Lower Bounds for Multi-Server Oblivious RAMs

Kasper Green Larsen* Mark Simkin† Kevin Yeo‡

Abstract

In this work, we consider the construction of oblivious RAMs (ORAM) in a setting with *multiple servers* and the adversary may corrupt a subset of the servers. We present an $\Omega(\log n)$ overhead lower bound for any k-server ORAM that limits any PPT adversary to distinguishing advantage at most $1/4k$ when only one server is corrupted. In other words, if one insists on negligible distinguishing advantage, then multi-server ORAMs cannot be faster than single-server ORAMs even with polynomially many servers of which only one unknown server is corrupted. Our results apply to ORAMs that may err with probability at most $1/8$ as well as scenarios where the adversary corrupts larger subsets of servers. We also extend our lower bounds to other important data structures including oblivious stacks, queues, deques, priority queues and search trees.

1 Introduction

With the ever increasing amount of data, it is becoming infeasible for users to store their data on consumer machines such as phones or laptops. Therefore, there has been significant movement to outsourcing data to larger cloud storage providers. In this work, we focus on *privacy-preserving storage protocols* that considers the setting where a client outsource the storages of data to a server such as a cloud storage provider in a privacy-preserving manner. For privacy, the client wishes to maintain privacy for the outsourced data from the adversarial server as the outsourced data might be sensitive. In addition, the client wishes to maintain the ability to perform operations over the outsourced data in an efficient manner. As a first step, the client might consider encrypting the data locally and only sending ciphertexts of the data to the server while storing the private key exclusively in client memory. As a result, the server never sees the data in the plaintext. However, the adversarial server will still observe the patterns of access performed by the client to the encrypted data. Several works in the past decade (see [23, 26, 31, 21] as some examples) have shown that access pattern leakage can be used to compromise the privacy of the encrypted data. Therefore, it is an important problem to also efficiently hide the patterns of access to encrypted data to maintain privacy.

To solve the problem of hiding patterns of access to data, Goldreich and Ostrovsky [17] introduced the oblivious RAM (ORAM) primitive. Oblivious RAMs consider the problem of enabling a client to outsource an array with n entries, each consisting of exactly r bits, to the server, while enabling the client to both retrieve and perform update operations on any of the n array entries. The server itself consists of a memory of cells, each storing w bits. The client performs array update and retrieval operations by reading and writing to the server memory cells in a manner that hides the underlying operation being performed.

In terms of access pattern privacy, oblivious RAMs provide the guarantee that the adversarial server will learn no information about the sequence of array operations performed by the client except the total number of array operations performed. In more detail, any adversary that is given two plaintext sequences of array operations of equal length and observes the pattern of server cell accesses incurred by an ORAM, cannot determine which of the two plaintext sequences induced the observed ORAM access pattern.

*Computer Science Department. Aarhus University. Supported by a Villum Young Investigator Grant and an AUFF Starting Grant. larsen@cs.au.dk.

†Computer Science Department. Aarhus University. Supported by the European Unions's Horizon 2020 research and innovation program under grant agreement No 669255 (MPCPRO) and No 731583 (SODA). simkin@cs.au.dk.

‡Google LLC. kwlyeo@google.com.

The ORAM primitive is extremely powerful because it can be used in a blackbox manner to convert any non-oblivious algorithm/data structure into an oblivious version. In particular, every plaintext retrieval or update of memory performed by the non-oblivious algorithm/data structure will be replaced with an ORAM operation. Therefore, an important problem is constructing efficient ORAMs that may be used to also construct efficient algorithms/data structures for more complex tasks. For this reason, ORAM has been a well-studied topic over the past decade.

Efficiency of ORAMs is typically measured in terms of the bandwidth overhead. The bandwidth overhead is defined as the multiplicative factor extra server cells that must be accessed to process a single ORAM operation, i.e. if the ORAM accesses t server cells on an ORAM operation, then the bandwidth overhead is tr/w. Goldreich and Ostrovsky [17] presented the first ORAM with poly-logarithmic amortized bandwidth overhead per operation. A series of works [44, 18, 47, 19, 28, 48] continued improving the efficiency until recent works by Patel *et al.* [40] introduced an ORAM with $O(\log n \log\log n)$ bandwidth overhead and Asharov *et al.* [1] presented an ORAM with $O(\log n)$ bandwidth. Other variants of ORAM such as statistically-secure ORAMs [10, 9], parallel ORAMs [2, 7, 5] and garbled RAMs [15, 14, 38] have also been studied. Additionally, there has been work for efficiently constructing other oblivious data structures [49] including priority queues [25, 46]. To summarize, there are optimal $\Theta(\log n)$ constructions for oblivious arrays (RAMs), stacks, queues, deques and priority queues in the single-server setting.

There has also been a lot of work proving lower bounds on the efficiency of ORAMs. Goldreich and Ostrovsky [17] presented an $\Omega(\log n)$ lower bound with certain restrictions of statistical adversaries and a no-coding assumption using a "balls-and-bins" model. Larsen and Nielsen [35] improve these lower bounds by considering computational adversaries where ORAMs can encode memory in any possible manner. Further works investigate the question of whether relaxations of the original ORAM setting allow for more efficient constructions and prove that the essentially same lower bound holds for other oblivious data structures [24, 34], weaker differentially private guarantees [43] and weaker adversaries [22]. A natural open question that remains is whether one can find a meaningful relaxation that allows us to break the logarithmic barrier. In this work, we investigate whether having access to multiple non-colluding servers can help us achieve this goal.

The ability to construct asymptotically faster schemes using multiple non-colluding servers has been exhibited previously in another privacy-preserving data structure known as *private information retrieval* (PIR). PIR was introduced by Chor *et al.* [8] for the information-theoretic setting with multiple non-colluding servers and Kushilevitz and Ostrovsky [30] for the computationally-secure setting with a single server. PIR and ORAM mainly differ in the capability of the client and server to hold state. PIR requires that both the client and server are stateless (beyond the server being able to hold a static database). In particular, clients are not even able to hold a private key that may be used between multiple queries. In contrast, ORAM enables both the client and server to be stateful and use information between multiple queries. For more information comparing ORAM and PIR, we refer readers to Section 1.2 of [41]. In the single-server variant of information-theoretic PIR, there are several proofs showing at least linear $\Omega(n)$ bandwidth is required for each query [27, 16]. These are matched by the simplest construction of information-theoretic PIR where the client downloads the entire database on each query. On the other hand, PIR in the two non-colluding servers scenario may be constructed using sublinear bandwidth. The original works by Chor *et al.* [8] showed that there existed two-server statistically-secure PIR constructions with $O(\sqrt{n})$ and $O(n^{1/3})$ bandwidth. In more recent works, it has been shown that there exist PIR schemes in two non-colluding servers setting with sub-polynomial bandwidth [11].

As the multi-server setting considers weaker adversaries, the lower bounds for the single-server setting [35, 43] do not directly apply. Therefore, it seems plausible that oblivious data structures can be constructed in this model with faster efficiency than their single-server counterparts. While there have been several works [37, 12, 29, 20, 4, 6] that consider oblivious RAMs in the two (or more) non-colluding servers model, all of them have an overhead of at least $\Omega(\log n)$, meaning that they are not asymptotically faster than single-server oblivious RAMs and data structures. This leads to the natural question of whether it is possible to construct $o(\log n)$ overhead oblivious RAMs and data structures in the multiple non-colluding servers setting.

Before presenting our results, we find it insightful to discuss two simple strategies for implementing multi-

server ORAMs. In a setup with k servers, one naive approach is to pick a uniform random server and simply store the array there, without any obfuscation. This gives a great overhead of $O(1)$ (for $r = \Theta(w)$), but unfortunately security is very weak: A single adversarial server can distinguish two sequences of operations with probability $1/k$. Another simple strategy is to just ignore the first $k-1$ servers and run an optimal single-server ORAM on the last server. This gives an overhead of $O(\log n)$ but fails to exploit the multi-server setting. Is there anything in between these two extremes?

Our main result is a surprising negative resolution to this question, that is, if one insists on $o(\log n)$ overhead, then the *only* solution is to pick a random server and store the array there without any obfuscation, resulting in $1/k$ distinguishing probability for an adversarial server!

1.1 Our Results

Before we formally present our main contributions of this paper, we start by briefly describing the setting for which our lower bounds will apply. Our scenario consists of $k \geq 2$ servers that a client may use to host storage of parts of an oblivious RAM construction. We strictly consider weak, probabilistically polynomial time adversaries that can corrupt exactly one server and see all the probes performed by the ORAM on the corrupted server. As we are proving lower bounds, our results also apply to stronger adversaries that may be able to corrupt a large number of servers such as a constant fraction of all k servers. By the same argument, our lower bound also applies to computationally unbounded adversaries.

We note our ORAM lower bounds apply to the natural setting where operations arrive in an online manner. That is, the ORAM must complete one operation before receiving the next operation. Furthermore, the adversary is aware of when the processing of one operation ends and the processing of another operation begins.

Finally, we prove our lower bounds in the *cell probe model* of Yao [51]. In this model, the memory of the k servers is viewed as cells of w bits. The only measured cost is the number of probes performed to server cells. Computation, accessing memory stored on the client, generating randomness and querying the random oracle (if one exists) is completely free. Once again, since we are proving lower bounds, our results also apply to more natural cost models where these operations are charged appropriately.

We now present our main result:

Theorem 1 (Informal) *Any online k-server ORAM with n blocks of memory, consisting of $r \geq 1$ bits each, must have expected amortized overhead of $\Omega(\log(nr/m))$ on sequences of $\Theta(n)$ operations where the client has m bits of memory. This holds for probabilistically polynomial time adversaries that corrupt exactly one server and have a distinguishing advantage of at most $1/4k$ for any pair of length n sequences.*

For the natural setting where $r \leq m \leq n^{1-\epsilon}$ for any constant $\epsilon > 0$, the above lower bound simplifies to $\Omega(\log n)$. The above lower bound holds in the random oracle model, for any number of servers k and for any cell size w.

Using the above result, we show that multi-server ORAMs cannot be asymptotically faster than single-server ORAMs for any reasonable number of servers.

Corollary 1 (Informal) *For any $k = \mathbf{poly}(n)$, any k-server ORAM where probabilistically polynomial time adversaries that corrupts one of the k servers has negligible distinguishing advantage, must have $\Omega(\log n)$ overhead, which is asymptotically equivalent to the optimal single-server ORAM.*

Finally, we note that our lower bounds may be extended to other important data structures including stacks, queues, deques, priority queues and search trees.

Theorem 2 (Informal) *Any online k-server oblivious stacks, queues, deques, priority queues or search trees storing at most n elements, consisting of $r \geq 1$ bits each, must have expected amortized overhead of $\Omega(\log(nr/m))$ on sequences of n operations where the client has m bits of memory. This holds for probabilistically polynomial time adversaries that corrupt exactly one server and have a distinguishing advantage of at most $1/4k$ for any pair of length n sequences of operations.*

1.2 Our Techniques

In this section, we present an overview of the new techniques needed to prove our lower bound. To do this, we briefly overview the previous lower bound for ORAMs in the single-server setting by Larsen and Nielsen [35]. In addition, we show why the original proof fails for the setting when there exists multiple servers where the adversary may corrupt a single server. Larsen and Nielsen [35] used the information transfer technique introduced by Patrascu and Demaine [45]. For a sequence of n ORAM operations, each either a READ or a WRITE into one of n array entries, the information transfer tree is a complete binary tree with exactly n leaf nodes where the first operation is assigned to the leftmost node, the second operation is assigned to the second leftmost node and so forth. As we consider the cell probe model, each operation consists of a series of cell probes. Here a cell probe is simply an access to a server memory cell. For each probe p to a cell c, we identify both the operation that incurred p as well as the most recent, past operation that overwrote the contents of cell c. The cell probe p is *assigned* to the lowest common ancestor of the two leaf nodes associated with the operation performing p and the most recent, past operation that overwrote the probed cell. If the probed cell was never overwritten previously, the probe is not assigned to any node in the tree. Note that all cell probes are assigned to at most one node in the tree. Therefore, a lower bound on the number of total assigned probes results in a cell probe lower bound.

Consider any internal node v in the information transfer tree and the n_d left and right leaf nodes of the subtree rooted at v. Consider the sequence z of n_d WRITE operations associated with the left leaf nodes each consist of writing uniformly at random chosen r-bit strings into n_d unique indices. Furthermore, suppose the operations in the right subtree of z perform n_d READ operations that retrieve the n_d different random bit strings written in the left subtree. If most of the n_d READ operations in the right subtree return the right answer, then a large portion of the $r \cdot n_d$ bits of entropy generated in WRITE operations in the left subtree must be transferred to the answers of READ operations in the right subtree. Information may only be transferred between the left and right subtree through client storage and the probes assigned to the root of the subtree v. As client storage is typically small, this implies a large number of probes must be assigned to v with high probability. Otherwise, one can construct an impossible compression scheme of the $r \cdot n_d$ random bits generated in the left subtree. Suppose there exists another sequence y of n ORAM operations that assigns significantly less probes to v. We can construct a simple and efficient adversary that distinguishes the two sequences by simply count the number of probes assigned to v in polynomial time, which contradicts the obliviousness assumption. The application of this argument to many nodes of the information transfer tree suffices to prove the single-server ORAM lower bound.

Moving to the multi-server setting with $k \geq 2$ servers, the adversary is able to only see the probes performed to one server. For any multi-server ORAM construction that performs most of its probes to a single server, we can extend the adversary from the single-server setting by corrupting a server uniformly at random. As a result, the adversary is able to distinguish with probability $\Omega(1/k)$ unless y also assigns similarly large number of probes with high probability to the corrupted server. However, there might exist intelligent schemes that evenly distribute all probes across all k servers such that the adversary will only be able to see a small number of probes. Even worse, the probes might be distributed such that the probability any server sees even one probe is small when the number of servers is large such as $k = \Omega(n^2)$. Therefore, the challenge is proving that sequence y must also assign a large number of probes to v even when the adversary corrupts only a single server.

We now give intuition as to why these simpler counting adversaries do not suffice to prove lower bounds for k-server ORAMs with the current techniques. Going back to the proof framework of [35], we can consider an impossible sequence y that assigns a small number of probes to v in expectation. We need to show that the adversary can distinguish sequences y and z where z is the worst-case sequence for node v that maximizes the number of probes assigned to node v described in the previous paragraph. In the proof of [35], it was critically shown that the number of probes assigned by z to v is large with very high probability. This is a strong statement about the distribution of probes (as opposed to just bounding the expected number of probes assigned to v) that showed the counting adversary to distinguish with an impossible advantage. We note that this argument heavily utilizes the fact that the contents of cells of probes assigned to node v encode almost all the information generated in the left subtree of v. Unfortunately, these techniques do not

work for multi-server ORAM schemes that can arbitrarily distribute probes to different servers. There is no requirement or guarantee that by restricting to probes that are both assigned to node v and all occur on a single server, the contents of these probed cells still encode the r-bit random strings generated in the left subtree of v. For example, a multi-server might distribute the r-bit random strings across several servers using an information-theoretic secret sharing scheme. Therefore, the necessary guarantees needed for the simple counting adversary to successfully distinguish sequences y and z might not be true in the multiple server setting.

The main idea of our proof technique is to consider a more sophisticated adversary that groups the number of observable probe counts to the corrupted server that are assigned to node v into geometrically increasing sets, $[2^0, 2^1)$, $[2^1, 2^2)$, $\ldots$, $[2^j, \infty)$ for sufficiently large j. The new adversary will attempt to distinguish sequences y and z by finding a grouping of probe counts that are more likely for sequence y instead of sequence z. If sequence z results in a probe count in group $[2^i, 2^{i+1})$ with probability p, then sequence y must also result in a probe count in that group with probability at least $p - \varepsilon$ if the adversary should not be able to distinguish the sequences y and z with probability greater than ε. As a result, we can show that the expectation of the probe counts assigned to v to each server by y must be similarly large as under z, completing the proof.

1.3 Related Works

In this section, we survey related works that have proved lower bounds in the cell probe model for data structures with and without privacy guarantees. Yao [51] introduced the cell probe model as a model for proving lower bounds. Fredman and Saks [13] presented the chronogram technique to prove almost logarithmic lower bounds. Patrascu and Demanie [45] introduced the information transfer technique to prove logarithmic lower bounds. Panigrahy *et al.* [39] present the cell-sampling technique to prove almost logarithmic lower bounds for static data structures. Larsen [32, 33] presented the first super-logarithmic lower bounds for data structures with $\Theta(\log n)$-bit outputs. The first super-logarithmic lower bounds for decision data structures was proved in [36]. The above list only several examples of the many works in cell probe lower bounds.

Additionally, there has been work showing that proving lower bounds for certain oblivious data structures will be very difficult. Boyle and Naor [3] show that proving lower bounds for offline ORAMs that receive all operations ahead of time is as hard as proving sorting circuit lower bounds. Furthermore, Weiss and Wichs [50] prove that lower bounds for read-only ORAMs would imply unknown lower bounds in either sorting circuits and/or locally decodable codes.

2 Formal Model

We prove our lower bounds in a variant of the oblivious cell probe model of Larsen and Nielsen [35], adapted to a setting with k servers. In this model, an ORAM consists of k servers $S_1, \ldots, S_k$, each with a server memory of w-bit cells, where each cell has an integer address in $[K]$ for some $K \leq 2^w$. We also assume $k \leq 2^w$ such that a cell has enough bits to store the index of a server. An ORAM is furthermore equipped with a client memory of m bits, which is free to access. A multi-server ORAM processes READ and WRITE operations by reading and writing to memory cells at the servers. For READ operations, the ORAM terminates by announcing the answer to the READ based on what it has probed.

We refer to the reading and writing of a memory cell simply as probing it - also to distinguish reading and writing cells from READ and WRITE operations. The running time is defined as the number of cells it probes when processing READ and WRITE operations. Randomized ORAMs furthermore have access to an arbitrarily long uniform random bit string R, which is referred to as the random oracle bit string. The bit string R is drawn before any operations are performed on the ORAM and is chosen independently of the future operations. We say that a randomized ORAM has failure probability δ if for every sequence of operations $\mathrm{op}_1, \ldots, \mathrm{op}_M$, and for every query op_i in that sequence, the probability that op_i is answered correctly is at least $1 - \delta$.

When processing READ and WRITE operations, the cells probed and the contents written to cells in each step may be an arbitrary deterministic function of the client memory, random oracle bit string and contents of all other cells probed so far while processing the current operation. The ORAM is also allowed to update the client memory in each step, again setting the contents to an arbitrary deterministic function of the current memory, random oracle bit string and contents of cells probed so far. Allowing an arbitrary deterministic function abstracts away the instruction set of a normal RAM and allows arbitrary computations free of charge.

To define the security requirements of a multi-server ORAM, let

$$y := (\mathrm{op}_1, \ldots, \mathrm{op}_M)$$

denote a sequence of M READ and WRITE operations. Let

$$A(y) := (A(\mathrm{op}_1), \ldots, A(\mathrm{op}_M))$$

denote the corresponding *probe sequence*, where each $A(\mathrm{op}_i)$ is the list of probes made while processing op_i. Note that $A(y)$ is a deterministic function of the random oracle bit string and the sequence y. Each probe in a list $A(\mathrm{op}_i)$ is described by a tuple (s, a), where s is the index of the server where the probe is made and a is the address of the memory cell accessed at the server. For a server S_i, we let $A_{|S_i}(\mathrm{op}_j)$ denote the sub list of $A(\mathrm{op}_j)$ containing only the probes (s, a) with $s = i$. We similarly define $A_{|S_i}(y) = (A_{|S_i}(\mathrm{op}_1), \ldots, A_{|S_i}(\mathrm{op}_M))$ as the probes seen by server S_i. A multi-server ORAM is secure if it satisfies the following security guarantee:

Definition 1 (Security) *A multi-server ORAM is (ε, δ)-secure if the following two properties hold:*

Indistinguishability: *For any two sequences of operations y and z of the same length n and for any server S_i, their probe sequences $A_{|S_i}(y)$ and $A_{|S_i}(z)$ cannot be distinguished with probability better than ε by an algorithm which is polynomial time in n. Formally, if $\mathcal{A}_{|S_i,n}$ denotes the image of $A_{|S_i}$ on sequences of length n and $f : \mathcal{A}_{|S_i,n} \to \{0, 1\}$ denotes a polynomial time computable function, then it must be the case that $|\Pr[f(A_{|S_i}(y)) = 1] - \Pr[f(A_{|S_i}(z)) = 1]| \leq \varepsilon$ for any two sequences y and z of length n. Here the probability is taken over the randomness R of the ORAM.*

Correctness: *The ORAM has failure probability at most δ.*

3 Lower Bound

We use the information transfer technique by Patrascu and Demaine [45], modified to multiple servers. We consider various sequences of n READ and WRITE operations to an ORAM $\mathcal{O}$ with memory size n. The READ and WRITE operations store and retrieve r-bit strings and the servers have cell size w bits. We prove the following theorem

Theorem 3 *Any ORAM with k servers that is $(1/4k, 1/8)$-secure, has server cell size w bits, has client memory size m bits and that supports storing r-bit values in n entries, must make an expected amortized $\Omega(r \log(nr/m)/w)$ probes per operation over sequences of n operations.*

First we define the information transfer tree $\mathcal{T}$. For any sequence of n operations $x = \mathrm{op}_1, \ldots, \mathrm{op}_n$, we construct a binary tree $\mathcal{T}$ with the operations as leaves. When processing the operations op_i, we assign the probes in $A(\mathrm{op}_i)$ to the nodes of $\mathcal{T}$. For each probe $p = (s, a) \in A(\mathrm{op}_i)$, consider the last time the cell (s, a) was probed during $\mathrm{op}_1, \ldots, \mathrm{op}_n$. If op_j with $j \leq i$ denotes the last operation in which the cell was probed, we assign p to the lowest common ancestor of op_i and op_j in $\mathcal{T}$. If p is the first probe to access (s, a) we do not assign it to any node of $\mathcal{T}$. For each node v of $\mathcal{T}$, we let $P(x, v)$ denote the set of probes assigned to v while processing x (note the $P(x, v)$ is a random variable due to the randomness R of the ORAM).

Observe that any probe is assigned to at most one node of $\mathcal{T}$.

We now consider a fixed "dummy" sequence of operations:

$$y := \textsc{read}(0), \textsc{read}(0), \textsc{read}(0), \cdots, \textsc{read}(0)$$

which always just reads the first ORAM memory cell. We say that the root of $\mathcal{T}$ has depth 0 and the leaves have depth $\log n$. For simplicity, we also assume n is a power of two. For a node $v \in \mathcal{T}$, we use $d(v)$ to denote its depth. We will prove the following:

Lemma 1 *If $\mathcal{O}$ is $(1/4k, 1/8)$-secure and has client memory size m, then for any node $v \in \mathcal{T}$ of depth $d = d(v) \leq \log(nr/m) - 3$, it holds that $\mathbb{E}_R[|P(y,v)|] = \Omega(nr/(w2^d))$.*

Lemma 1 immediately gives our result, since by linearity of expectation we get that the total number of probes T made by $\mathcal{O}$ satisfies:

$$\begin{aligned}
\mathbb{E}[T] &\geq \sum_{v \in \mathcal{T}} \mathbb{E}[|P(y,v)|] \\
&\geq \sum_{d=0}^{\log(nr/m)-3} \sum_{v \in \mathcal{T}: d(v)=d} \mathbb{E}[|P(y,v)|] \\
&\geq \sum_{d=0}^{\log(nr/m)-3} 2^d \cdot \Omega(nr/(w2^d)) \\
&= \Omega(nr \log(nr/m)/w).
\end{aligned}$$

Thus what remains is to prove Lemma 1. To do so, consider a node $v \in \mathcal{T}$ of depth $d(v) \leq \log(nr/m) - 3$. We consider a distribution $\mathcal{D}_v$ over sequences of n operations $\mathrm{op}_1, \ldots, \mathrm{op}_n$. The distribution is as follows: For every op_i that is outside the subtree rooted at v, we let $\mathrm{op}_i = \textsc{read}(0)$. We let the $n_d = n/2^{d+1}$ operations in v's left subtree be $\textsc{write}(1, r_1), \ldots, \textsc{write}(n_d, r_{n_d})$ where each r_i is a uniform random r-bit string. We let the n_d operations in v's right subtree be $\textsc{read}(1), \ldots, \textsc{read}(n_d)$. As in previous ORAM lower bounds, we first argue that under distribution $\mathcal{D}_v$, there must be many probes assigned to v in expectation:

Lemma 2 *Let $z \sim \mathcal{D}_v$ be a sequence of n operations. If $\mathcal{O}$ is $(1/4k, 1/8)$-secure and has client memory size m, then $\Pr_{z,R}[|P(z,v)| \geq (1/12)nr/(w2^d)] \geq 3/4$.*

We defer the proof of Lemma 2 to Section 3.1 as it follows previous proofs uneventfully.

We will now use the security guarantees of $\mathcal{O}$ and Lemma 2 to prove Lemma 1. To do so, start by partitioning the set $P(x,v)$ into k sets $P_{|S_1}(x,v), \ldots, P_{|S_i}(x,v)$ where $P_{|S_i}(x,v)$ contains all probes to a cell at server S_i while processing a sequence of operations x. Let $z \sim \mathcal{D}_v$. For each $j \in \{0, \ldots, \log(nr/(48w2^d))\}$ define $q_{i,j}$ as

$$q_{i,j} := \Pr_{z,R}[|P_{|S_i}(z,v)|] \in [2^j, 2^{j+1}).$$

when $j < \log(nr/(48w2^d))$, and define

$$q_{i,\log(nr/(48w2^d))} := \Pr_{z,R}[|P_{|S_i}(z,v)|] \geq nr/(48w2^d).$$

Similarly, define

$$\hat{q}_{i,j} := \Pr_{R}[|P_{|S_i}(y,v)|] \in [2^j, 2^{j+1}).$$

and

$$\hat{q}_{i,\log(nr/(48w2^d))} := \Pr_{R}[|P_{|S_i}(y,v)|] \geq nr/(48w2^d).$$

We first observe that for all i, j, we must have $\hat{q}_{i,j} \geq q_{i,j} - 1/4k$. To see this, observe that if $\hat{q}_{i,j} < q_{i,j} - 1/4k$ then for an $x \in \{y, z\}$, the server S_i can distinguish whether $x = y$ or $x = z$ with probability greater than $1/4k$ as follows: When seeing $A_{|S_i}(x)$, output 1 if $|P_{|S_i}(x,v)| \in [2^j, 2^{j+1})$ and 0 otherwise. Notice that this information can be computed from $A_{|S_i(x)}$. As a technical caveat, not that z is random and not a fixed sequence as in the definition of the security guarantee. But if the adversary can distinguish the random z from y, then by averaging, there must exist a fixed sequence in the support of z which can also be distinguished from y with the same advantage. Hence $\hat{q}_{i,j} \geq q_{i,j} - 1/4k$ for all i, j.

We now split the proof in two cases. Assume first that $\sum_i q_{i,\log(nr/(48w2^d))} \geq 1/2$. In this case, we have $\sum_i \hat{q}_{i,\log(nr/(48w2^d))} \geq 1/2 - k/4k = 1/4$. By linearity of expectation, this implies $\mathbb{E}_R[|P(y,v)|] \geq (1/4)(nr/48w2^d) = \Omega(nr/(w2^d))$ as claimed. Next, assume that $\sum_i q_{i,\log(nr/(48w2^d))} < 1/2$. By Lemma 2, we have

$$\Pr_{z,R}[|P(z,v)| \geq (1/12)nr/(w2^d)] \geq 3/4.$$

Now let E denote the event that for all i, we have:

$$|P_{|S_i}(z,v)| < nr/(48w2^d).$$

We then have:

$$\Pr_{z,R}[|P(z,v)| \geq (1/12)nr/(w2^d) \wedge E] \geq 1 - 1/4 - (1 - \Pr[E]) = \Pr[E] - 1/4$$

Therefore

$$\begin{aligned}
\Pr_{z,R}[|P(z,v)| \geq (1/12)nr/(w2^d) \mid E] &= \Pr_{z,R}[|P(z,v)| \geq (1/12)nr/(w2^d) \wedge E]/\Pr[E] \\
&\geq (\Pr[E] - 1/4)/\Pr[E] \\
&= 1 - 1/(4\Pr[E]) \\
&\geq 1/2.
\end{aligned}$$

This implies that

$$\mathbb{E}_{z,R}[|P(z,v)| \mid E] \geq (1/24)nr/(w2^d).$$

We will show that this means that

$$\mathbb{E}_R[|P(y,v)|] = \Omega(nr/(w2^d)).$$

To see this, consider what happens if we modify the definition of $P(z,v)$ such that we set $P(z,v) = \emptyset$ if there is at least one server S_i such that $|P_{|S_i}(z,v)| \geq nr/(48w2^d)$. Let $P^*(z,v)$ denote this modified version of $P(z,v)$ and let $q^*_{i,j}$ denote the corresponding versions of the $q_{i,j}$'s. We clearly have $q^*_{i,j} \leq q_{i,j}$ for all i,j. Moreover, conditioned on E, we have $P(z,v) = P^*(z,v)$. It follows that

$$\mathbb{E}_{z,R}[|P^*(z,v)|] = \Pr_{z,R}[E]\mathbb{E}_{z,R}[|P(z,v)| \mid E] \geq (1/48)nr/(w2^d).$$

At the same time, we also have

$$\mathbb{E}_{z,R}[|P^*(z,v)|] \leq \sum_{i=1}^{k} \sum_{j=0}^{\log(nr/(48w2^d))-1} q^*_{i,j} 2^{j+1}$$

Using that $q^*_{i,j} \leq q_{i,j}$, this means that

$$\sum_{i=1}^{k} \sum_{j=0}^{\log(nr/(48w2^d))-1} q_{i,j} 2^{j+1} \geq (1/48)nr/(w2^d).$$

Now $\hat{q}_{i,j} \geq q_{i,j} - 1/4k$ thus

$$\begin{aligned}
\sum_{i=1}^{k} \sum_{j=0}^{\log(nr/(48w2^d))-1} \hat{q}_{i,j} 2^{j+1} &\geq (1/48)nr/(w2^d) - \sum_{i=1}^{k} \sum_{j=0}^{\log(nr/(48w2^d))-1} 2^{j+1}/4k \\
&= (1/48)nr/(w2^d) - \sum_{i=1}^{k} 2^{\log(nr/(48w2^d))+1}/4k \\
&= (1/48)nr/(w2^d) - (1/96)nr/(w2^d) \\
&= (1/96)nr/(w2^d).
\end{aligned}$$

But

$$\begin{aligned}\mathbb{E}_R[|P(y,v)|] &\geq \sum_{i=1}^{k}\sum_{j=1}^{\log(nr/(48w2^d))} \hat{q}_{i,j}2^j \\ &\geq (1/2)\sum_{i=1}^{k}\sum_{j=1}^{\log(nr/(48w2^d))-1} \hat{q}_{i,j}2^{j+1} \\ &= \Omega(nr/(w2^d)).\end{aligned}$$

This completes the proof of Lemma 1.

3.1 Proof of Lemma 2

We prove this via an encoding argument. An encoder Alice and a decoder Bob share access to the random oracle bit string R used by $\mathcal{O}$. Alice receives as input the $n_d = n/2^{d+1}$ random bit strings $r_1, \ldots, r_{n_d}$ given as arguments to the WRITE operations in v's left subtree and wants to transmit them to Bob. By Shannon's source coding theorem, if Alice sends a prefix free code, then the expected length of the message must be at least $n_d r = nr/2^{d+1}$ bits. They proceed as follows:

Encoding. Alice constructs the sequence of operations $z = \mathrm{op}_1, \ldots, \mathrm{op}_n$ where the WRITE operations in v's left subtree write the values $r_1, \ldots, r_{n_d}$ to entries $1, \ldots, n_d$, and the READ operations in v's right subtree read the entries $1, \ldots, n_d$. All op_i outside v's subtree are simply READ(0) operations. Thus z is distributed according to $\mathcal{D}_v$. Alice runs the sequence of operations on $\mathcal{O}$ and constructs the set $P(z, v)$ and also counts how many of the READ operations in v's right subtree that fail. Let f denote the number of READ operations that err. Her message to Bob is as follows:

1. If $f \geq 3n_d/4$ or $|P(z,v)| \geq (1/12)nr/(w2^d)$, then Alice sends a 0-bit, followed by $n_d r = nr/2^{d+1}$ bits giving a naive encoding of $r_1, \ldots, r_{n_d}$. This costs $1 + nr/2^{d+1}$ bits.
2. Otherwise, Alice starts by sending a 1-bit. For each probe $p = (s, a)$, Alice sends s, a and the contents of the cell with address a at server S_s as it was immediately after processing the operations in v's subtree. She also sends the contents of the client memory as it was immediately after processing v's left subtree. This costs $1 + |P(z,v)|(\log k + 2w) \leq 1 + 3w(1/12)nr/2^d + m = 1 + nr/2^{d+2} + m$. We required $d \leq \log(nr/m) - 3$, hence $m \leq nr/2^{d+3}$ and it follows that the cost is no more than $1 + (3/4)nr/2^{d+1}$.

Decoding. Bob starts by checking the first bit of Alice's message. If this is a 0-bit, Bob immediately recovers $r_1, \ldots, r_{n_d}$ from the remaining part of Alice's message. Otherwise, Bob reconstructs the set $P(z, v)$ and the contents of those cells as they were right after processing v's left subtree. Bob now runs $\mathcal{O}$ using the randomness R on the sequence z until just before v's left subtree (this is solely READ(0) operations, so Bob knows these). He then skips over all operations in the left subtree and continues running the READ$(1), \ldots,$ READ(n_d) in v's right subtree. While processing these operations, Bob checks each cell that is probed. If the cell is in $P(z, v)$, Bob knows the contents from Alice's message. If it is not in $P(z, v)$, then Bob already knows the contents as they were not updates during v's left subtree by definition of $P(z, v)$. Thus Bob can process all the READ operations and recovers $r_1, \ldots, r_{n_d}$.

Analysis. Let $\alpha = \Pr[|P(z,v)| < (1/12)nr/(w2^d)]$. Then the probability that Alice sends a non-trivial encoding (step 2.) is at least $1 - (1 - \alpha) - 1/6 = \alpha - 1/6$. This follows by a union bound and Markov's inequality since $\mathbb{E}[f] = n_d/8$ due to $\mathcal{O}$ being $(1/4k, 1/8)$-secure. The expected length of the encoding is hence at most

$$1 + (1 - \alpha + 1/6)nr/2^{d+1} + (\alpha - 1/6)(3/4)nr/2^{d+1}.$$

This is less than $n_d r$ for any constant $\alpha > 1/6$. We thus conclude that $\Pr[|P(z,v) \geq (1/12)nr/(w2^d)] \geq 3/4$.

4 Extension to Oblivious Data Structures

In this section, we show that the above lower bound may be extended to other oblivious data structures including stacks, queues, deques, priority queues and search trees using techniques by Jacob *et al.* [24]. We describe how to modify the lower bound to handle stacks and queues. Since one can use deques, priority queues and search trees to simulate a stack and/or queue, we only need to prove a lower bound for oblivious stacks and queues.

For the "dummy" sequence of operations in the lower bound, we will use the following sequence for both stacks and queues:

$$\text{PUSH}(\bar{0}), \text{POP}(), \text{PUSH}(\bar{0}), \text{POP}(), \ldots, \text{PUSH}(\bar{0}), \text{POP}()$$

where $\bar{0}$ is the all-zeroes bit string of length r. The lower bound also requires designing a worst case sequence for each node v in the information transfer tree. If we let n_d be the number of operations in the left and right subtree of v, then we make the operations of the leaf nodes of the subtree rooted at v be

$$\text{PUSH}(r_1), \ldots, \text{PUSH}(r_{n_d}), \text{POP}(), \ldots, \text{POP}()$$

where each r_i is also a uniformly random r-bit string. Outside v's subtree, we make alternating $\text{PUSH}(\bar{0}), \text{POP}()$ operations. This sequence has the desired property (for both stacks and queues) that the queries to the right subtree of v have to retrieve the random strings generated from the left subtree of v. The rest of the lower bound proof proceeds identically using these new hard operational sequences for stacks and queues.

Theorem 4 *Any oblivious stack, queue, deque, priority queue or search tree with k servers that is $(1/4k, 1/8)$-secure, has server cell size w bits, has client memory size m bits and that supports storing up to n r-bit elements, must make an expected amortized $\Omega(r \log(nr/m)/w)$ probes per operation over sequences of n operations.*

5 Conclusions

In this work, we study oblivious data structures that enable performing operations without revealing information about these operations. There has been a long line of work for oblivious data structures that has led to tight $\Theta(\log n)$ constructions for many oblivious data structures. However, this means there is a significant gap between plaintext and oblivious operations for many data structures such as arrays (RAMs). A natural next question is: are there any settings where we can achieve meaningful privacy with smaller $o(\log n)$ overhead? This question was investigated in [43, 42] that considered weaker differentially private access hiding only operational sequences that differ in very few operations. Additionally, [22] considered weaker adversaries that may not view the beginning and ending of operations. In both cases, the weakening of the adversaries was not sufficient to achieve $o(\log n)$ overhead. We continue along this line of research by showing weaker adversaries that only corrupt one server in the multi-server model does not suffice to achieve $o(\log n)$ overhead.

References

[1] G. Asharov, I. Komargodski, W.-K. Lin, K. Nayak, E. Peserico, and E. Shi. OptORAMa: Optimal oblivious RAM. Cryptology ePrint Archive, Report 2018/892.

[2] E. Boyle, K.-M. Chung, and R. Pass. Oblivious parallel RAM and applications. In *Theory of Cryptography Conference*, pages 175–204. Springer, 2016.

[3] E. Boyle and M. Naor. Is there an oblivious RAM lower bound? In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pages 357–368. ACM, 2016.

[4] P. Bunn, J. Katz, E. Kushilevitz, and R. Ostrovsky. Efficient 3-party distributed oram. *IACR Cryptology ePrint Archive*, 2018:706, 2018.

[5] T.-H. H. Chan, Y. Guo, W.-K. Lin, and E. Shi. Oblivious hashing revisited, and applications to asymptotically efficient ORAM and OPRAM. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 660–690. Springer, 2017.

[6] T.-H. H. Chan, J. Katz, K. Nayak, A. Polychroniadou, and E. Shi. More is less: Perfectly secure oblivious algorithms in the multi-server setting. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 158–188. Springer, 2018.

[7] B. Chen, H. Lin, and S. Tessaro. Oblivious parallel RAM: improved efficiency and generic constructions. In *Theory of Cryptography Conference*, pages 205–234. Springer, 2016.

[8] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 41–50. IEEE, 1995.

[9] K.-M. Chung, Z. Liu, and R. Pass. Statistically-secure ORAM with $\tilde{O}(\log^2 n)$ overhead. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 62–81. Springer, 2014.

[10] I. Damgård, S. Meldgaard, and J. B. Nielsen. Perfectly secure oblivious RAM without random oracles. In *Theory of Cryptography Conference*, pages 144–163. Springer, 2011.

[11] Z. Dvir and S. Gopi. 2-server pir with subpolynomial communication. *Journal of the ACM (JACM)*, 63(4):39, 2016.

[12] S. Faber, S. Jarecki, S. Kentros, and B. Wei. Three-party oram for secure computation. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 360–385. Springer, 2015.

[13] M. Fredman and M. Saks. The cell probe complexity of dynamic data structures. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 345–354. ACM, 1989.

[14] S. Garg, S. Lu, R. Ostrovsky, and A. Scafuro. Garbled RAM from one-way functions. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 449–458. ACM, 2015.

[15] C. Gentry, S. Halevi, S. Lu, R. Ostrovsky, M. Raykova, and D. Wichs. Garbled RAM revisited. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 405–422. Springer, 2014.

[16] O. Goldreich, H. Karloff, L. J. Schulman, and L. Trevisan. Lower bounds for linear locally decodable codes and private information retrieval. In *Proceedings 17th IEEE Annual Conference on Computational Complexity*, pages 175–183. IEEE, 2002.

[17] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM (JACM)*, 1996.

[18] M. T. Goodrich and M. Mitzenmacher. Privacy-preserving access of outsourced data via oblivious RAM simulation. In *International Colloquium on Automata, Languages, and Programming*, pages 576–587. Springer, 2011.

[19] M. T. Goodrich, M. Mitzenmacher, O. Ohrimenko, and R. Tamassia. Privacy-preserving group data access via stateless oblivious RAM simulation. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 157–167. Society for Industrial and Applied Mathematics, 2012.

[20] S. D. Gordon, J. Katz, and X. Wang. Simple and efficient two-server oram. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 141–157. Springer, 2018.

[21] P. Grubbs, M.-S. Lacharité, B. Minaud, and K. G. Paterson. Learning to reconstruct: Statistical learning theory and encrypted database attacks. Cryptology ePrint Archive, Report 2019/011.

[22] P. Hub'avcek, M. Koucký, K. Král, and V. Slívová. Stronger lower bounds for online ORAM. *CoRR*, abs/1903.03385, 2019.

[23] M. S. Islam, M. Kuzu, and M. Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *NDSS*, 2012.

[24] R. Jacob, K. G. Larsen, and J. B. Nielsen. Lower bounds for oblivious data structures. In *SODA '19*, 2019.

[25] Z. Jafargholi, K. G. Larsen, and M. Simkin. Optimal oblivious priority queues and offline oblivious ram. Cryptology ePrint Archive, Report 2019/237, 2019. `https://eprint.iacr.org/2019/237`.

[26] G. Kellaris, G. Kollios, K. Nissim, and A. O'Neill. Generic attacks on secure outsourced databases. In *CCS '16*, 2016.

[27] I. Kerenidis and R. De Wolf. Exponential lower bound for 2-query locally decodable codes via a quantum argument. *Journal of Computer and System Sciences*, 69(3):395–420, 2004.

[28] E. Kushilevitz, S. Lu, and R. Ostrovsky. On the (in) security of hash-based oblivious RAM and a new balancing scheme. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 143–156. Society for Industrial and Applied Mathematics, 2012.

[29] E. Kushilevitz and T. Mour. Sub-logarithmic distributed oblivious ram with small block size. *arXiv preprint arXiv:1802.05145*, 2018.

[30] E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pages 364–373. IEEE, 1997.

[31] M.-S. Lacharité, B. Minaud, and K. G. Paterson. Improved reconstruction attacks on encrypted data using range query leakage. In *IEEE S&P '18*, 2018.

[32] K. G. Larsen. The cell probe complexity of dynamic range counting. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 85–94. ACM, 2012.

[33] K. G. Larsen. Higher cell probe lower bounds for evaluating polynomials. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 293–301. IEEE, 2012.

[34] K. G. Larsen, T. Malkin, O. Weinstein, and K. Yeo. Lower bounds for oblivious near-neighbor search. *arXiv preprint arXiv:1904.04828*, 2019.

[35] K. G. Larsen and J. B. Nielsen. Yes, there is an Oblivious RAM lower bound! In *Annual International Cryptology Conference*, pages 523–542. Springer, 2018.

[36] K. G. Larsen, O. Weinstein, and H. Yu. Crossing the logarithmic barrier for dynamic boolean data structure lower bounds. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 978–989. ACM, 2018.

[37] S. Lu and R. Ostrovsky. Distributed oblivious ram for secure two-party computation. In *Theory of Cryptography Conference*, pages 377–396. Springer, 2013.

[38] S. Lu and R. Ostrovsky. Black-box parallel garbled RAM. In *Annual International Cryptology Conference*, pages 66–92. Springer, 2017.

[39] R. Panigrahy, K. Talwar, and U. Wieder. Lower bounds on near neighbor search via metric expansion. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 805–814. IEEE, 2010.

[40] S. Patel, G. Persiano, M. Raykova, and K. Yeo. PanORAMa: Oblivious RAM with logarithmic overhead. In *FOCS '18*, 2018.

[41] S. Patel, G. Persiano, and K. Yeo. Private stateful information retrieval. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1002–1019. ACM, 2018.

[42] S. Patel, G. Persiano, and K. Yeo. What storage access privacy is achievable with small overhead? In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 182–199. ACM, 2019.

[43] G. Persiano and K. Yeo. Lower bounds for differentially private RAMs. In *EUROCRYPT '19*, 2019.

[44] B. Pinkas and T. Reinman. Oblivious RAM revisited. In *Annual Cryptology Conference*, pages 502–519. Springer, 2010.

[45] M. Pătraşcu and E. D. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM Journal on Computing*, 35(4):932–963, 2006.

[46] E. Shi. Path oblivious heap. Cryptology ePrint Archive, Report 2019/274, 2019. `https://eprint.iacr.org/2019/274`.

[47] E. Stefanov, E. Shi, and D. Song. Towards practical oblivious RAM. *arXiv preprint arXiv:1106.3652*, 2011.

[48] E. Stefanov, M. Van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas. Path ORAM: an extremely simple oblivious RAM protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 299–310. ACM, 2013.

[49] X. S. Wang, K. Nayak, C. Liu, T. Chan, E. Shi, E. Stefanov, and Y. Huang. Oblivious data structures. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 215–226. ACM, 2014.

[50] M. Weiss and D. Wichs. Is there an oblivious ram lower bound for online reads? In *Theory of Cryptography Conference*, pages 603–635. Springer, 2018.

[51] A. C.-C. Yao. Should tables be sorted? *Journal of the ACM (JACM)*, 28(3):615–628, 1981.