II*, volume 9563 of *LNCS*, pages 205–234. Springer, Heidelberg, January 2016.

[CT14] David Cash and Stefano Tessaro. The locality of searchable symmetric encryption. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 351–368. Springer, Heidelberg, May 2014.

[dbi] DBIR: Data Breach Investigations Report. https://www.verizon.com/business/resources/T61c/reports/dbir/2022-data-breach-investigations-report-dbir.pdf.

[DGG22] Yang Du, Daniel Genkin, and Paul Grubbs. Snapshot-oblivious RAMs: Sub-logarithmic efficiency for short transcripts. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 152–181. Springer, Heidelberg, August 2022.

[DPPS20] Ioannis Demertzis, Dimitrios Papadopoulos, Charalampos Papamanthou, and Saurabh Shintre. SEAL: Attack mitigation for encrypted databases via adjustable leakage. In Srdjan Capkun and Franziska Roesner, editors, *USENIX Security 2020*, pages 2433–2450. USENIX Association, August 2020.

[Ds17] Jack Doerner and abhi shelat. Scaling ORAM for secure computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 523–535. ACM Press, October / November 2017.

[FS89] Michael L. Fredman and Michael E. Saks. The cell probe complexity of dynamic data structures. In *21st ACM STOC*, pages 345–354. ACM Press, May 1989.

[GKL+20] Paul Grubbs, Anurag Khandelwal, Marie-Sarah Lacharité, Lloyd Brown, Lucy Li, Rachit Agarwal, and Thomas Ristenpart. Pancake: Frequency smoothing for encrypted data stores. In Srdjan Capkun and Franziska Roesner, editors, *USENIX Security 2020*, pages 2451–2468. USENIX Association, August 2020.

[GKM21] Marilyn George, Seny Kamara, and Tarik Moataz. Structured encryption and dynamic leakage suppression. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part III*, volume 12698 of *LNCS*, pages 370–396. Springer, Heidelberg, October 2021.

[GLMP19] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. Learning to reconstruct: Statistical learning theory and encrypted database attacks. In *2019 IEEE Symposium on Security and Privacy*, pages 1067–1083. IEEE Computer Society Press, May 2019.

[GMOT12] Michael T. Goodrich, Michael Mitzenmacher, Olga Ohrimenko, and Roberto Tamassia. Privacy-preserving group data access via stateless oblivious RAM simulation. In Yuval Rabani, editor, *23rd SODA*, pages 157–167. ACM-SIAM, January 2012.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the ACM (JACM)*, 1996.

[HK14] Florian Hahn and Florian Kerschbaum. Searchable encryption with secure and efficient updates. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014*, pages 310–320. ACM Press, November 2014.

[HKKS19] Pavel Hubácek, Michal Koucký, Karel Král, and Veronika Slívová. Stronger lower bounds for online ORAM. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part II*, volume 11892 of *LNCS*, pages 264–284. Springer, Heidelberg, December 2019.

[IKK12] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *NDSS 2012*. The Internet Society, February 2012.

[JLN19] Riko Jacob, Kasper Green Larsen, and Jesper Buus Nielsen. Lower bounds for oblivious data structures. In Timothy M. Chan, editor, *30th SODA*, pages 2439–2447. ACM-SIAM, January 2019.

[KKNO16] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. Generic attacks on secure outsourced databases. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1329–1340. ACM Press, October 2016.

[KL21] Ilan Komargodski and Wei-Kai Lin. A logarithmic lower bound for oblivious RAM (for all parameters). In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 579–609, Virtual Event, August 2021. Springer, Heidelberg.

[KLO12] Eyal Kushilevitz, Steve Lu, and Rafail Ostrovsky. On the (in)security of hash-based oblivious RAM and a new balancing scheme. In Yuval Rabani, editor, *23rd SODA*, pages 143–156. ACM-SIAM, January 2012.

[KM17] Seny Kamara and Tarik Moataz. Boolean searchable symmetric encryption with worst-case sub-linear complexity. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 94–124. Springer, Heidelberg, April / May 2017.

[KM18] Seny Kamara and Tarik Moataz. SQL on structurally-encrypted databases. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part I*, volume 11272 of *LNCS*, pages 149–180. Springer, Heidelberg, December 2018.

[KM19] Seny Kamara and Tarik Moataz. Computationally volume-hiding structured encryption. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 183–213. Springer, Heidelberg, May 2019.

[KMO18] Seny Kamara, Tarik Moataz, and Olga Ohrimenko. Structured encryption and leakage suppression. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 339–370. Springer, Heidelberg, August 2018.

[KPR12] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012*, pages 965–976. ACM Press, October 2012.

[KPT21] Evgenios M. Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. Response-hiding encrypted ranges: Revisiting security via parametrized leakage-abuse attacks. In *2021 IEEE Symposium on Security and Privacy*, pages 1502–1519. IEEE Computer Society Press, May 2021.

[LD13] Lichun Li and Anwitaman Datta. Write-only oblivious RAM based privacy-preserved access of outsourced data. Cryptology ePrint Archive, Report 2013/694, 2013. https://eprint.iacr.org/2013/694.

[LMP18] Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. Improved reconstruction attacks on encrypted data using range query leakage. In *2018 IEEE Symposium on Security and Privacy*, pages 297–314. IEEE Computer Society Press, May 2018.

[LMWY20] Kasper Green Larsen, Tal Malkin, Omri Weinstein, and Kevin Yeo. Lower bounds for oblivious near-neighbor search. In Shuchi Chawla, editor, *31st SODA*, pages 1116–1134. ACM-SIAM, January 2020.

[LN18] Kasper Green Larsen and Jesper Buus Nielsen. Yes, there is an oblivious RAM lower bound! In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 523–542. Springer, Heidelberg, August 2018.

[LSY20] Kasper Green Larsen, Mark Simkin, and Kevin Yeo. Lower bounds for multi-server oblivious RAMs. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 486–503. Springer, Heidelberg, November 2020.

[MR22] Brice Minaud and Michael Reichle. Dynamic local searchable symmetric encryption. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 91–120. Springer, Heidelberg, August 2022.

[PD06] Mihai Patrascu and Erik D Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM Journal on Computing*, 2006.

[PPRY18] Sarvar Patel, Giuseppe Persiano, Mariana Raykova, and Kevin Yeo. PanORAMa: Oblivious RAM with logarithmic overhead. In Mikkel Thorup, editor, *59th FOCS*, pages 871–882. IEEE Computer Society Press, October 2018.

[PPSY21] Sarvar Patel, Giuseppe Persiano, Joon Young Seo, and Kevin Yeo. Efficient boolean search over encrypted data with reduced leakage. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 577–607. Springer, Heidelberg, December 2021.

[PPY19] Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. What storage access privacy is achievable with small overhead? In *ACM PODS*, 2019.

[PPY20] Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. Lower bounds for encrypted multi-maps and searchable encryption in the leakage cell probe model. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 433–463. Springer, Heidelberg, August 2020.

[PPYY19] Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 79–93. ACM Press, November 2019.

[PY19] Giuseppe Persiano and Kevin Yeo. Lower bounds for differentially private RAMs. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 404–434. Springer, Heidelberg, May 2019.

[PY22] Giuseppe Persiano and Kevin Yeo. Limits of preprocessing for single-server PIR. In *Annual ACM-SIAM Symposium on Discrete Algorithms*, 2022.

[PY23] Giuseppe Persiano and Kevin Yeo. Lower bound framework for differentially private and oblivious data structures. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part I*, volume 14004 of LNCS, pages 487–517. Springer, 2023.

[RACM17] Daniel S. Roche, Adam J. Aviv, Seung Geol Choi, and Travis Mayberry. Deterministic, stash-free write-only ORAM. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 507–521. ACM Press, October / November 2017.

[RFK+15] Ling Ren, Christopher W. Fletcher, Albert Kwon, Emil Stefanov, Elaine Shi, Marten van Dijk, and Srinivas Devadas. Constants count: Practical improvements to oblivious RAM. In Jaeyeon Jung and Thorsten Holz, editors, *USENIX Security 2015*, pages 415–430. USENIX Association, August 2015.

[SvS+13] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher W. Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: an extremely simple oblivious RAM protocol. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 299–310. ACM Press, November 2013.

[SWP00] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy*, pages 44–55. IEEE Computer Society Press, May 2000.

[WCM18] Sameer Wagh, Paul Cuff, and Prateek Mittal. Differentially private oblivious ram. *Proceedings on Privacy Enhancing Technologies*, 2018.

[WHC+14] Xiao Shaun Wang, Yan Huang, T.-H. Hubert Chan, abhi shelat, and Elaine Shi. SCORAM: Oblivious RAM for secure computation. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014*, pages 191–202. ACM Press, November 2014.

[WW18] Mor Weiss and Daniel Wichs. Is there an oblivious RAM lower bound for online reads? In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 603–635. Springer, Heidelberg, November 2018.

[Yao81] Andrew Chi-Chih Yao. Should tables be sorted? *Journal of the ACM (JACM)*, 1981.

[Yeo23] Kevin Yeo. Lower bounds for (batch) PIR with private preprocessing. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part I*, volume 14004 of LNCS, pages 518–550. Springer, 2023.

[ZKP16] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 707–720. USENIX Association, August 2016.

[ZWR+16] Samee Zahur, Xiao Shaun Wang, Mariana Raykova, Adria Gascón, Jack Doerner, David Evans, and Jonathan Katz. Revisiting square-root ORAM: Efficient random access in multi-party computation. In *2016 IEEE Symposium on Security and Privacy*, pages 218–234. IEEE Computer Society Press, May 2016.

A Breach-Resistant Structured Encryption

In this section we present the proof of the following result that is implicit in [AKM19]. See the discussion in Section 6.2.

Proof of Theorem 7. The server memory is initialized by filling each location with a block of 0's. The client's initialization consists in the selection of an encryption key K and of a seed s for a family $\mathcal{F} = \{F_s\}$ of pseudo-random functions. Both are kept in client memory along with the current version of the dummy index `idx` initialized to 0. Each new version of a block is stored in a fresh location and the j-th version of the i-block is stored in location $F_s(i \mid\mid j)$ of the server memory. In addition to the n real blocks, we consider one extra *dummy* block with index $n + 1$ whose current version number is kept in client memory as the dummy index `idx`.

To perform a read or a write of block i, the client first finds out the current version of the block. This is achieved by probing location $F_s(i \mid\mid j)$, from $j = 1$ and doubling the index until the first index $j^\star$ that does not contain a version of block i is found. Then the current block i has version index j for some j $j^\star/2 \leq j < j^\star$ and it can be found by performing binary search in the interval.

Once the current version j of block i is found, a read is completed by returning the block at location $F_s(i \mid\mid j)$, by incrementing `idx`, and by writing a dummy encrypted block at memory location $F_s(n+1 \mid\mid \texttt{idx})$. A write operation instead is completed by writing an encryption of the block received as input at memory location $F_s(i \mid\mid j+1)$. For the running time we observe that the current version j of a block can be found in time $O(\log j)$. Indeed $j^\star \leq 2 \cdot j$ and it is reached in $O(\log j)$ steps and the subsequent binary search is performed in an interval of length $j^\star/2 \leq j$ and thus it takes time $O(\log(j^\star/2)) = O(\log j)$.

In the above description, the server's memory grows unboundedly. However, the original work [AKM19] also presented a way to rebuild in time linear in the total number of operations while still remaining breach-resistant. To ensure maintain optimal server storage, this rebuilding algorithm may be executed every $O(n)$ operations to remove any useless information and maintain $O(n)$ storage.

For the security, we note that, for each operation, exactly one memory location at a pseudorandom location is overwritten. Therefore, the construction described above is $(s, 0)$-snapshot secure for every s. We stress that we have one construction for all values of s. $\square$