A Logarithmic Lower Bound for Oblivious RAM (for all parameters)[⋆]

Ilan Komargodski[1] and Wei-Kai Lin[2]

[1] Hebrew University and NTT Research
ilank@cs.huji.ac.il
[2] Cornell University
wklin@cs.cornell.edu

Abstract. An Oblivious RAM (ORAM), introduced by Goldreich and Ostrovsky (J. ACM 1996), is a (probabilistic) RAM that hides its access pattern, i.e., for every input the observed locations accessed are similarly distributed. In recent years there has been great progress both in terms of upper bounds as well as in terms of lower bounds, essentially pinning down the smallest overhead possible in various settings of parameters.

We observe that there is a very natural setting of parameters in which *no* non-trivial lower bound is known, even not ones in restricted models of computation (like the so called balls and bins model). Let N and $\boldsymbol{w}$ be the number of cells and bit-size of cells, respectively, in the RAM that we wish to simulate obliviously. Denote by $\boldsymbol{b}$ the cell bit-size of the ORAM. *All* previous ORAM lower bounds have a multiplicative $\boldsymbol{w}/\boldsymbol{b}$ factor which makes them trivial in many settings of parameters of interest.

In this work, we prove a new ORAM lower bound that captures this setting (and in all other settings it is at least as good as previous ones, quantitatively). We show that any ORAM must make (amortized)

$$\Omega\left(\log\left(\frac{N\boldsymbol{w}}{m}\right)/\log\left(\frac{\boldsymbol{b}}{\boldsymbol{w}}\right)\right)$$

memory probes for every logical operation. Here, m denotes the bit-size of the local storage of the ORAM. Our lower bound implies that logarithmic overhead in accesses is necessary, even if $\boldsymbol{b} \gg \boldsymbol{w}$. Our lower bound is tight for *all* settings of parameters, up to the $\log(\boldsymbol{b}/\boldsymbol{w})$ factor. Our bound also extends to the non-colluding multi-server setting.

As an application, we derive the first (unconditional) separation between the overhead needed for ORAMs in the *online* vs. *offline* models. Specifically, we show that when $\boldsymbol{w} = \log N$ and $\boldsymbol{b}, m \in \mathsf{poly}\log N$, there exists an offline ORAM that makes (on average) $o(1)$ memory probes per logical operation while every online one must make $\Omega(\log N/\log\log N)$ memory probes per logical operation. No such previous separation was known for any setting of parameters, not even in the balls and bins model.

Keywords: Oblivious RAM · Lower bound · Cell-probe model.

[⋆] The full version is posted on Cryptology ePrint Archive, Report 2020/1132 [29].

1 Introduction

An oblivious RAM (ORAM), introduced by Goldreich and Ostrovsky [22], is a probabilistic RAM machine whose goal is to simulate an arbitrary RAM program while ensuring observable access patterns do not reveal information neither about the underlying data nor about the program being executed. This is obtained by making sure that any two sequences of *logical* operations on the memory (either reads or writes) translate into *indistinguishable* sequences of physical probes to the memory. ORAMs have become an indispensable tool in the design of cryptographic systems where it is necessary to make the observable access pattern independent of the underlying sensitive data. Somewhat surprisingly, this task comes up not only in the context of software protection, as originally suggested by [22], but also in less directly related contexts such as the design of secure processor [15,16], secure multi-party computation [5,21,25,38,40,56], and other central notions in computer science [4,6,9,12,20,37,39,46,52,53,59,61].

A trivial way to construct an ORAM is to replace every logical access with a scan of the entire memory. While this solution is perfectly secure, it is highly inefficient and so the question is how efficient could an ORAM be compared to an insecure RAM. The primary efficiency metric of interest is:

I/O efficiency: The total number of physical probes to the memory of the ORAM amortized per logical operation.

Some previous works use bandwidth as the metric, but we chose to use I/O efficiency as our central metric since it is robust and well-defined in various ORAM settings. I/O efficiency can be translated into communication/bandwidth by multiplying by the ORAM cell size. See Remark 2.

Following Boyle and Naor [7], we shall distinguish between two classes of ORAM schemes: *offline* and *online*. An ORAM scheme is online if it supports accesses arriving in an online manner, one by one. An ORAM scheme is offline if it requires all accesses to be specified at once in advance. Most known ORAM constructions (e.g., [3,10,22,24,30,41,48,51,55]) work in the online setting as well with few exceptions (e.g., [7,28,47]). Also, most applications of ORAM schemes require that the scheme is online.

Existing lower bounds. Assume that the goal is to obliviously simulate a RAM of N cells each of size $\boldsymbol{w}$ bits on a RAM with N' cells each of size $\boldsymbol{b}$ bits and using a local storage of size m bits. In the original work of Goldreich and Ostrovsky [22] it was shown that any ORAM scheme (even offline ones) must have I/O efficiency[3] [4]

$$\Omega\left(\frac{\boldsymbol{w}}{\boldsymbol{b}} \cdot \frac{\log N}{1 + \log(m/\boldsymbol{b})}\right).$$

[3] To the best of our knowledge, the lower bound technique of [22] was never analyzed without assuming that $\boldsymbol{b} = \boldsymbol{w}$. For completeness, we add a proof in the full version [29]. The bound that we state here is a little bit simplified for presentation purposes.

[4] Throughout this paper, unless otherwise stated, log stands for $\log_2$.

In one sense, this lower bound is very powerful: (1) It is pretty robust to the choice of $\boldsymbol{w}$ and $\boldsymbol{b}$ as long as $\boldsymbol{b} = \boldsymbol{w}$, (2) it can be cast for few other efficiency metrics besides I/O (see [55] for details), and (3) it applies to schemes that have $O(1)$ statistical failure probability. However, as observed by Boyle and Naor [7] this lower bound only applies to schemes in the so called "balls and bins" model[5] which do not use cryptographic assumptions, leaving the possibility of more efficient constructions outside of this model.

In a beautiful recent work, Larsen and Nielsen [33, Theorem 2] proved a lower bound that applies to any online ORAM scheme, even ones that are not in the balls and bins model and ones that use cryptographic assumptions. They prove that any online ORAM must have I/O efficiency

$$\Omega\left(\frac{\boldsymbol{w}}{\boldsymbol{b}} \cdot \log\left(\frac{N\boldsymbol{w}}{m}\right)\right).$$

Similarly to the lower bound of Goldreich and Ostrovsky [22], this lower bound is also pretty robust to the choice of $\boldsymbol{b}$ and $\boldsymbol{w}$ as long as $\boldsymbol{b} = \boldsymbol{w}$.

Is sub-logarithmic efficiency possible? The above two lower bounds become completely trivial in the setting where, say, $\boldsymbol{w} = \log N$ and $\boldsymbol{b}, m \in \Theta(\log^2 N)$. In this case, both lower bounds simplify to $\Omega(1)$. This is by no means an esoteric setting of parameters. It is quite common and natural to consider RAM algorithms that take advantage of being able to place multiple elements in one cell and process all of them within a single memory access. Indeed, there is a long line of work in core algorithms literature designing efficient algorithms and studying tradeoffs in this setting (e.g., [2,17,23,54]).

Focusing on oblivious sorting, one notable result is due to Goodrich [23] (see also a follow-up by Chan et al. [11][6]) who showed an oblivious sorting algorithm that sorts N elements each of size $\boldsymbol{w}$ bits with $O((N\boldsymbol{w}/\boldsymbol{b}) \cdot \log_{m/\boldsymbol{b}}(N\boldsymbol{w}/\boldsymbol{b}))$ memory probes on a RAM with cells of size $\boldsymbol{b}$ bits and local storage of size m bits. Setting $\boldsymbol{w} = \log N$ and $\boldsymbol{b}, m \in O(\log^3 N)$ (see also the full version [29] for the parameterization), we obtain an oblivious sorting algorithm with $O(N)$ memory probes. In contrast, when $\boldsymbol{w} = \boldsymbol{b}$ we have existing $\Omega(N \cdot \log N)$ lower bounds on the number of memory probes, either in the balls and bins model [36] or assuming a well-known network coding conjecture [14].

Oblivious sorting is one of the core building blocks in the design of many oblivious RAM constructions (for example, [3,10,22,24,30,41]), suggesting that it may be possible to use the algorithms of [11,23] to get an ORAM construction with sub-logarithmic I/O efficiency. This direction was pursued first by Goodrich and Mitzenmacher [23,24] and then by Chan et al. [11], but they were only able

[5] In the balls and bins model, items are modeled as "balls", CPU registers and server-side data storage locations are modeled as "bins", and the set of allowed data operations consists *only* of moving balls between bins. See the full version [29] for the definition of the model.

[6] Chan et al. [11]'s algorithm has the same asymptotic efficiency and it is additionally in the balls and bins model.

to construct an ORAM with $O(\log N)$ I/O efficiency,[7] assuming that $\boldsymbol{w} = \log N$ and $\boldsymbol{b}, m \in O(\log^3 N)$. By now, we already have an ORAM construction, due to Asharov et al. [3], with $O(\log N)$ I/O efficiency assuming only $\boldsymbol{w} = \boldsymbol{b}$ and $m \in O(\boldsymbol{b})$.

Given the state of affairs, it is an intriguing question whether more efficient ORAM constructions exist when $\boldsymbol{b} \gg \boldsymbol{w}$:

Is the linear dependence on $\boldsymbol{w}/\boldsymbol{b}$ necessary? Alternatively,
is it possible to break the logarithmic barrier for ORAM efficiency if $\boldsymbol{b} \gg \boldsymbol{w}$?

1.1 Our Results

In this work, we answer the above question *negatively* by showing that any online ORAM construction, including ones that are not in the balls and bins model and perhaps use cryptographic assumptions, cannot go below the logarithmic I/O efficiency barrier even if $\boldsymbol{b} \gg \boldsymbol{w}$. Restricted to online schemes, for a wide ranges of parameters, our lower bound improves on the lower bound of Goldreich and Ostrovsky [22] as well as the one of Larsen and Nielsen [33]. Specifically, we prove the following theorem.

Theorem 1 (Informal; See Theorem 3). *Consider a RAM with memory of N cells, each of size $\boldsymbol{w}$ bits. Any online ORAM that simulates such a RAM using cells of size $\boldsymbol{b}$ bits and local storage of size m bits, must have I/O efficiency*

$$\Omega\left(\log\left(\frac{N\boldsymbol{w}}{m}\right) / \left(1 + \log\left(\frac{\boldsymbol{b}}{\boldsymbol{w}}\right)\right)\right).$$

When $\boldsymbol{b} = \boldsymbol{w}$, our lower bound is identical to the one of Larsen and Nielsen [33] and is at least as good as the one of Goldreich and Ostrovsky [22]. However, when $\boldsymbol{b} \in \omega(\boldsymbol{w})$, our lower bound is already better than both. For example, when $\boldsymbol{w} = \log N$ and $\boldsymbol{b}, m \in O(\log^c N)$ for any $c \geq 2$, our lower bound is $\Omega(\log N / \log\log N)$ while the ones of Goldreich and Ostrovsky [22] and Larsen and Nielsen [33] are both only $\Omega(1)$. As in [33]'s lower bound, our lower bound applies to ORAM schemes satisfying computational indistinguishability only with probability p and having δ failure probability in correctness for some fixed constants $0 < p, \delta < 1$. While this makes schemes somewhat weak, this only makes our lower bound stronger. Lastly, let us mention that our technique is pretty general and can be used to extend and improve other related lower bounds when $\boldsymbol{b} \gg \boldsymbol{w}$ (see Section 1.2 for pointers). For example, in the full version [29] we extend our lower bound to apply to the non-colluding multi-server setting, improving the recent lower bound of Larsen et al. [34] whenever $\boldsymbol{b} \gg \boldsymbol{w}$.

We remark that our lower bound in Theorem 1 is tight for all settings of parameters up to the $\log(\boldsymbol{b}/\boldsymbol{w})$ factor. This is due to the construction of Asharov

[7] Actually, these works [11, 24] give ORAM constructions in a more general model called the *external memory* model, where there are three entities, a CPU, a cache, and a memory. The standard ORAM setting (which we consider here) is a special case of that model.

et al. [3] who constructed an ORAM with $O(\log N)$ I/O efficiency for all values of $\boldsymbol{w} \geq \log N$ assuming only $m \geq \boldsymbol{b} \geq \boldsymbol{w}$ (and assuming that one-way functions exist).[8]

Separating offline and online ORAM. We use Theorem 1 to obtain the first separation between offline and online ORAM schemes. Specifically, we show that when we want to obliviously simulate a RAM with N cells of logarithmic size using a RAM with cells and local storage of poly-logarithmic size (in N), then there is an offline ORAM with $o(1)$ I/O efficiency while every online ORAM must have $\tilde{\Omega}(\log N)$ I/O efficiency. This separation is essentially optimal in terms of the gap between the cost of the offline and the online oblivious simulations.

Theorem 2 (Informal; See Theorem 6). *Consider the task of obliviously simulating a RAM with N cells each of size $\boldsymbol{w} = \log N$ bits using an ORAM with cells of size $\boldsymbol{b}$ bits and using local storage of size m bits such that $\boldsymbol{b}, m \in \mathsf{poly} \log N$. There exists an* offline *ORAM scheme with $o(1)$ I/O efficiency, while every* online *ORAM scheme for this task must have $\Omega(\log N/\log\log N)$ I/O efficiency.*

We emphasize that the separation is *unconditional* in the sense that it neither assumes that schemes are in the balls and bins model (for the lower bound), nor that one-way functions exist (for the upper bound). Prior to this work, there was no such separation, even assuming either of these assumptions (and in any range of parameters).

1.2 Related Work

Passive Server. It is implicit in the standard definition of an ORAM that the server merely acts as a storage provider and does not perform any computation for the client. There are constructions where the server is actively performing computation (including memory I/O) for the client and this is not counted in the total I/O efficiency of the scheme (e.g., [1, 13, 19–21, 45, 53]). Many of these schemes achieve sub-logarithmic client-side I/O efficiency. Our lower bound shows that, in such cases, the server must have logarithmic I/O efficiency.

Related oblivious lower bounds. The beautiful result and technique of Larsen and Nielsen [33] inspired a fruitful line of works [26,27,32,34,42,43]. Most related to the ORAM problem are [26,27,34,43] on which we briefly elaborate. Jacob et al. [27] showed that the lower bound technique of [33] can be used to show logarithmic lower bounds on the overhead of oblivious simulation of various specific data structures like stacks, queues, and more. Persiano and Yeo [43] showed that logarithmic overhead is necessary for RAM simulation even if the the security

[8] We believe that the $\log(\boldsymbol{b}/\boldsymbol{w})$ factor is necessary in the lower bound, at least for some range of parameters. Specifically, when $\boldsymbol{b}, m \in N^{\Theta(1)}$ and $\boldsymbol{w} = \log N$, by re-parameterizing Path ORAM [51], we obtain an ORAM with $O(1)$ I/O efficiency.

requirement is differential privacy, intuitively hiding only one access.[9] Hubáček et al. [26] extended [33]'s logarithmic lower bound to the setting where the adversary does not see boundaries between queries. Larsen et al. [34] showed that logarithmic overhead in oblivious simulation is necessary even if data is allowed to be split over multiple servers, only one of which is controlled by an attacker.

All of the above papers give lower bounds that mostly apply to the symmetric setting where the cell size is identical in the given RAM and the simulated one since they suffer from a $\boldsymbol{w}/\boldsymbol{b}$ factor loss. We believe that considering those problems and extending the lower bounds to the asymmetric setting (when possible) is intriguing, and we hope that our techniques in this paper will be helpful. In the full version [29], we show that using our techniques it is possible to improve the lower bound of Larsen et al. [34] to not suffer from a loss of $\boldsymbol{w}/\boldsymbol{b}$ multiplicative factor even in the multi-server setting. This lower bound generalized our main result (Theorem 1) as it implies the latter when restricting to a single server. We refer to the full version [29] for the precise problem definition and statement of the result.

We believe that similarly, using our technique, one can improve the results in [26, 42] as they rely on a similar hard distribution to that of [33]. This is left for future work.

The cell probe model. Following Larsen and Nielsen [33], our lower bound holds in an augmented version of the well known cell probe model (to capture the obliviousness requirement). Details about our model are given in Section 3; Here, we mention some classical and notable facts about the cell probe model. The cell probe model, introduced by Yao [60], is a model of computation similar to the RAM model, except that all computational operations are free of charge except memory access. This model is useful in the analysis of data structures, especially for proving lower bounds on the number of memory accesses needed to solve a given problem.

By now, there are few techniques for proving lower bounds in the cell probe model. The strongest technique [31,35] can prove super-logarithmic lower bounds and therefore should not be applicable as is to the ORAM setting where logarithmic upper bounds are known (unless additional requirements are made). Another technique, due to Pǎtraşcu and Demaine [44], is the so called *information transfer method* which is used to prove logarithmic lower bounds in the cell probe model. Larsen and Nielsen [33] were able to use this technique to prove their lower bound on ORAM constructions. We also use this technique. Persiano and Yeo's [43] lower bound, mentioned above, were able to adapt the *chronogram* technique due to Fredman and Saks [18] which can also be used to prove logarithmic lower bounds.

Other related work. In the balls and bins model and where the server is passive (i.e., not performing any computation), Cash et al. [8] proved that any

[9] The lower bound of Persiano and Yeo [43] also looses the $\boldsymbol{w}/\boldsymbol{b}$ factor, similarly to Larsen and Nielsen. Specifically, it is $\Omega((\boldsymbol{w}/\boldsymbol{b})\cdot\log(N/m))$ which is trivial if $\boldsymbol{b} \gg \boldsymbol{w}$. It is an open problem to improve their lower bound in the setting where $\boldsymbol{b} \gg \boldsymbol{w}$.

one-round ORAM must have either $\Omega(\sqrt{N})$ I/O efficiency or $\Omega(\sqrt{N})$-bit local storage.

Boyle and Naor [7] proved that an unconditional lower bounds for offline ORAMs would imply a non-trivial circuit lower bound which is a long standing open problem. This result is obtained by constructing an offline ORAM from any sorting circuit, where the efficiency of the resulting ORAM is proportional to the size of the circuit. In a followup work, Weiss and Wichs [58] showed that proving a lower bound for *online read-only* ORAM is at least as hard as either proving a non-trivial circuit lower bound or ruling out a very good locally decodable code.

As mentioned, some ORAM constructions have improved I/O efficiency at the cost of setting the cell size $\boldsymbol{b}$ to be super-logarithmic in the memory size. These works include not only schemes based on oblivious sorting [11, 23, 24, 53], but also several "tree-based" constructions [48, 51].

2 Technical Overview

This section gives a high level overview of our results. We first briefly recall the model and problem we want to solve. We proceed with explaining the beautiful technique of Larsen and Nielsen [33] and why it fails to give our desired lower bound. Lastly, building on the intuition we gained up to that point, we explain the main ideas in our proof and highlighting some of the technical challenges we are faced with.

2.1 The Model, Problem, and Recap of Larsen and Nielsen [33]

The model and problem. As observed by Larsen and Nielsen [33], it is convenient to state the ORAM problem as an oblivious data structure, as defined in [57], solving the *array maintenance* problem, where the goal is to maintain an array of N entries, each of size $\boldsymbol{w}$ bits, while supporting two operations: (1) (write, a, x): set the content of entry $a \in [N]$ to $x \in \{0,1\}^{\boldsymbol{w}}$ and (2) (read, a): return the content of entry $a \in [N]$. The lower bound that we prove, identical to [33], is on the *cell probe complexity* of any oblivious data structures solving the array maintenance problem. To get a lower bound on the I/O efficiency of ORAMs, it suffices to divide the number of probes by the number of operations.

Briefly, an oblivious data structure is a data structure that solves some given problem with an additional security guarantee which says that the (physical, observable) access patterns resulting from a sequence of logical data structure operations should reveal nothing on the latter sequence other than its length. For this purpose the oblivious data structure can use a small trusted/secure local storage ("cache") on which it can perform operations "for free" and without leaking any data. The oblivious data structure is therefore parametrized by $N', \boldsymbol{b}, m$, its total number of cells, the bit-size of each cell, and the bit-size of its local storage, respectively. The efficiency metric of interest is the *number of probes* to the physical memory needed to answer one logical access. It is typically assumed that $m \geq \boldsymbol{b} \geq \log N'$ so that the local storage can hold at least a single cell from the memory and that a single cell can hold a pointer to another cell.

Throughout most of this overview (except where we explicitly say otherwise), we consider the simpler setting where the oblivious data structure has *perfect* security and correctness. Perfect security means that for all sequence of logical operations of the same length, the observable sequence of physical memory probes is identically distributed. Perfect correctness means that the data structure never makes mistakes. With some additional technical work, these two assumptions can be relaxed.

Larsen and Nielsen's lower bound. The lower bound of Larsen and Nielsen [33] adapts the *information transfer* technique of Pătraşcu and Demaine [44] to the oblivious setting. We give a high level overview next. Fix a given oblivious data structure for the array maintenance problem (i.e., an ORAM). For any sequence of N operations, we associate a complete binary tree with N leaves (we assume that N is a power of two for simplicity). The leaves are associated with the logical operations and their associated physical probes, in chronological order. That is, during the execution of the sequence, for each i, all cell addresses probed during the ith operation are associated with the ith leaf. Next, the leaf-level probes are *partially assigned* to internal nodes: for each probe to cell address q that is associated with leaf i, if chronologically the most recent probe to cell q happened during the jth operation (so that $j < i$), then the probe (i, q) is *assigned* to the lowest common ancestor of leaves i and j. Notice that the assignment is partial, i.e., some physical probes may not be assigned to any internal node, and thus it suffices to prove a lower bound on the total number of probes assigned to internal nodes.

For each fixed internal node v, Larsen and Nielsen [33] used the information transfer technique [44] to prove a lower bound on the number of associated physical probes with v by designing a hard distribution of sequences of operations. Let n be the number of leaves and thus operations in the subtree induced by v. In the hard distribution, all $N - n$ operations that are not in the subtree of v are just dummy reads from a fixed address. In the subtree induced by v, the first $n/2$ operations are writes to addresses $1, 2, \ldots, n/2$ with uniformly random values $x_1, \ldots, x_{n/2} \leftarrow \{0,1\}^{\boldsymbol{w}}$, and then the second $n/2$ operations are reads from addresses $1, 2, \ldots, n/2$. That is,

$$(\mathsf{write}, 1, x_1), \ldots, (\mathsf{write}, n, x_{n/2}), \quad (\mathsf{read}, 1), \ldots, (\mathsf{read}, n/2).$$

To show that node v is associated with "many" probes when executing a sequence of operations from this distribution, the intuition is that in order to *correctly answer* the $n/2$ read operations, any data structure for the array maintenance problem (even non-oblivious ones!) must probe "many" cells that were also probed during the $n/2$ write operations. This intuition is formalized by a compression argument. Quantitatively, recalling that each cell in the array maintenance problem consists of $\boldsymbol{w}$ bits and each cell in the data structure consists of $\boldsymbol{b}$ bits, there must exist a set of $\Omega(n \cdot \boldsymbol{w}/\boldsymbol{b})$ cells from the data structure that are probed during the first as well as the second $n/2$ operations (here, we ignore the local storage of m bits for simplicity). By the definition of our binary tree, all of these $\Omega(n \cdot \boldsymbol{w}/\boldsymbol{b})$ probes are associated with node v.

The proof proceeds by using the security guarantee of the data structure (as the above argument relied solely on correctness). The main observation is that since the tree and the associated probes of each node are efficiently computable by the adversary who only sees physical probes, then by security, the number of associated probes of each node must be the same for *all sequences of operations*. Namely, if node v is associated with $\Omega(n \cdot \boldsymbol{w}/\boldsymbol{b})$ probes when executing the hard distribution, then node v must also be associated with $\Omega(n \cdot \boldsymbol{w}/\boldsymbol{b})$ probes when executing any other sequence of operations of the same length; otherwise, an adversary can easily distinguish the two. Since the tree is a complete binary tree with N leaves, by summation there are $\Omega(N \cdot (\boldsymbol{w}/\boldsymbol{b}) \cdot \log N)$ associated probes to internal nodes which implies their lower bound.

Losing the $\boldsymbol{w}/\boldsymbol{b}$ term is inherent when using the hard distribution designed by Larsen and Nielsen [33]. Recall that in their distribution we first write random values to addresses $1, \ldots, n/2$ and then read those addresses in order. Indeed, using only correctness, each probe can carry information regarding $\boldsymbol{b}/\boldsymbol{w}$ values and so the whole sequence of writes can be read using only $O(n \cdot \boldsymbol{w}/\boldsymbol{b})$ probes. The fundamental reason for the loss is therefore that the sequence of addresses in the read phase is completely determined a priori and the data structure can use this information during the write phase to organize data cleverly.

2.2 Our Hard Distribution and Information Transfer Tree

We propose the following hard distribution of sequences of $n + k \leq N$ operations. The first n operations are writes to addresses $1, 2, \ldots, n$ with uniformly random values $x_1, \ldots, x_n \leftarrow \{0,1\}^{\boldsymbol{w}}$ (same as in [33]). Then, in the last k operations, instead of sequentially reading from those addresses, we perform read from *uniformly random* words $a_1, a_2, \ldots, a_k \leftarrow [n]$. That is,

$$(\mathsf{write}, 1, x_1), \ldots, (\mathsf{write}, n, x_n), \quad (\mathsf{read}, a_1), \ldots, (\mathsf{read}, a_k).$$

Indeed, now the sequence of reads is not known during the write phase so we avoid the aforementioned optimization the construction can use. But is this the only optimization? We prove that it is. The intuition is that no matter how large the cell size $\boldsymbol{b}$ is, no matter how the data structure scheme processes the n write requests, in order to read from a uniformly random address $a_i \in [n]$ correctly, the construction must probe at least one cell (unless the construction got lucky and the corresponding value to address a_i was accidentally in the local storage). That is true only because the address a_i is chosen both randomly and online and therefore any pre-computation or pre-fetching that uses the fact that cells are moderately large is useless. In a high level, using a *compression argument* we show that for $k \leq n \cdot \boldsymbol{w}/\boldsymbol{b}$, the following holds:

> *Lemma:* Any *correct* data structure solving the array maintenance problem when fed a length $n+k$ sequence of requests sampled from our hard

distribution, must probe $\Omega(k)$ cells during the read phase that were also probed during the write phase.

Whenever $\boldsymbol{b} \in \omega(\boldsymbol{w})$, this lower bound is better than the $\Omega(k \cdot \boldsymbol{w}/\boldsymbol{b})$ lower bound obtained with Larsen and Nielsen's hard distribution. We note that we are only able to prove that the above statement holds with high-enough probability, smaller than 1 (which is enough to carry out the rest of the argument). Indeed, there will always be "easy" read sequences, like the one of Larsen and Nielsen, where the number of necessary probes will be smaller. Finally, we emphasize that in the above lemma, the read phase consists of only k operations (which differs from Larsen and Nielsen's hard distribution which has n reads). This is specially designed to work with the information transfer tree that we will introduce below.

This lemma is central to our proof and while it may seem intuitively correct, the actual proof turns out to require very delicate and non-trivial probability analysis. We will get back to this in Section 2.3, where we will explain the main challenges and describe our solutions. Meanwhile, we proceed to explain how the lemma is used to derive the final lower bound using a generalized version of the information transfer tree described above.

Revisiting the information transfer tree. Recall that in the partial assignment of Larsen and Nielsen [33], a probe to a cell is assigned to a node v only if v is the lowest common ancestor between the probe and the most recent probe to the same cell. However, if a cell is probed 100 times during the read phase corresponding to v (i.e., v's right subtree), it will be counted and associated to v at most once! Working out the details, it turns out that even if we use our improved lemma from above in the binary tree approach, we would still lose the $\boldsymbol{w}/\boldsymbol{b}$ factor. Therefore, we need to find a more fine-grained way to account for multiple probes to the same cell during the read phase.

Our solution is to consider a tree with larger arity so that we could count several probes to the same cell during the read phase of a given node (i.e., with multiplicity). We let χ, the arity of the tree, be proportional to $\boldsymbol{b}/\boldsymbol{w}$ and consider a complete χ-ary tree with N leaves. Consider a node v that has an induced subtree of $2n$ leaves and consider an associated sequence of n writes followed by n reads. Divide the n read operations into $\chi/2$ equal-size groups so that each group has $k \triangleq n/(\chi/2)$ reads. For each such group we imagine a child node which is "in charge" of this group. Let the children of v that correspond to the read phase be $u_1, \ldots, u_{\chi/2}$ so that each u_i is in charge of k disjoint read operations. Next in the partial assignment, we associate with v index-cell pairs of the form (i, q), where i is an index from $[\chi/2]$ and q is a physical address of a probed cell. The index i tells us from which group the probe came and q tells us to which cell. Intuitively, this allows us to count probes to the same cell q with multiplicity, distinguishing them by the value of i. (In comparison, Larsen and Nielsen [33] only associated q's to nodes and so they do not distinguish multiple accesses to the same cell.) See Figure 1 for an illustration.

Using our Lemma. Our lemma from above almost fits this framework. To prove that a group of k operations associated to node u_i introduces $\Omega(k)$ accesses that are counted in v, we slightly modify the hard distribution to consist of a

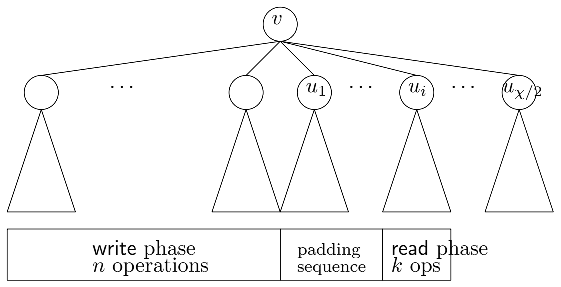

Fig. 1: Hard distribution on χ-ary tree.

padding sequence of read operations (say from address 1) between the write phase and the reads that u_i is in charge of. Summing up over all u_i's, the node v will be associated with $\Omega(\chi \cdot k) = \Omega(n)$ index-cell pairs, which is our goal and the best one can hope for.

The last step, where we use the obliviousness of the data structure in order to argue that any sequence of operations behaves as "the hardest one", is similar to Larsen and Nielsen [33]. Recall that the tree is of depth $\log_\chi N$, the arity is χ, and for each level d, there are χ^d nodes at that level each has associated $\Omega(N/\chi^d)$ probes. Therefore, we get a lower bound of $\Omega(N \cdot \log N/\log(\boldsymbol{b}/\boldsymbol{w}))$ probes to perform N operations. This is essentially the lower bound claimed in Theorem 1, omitting the size of local storage m (which we ignored throughout this overview and only complicates the proof slightly).

Remark 1 (Relation to [44]). Pătraşcu and Demaine [44, Section 7] consider a related problem in a somewhat different context. There, they observe that the basic information transfer method suffers from the $\boldsymbol{w}/\boldsymbol{b}$ factor loss. To remedy the situation they propose a new hard distribution, similar to ours, and also propose to consider an information transfer tree with higher arity, as we do. Essentially, our proof could be seen as an extension of their technique to the oblivious setting. The latter introduces many technical challenges, especially in the compression argument, as we elaborate next.

2.3 Our Compression Argument

Recall that our hard sequence consists of n writes to fixed addresses $1, \dots, n$ of uniformly random values followed by $k \leq n \cdot \boldsymbol{w}/\boldsymbol{b}$ reads from uniformly random addresses from $[n]$.[10] Our goal is to argue that during the read phase, $\Omega(k)$ distinct cells must be probed. Let us refer to the write sequence as L and the read sequence as R (for left- and right-side). Denote by $\mathsf{Cells}(L)$ the cells probed

[10] In fact, as mentioned we will need to consider an augmented sequence that has a padding sequence of reads from some fixed address in between the write sequence and the read sequence mentioned above. This will complicate the argument slightly so for simplicity we ignore it here.

during the execution of the L sequence of accesses and by $\mathsf{Cells}(R)$ the cells probed during the execution of the R sequence (after executing the L sequence). Note that $L, R, \mathsf{Cells}(L), \mathsf{Cells}(R)$ are all random variables. We want to prove that *with high probability* $|\mathsf{Cells}(L) \cap \mathsf{Cells}(R)| \in \Omega(k)$. That is, for some constant $\epsilon < 1$,

$$\mathbf{Pr}\left[|\mathsf{Cells}(L) \cap \mathsf{Cells}(R)| \geq \epsilon k\right] > 3/4, \tag{1}$$

where the probability is over the choice of L and R, and the randomness of the ORAM which influences $\mathsf{Cells}(L)$ and $\mathsf{Cells}(R)$.

The proof is done via a compression argument where we imagine two communication parties Alice and Bob. Alice gets as input $\boldsymbol{x} = x_1, \ldots, x_n \leftarrow \{0,1\}^{\boldsymbol{w}}$ (chosen uniformly at random) and she sends one message to Bob who is able to recover $\boldsymbol{x}$. If the message sent by Alice contains $< n \cdot \boldsymbol{w}$ bits, we get a contradiction. To this end, we assume that Inequality (1) is false, namely that the read phase can be implemented with ϵk probes for some small enough ϵ, and use that to get a too good to be true encoding scheme. This implies a contradiction, as needed. This proof is somewhat technical so we provide some intuition on how it works and refer to the technical section for full details.

Warmup: an expectation argument. It is insightful to first prove a weaker statement (which does not suffice for us) and then explain how to improve it. Here, we argue that

$$\mathbf{E}\left[|\mathsf{Cells}(L) \cap \mathsf{Cells}(R)|\right] \geq \epsilon k. \tag{2}$$

The proof is by contradiction, namely, we assume that Inequality (2) is false and obtain an impossible compression scheme. To this end, Alice and Bob share a long string $\mathcal{S}$ that is chosen completely independent of the input to Alice. The string consists of (1) a sequence of k addresses $a_1, \ldots, a_k \leftarrow [n]$ that define the R, (2) a random tape ρ for the ORAM, and (3) an integer $t \leftarrow [k]$ sampled uniformly at random. Note that even conditioned on the shared string $\mathcal{S}$, the entropy in the input to Alice, namely $x_1, \ldots, x_n \leftarrow \{0,1\}^{\boldsymbol{w}}$, is still $n\boldsymbol{w}$. Therefore, by Shannon's source coding theorem, the only way for Alice to correctly transmit them to Bob is by sending at least $n\boldsymbol{w}$ bits.

In a high level, Alice splits the indices $[n]$ into two groups: *easy* and *hard*. An index i is easy if Bob can learn value x_i *without* making a probe to $\mathsf{Cells}(L)$, that is, a probe to a cell that was written to during the write sequence. All other indices are hard. By our assumption, the set of hard indices cannot be too large. Alice sends those hard values explicitly to Bob. To learn the values corresponding to easy indices, we use the correctness of the data structure to transfer them. The challenging part is for Alice to determine which index is easy and which is hard. Alice does this by seeing how likely it is to make the probe in $\mathsf{Cells}(L)$ from a given index by "planting" that index in the random read operation given in $\mathcal{S}$ (while keeping the rest of the operations fixed). If any $\mathsf{Cells}(L)$-probe occurs, this index is considered hard, otherwise it is easy. A more precise description follows.

Alice's encoding on input $n\boldsymbol{w}$ bits interpreted as $x_1, \ldots, x_n \in \{0,1\}^{\boldsymbol{w}}$:

1. Using the ORAM, Alice executes the sequence of operations (L, R) prescribed by $x_1, \ldots, x_n$ and $a_1, \ldots, a_k$. Then, Alice sends the contents of *overlapping cells* (yielded by the execution) to Bob, where the overlapping cells are defined as the cells probed during the write sequence L and then probed during the read sequence R (i.e., $\mathsf{Cells}(L) \cap \mathsf{Cells}(R)$).
2. For each $i \in [n]$, Alice replaces the tth read with operation (read, i) and (using the ORAM) executes the replaced sequence, that is, the sequence $(L, \widehat{R}_{t,i})$ where

$$L := \underbrace{(\mathsf{write}, 1, x_1), \ldots, (\mathsf{write}, n, x_n)}_{\mathsf{write}\text{ phase}},$$

$$\widehat{R}_{t,i} := (\mathsf{read}, a_1), \ldots, (\mathsf{read}, a_{t-1}), \underbrace{(\mathsf{read}, i)}_{\text{planted } \mathsf{read}}, (\mathsf{read}, a_{t+1}), \ldots, (\mathsf{read}, a_k).$$

   Depending on the probed locations induced by (read, i), do:
   (a) If (read, i) probes at least one cell that was written to during the write phase (i.e., in $\mathsf{Cells}(L)$), then i is called *hard.* Alice sends value (i, x_i) directly to Bob.
   (b) Otherwise, (read, i) probes no cell in $\mathsf{Cells}(L)$ and i is called *easy.* Alice sends nothing to Bob as Bob can recover x_i by executing (read, i) himself.

On Bob's side, the hard x_i's are received from Alice directly, while the easy x_i's are recovered by executing (read, i) planted as the tth read operation, that is, after the prefix $(\mathsf{read}, a_1), \ldots, (\mathsf{read}, a_{t-1})$. Bob indeed recovers all easy x_i's correctly: Bob received the content of the overlapping cells that suffice to execute the prefix read sequence.

Analyzing the size of the message from Alice to Bob is a bit more challenging. In a high level, Alice's message consists of just two parts, the contents of overlapping cells and the values of "hard" inputs. By assumption (Inequality (2) is false), the number of overlapping cells is ϵk and so the first part consists of at most $\epsilon k \boldsymbol{b} \leq \epsilon n \boldsymbol{w}$ bits. For the second part, roughly speaking, we consider all possible samples of $(a_1, a_2, \ldots, a_k, t) \in [n]^k \times [k]$ while fixing $x_1, \ldots, x_n$. By assumption, with probability at most ϵ, (read, a_t) is hard, which means that at most ϵ fraction of all such samples are hard. Then, for any set of n distinct samples, on average, there are at most ϵn hard samples. Noticing that Alice's procedure is choosing a random set of n samples, we conclude that *in expectation* there are ϵn hard samples, which means ϵn hard x_i's on average. It follows that the second part of Alice's message consumes $\epsilon n \boldsymbol{w}$ bits, and then the total message length is $2\epsilon n \boldsymbol{w}$ bits, which is a contradiction when ϵ is small enough.

The high probability argument. Recall that in the last step of lower bound proof we need to move from a claim about the load of a node in the information transfer tree to the load of the same node under any other input sequence of operations. Since security only holds with constant probability, this step loses a constant factor and therefore we need our original compression argument to hold *with high probability* and not just in expectation.

This complicates the compression argument as follows. Now, Alice cannot just send the content of the overlapping cells directly to help Bob answer easy queries (for which it uses the correctness of the data structure), since there is no bound on the expected number of overlapping cells. Instead, we modify Alice's procedure to distinguish between two cases, either sending the overlapping cells directly is too expensive or it is not. In the latter case, we need to analyze and bound the number of hard indices i *conditioned* on the event that the number of overlapping cells is small. This requires delicate conditional probability analysis on which we elaborate next. In the former case, there is no compression since Alice just sends all $x_1, \ldots, x_n$ in the clear but we can show that this case does not happen too often due to the assumption (Inequality (1) is false).

Specifically, the most challenging is to prove that *conditioned on the overlapping cells set being small*, the expected size of the set of hard indices is bounded by a sufficiently small constant times n. Let $\mathsf{Good}_{L,R}$ be the conditioned event. What we show is that if $\beta < 3/4$ is a constant for which $\mathbf{Pr}\left[|\mathsf{Cells}(L) \cap \mathsf{Cells}(R)| \geq \epsilon k\right] = \beta$ (our assumption, see Inequality (1)), then:

Lemma: $\mathbf{E}\left[|H| \mid \mathsf{Good}_{L,R}\right] < (\beta + \epsilon/(1-\beta))n$.

We define $\mathsf{Good}_{L,\widehat{R}_{t,i}}$ similarly as the event when the overlapping cells between $(L, \widehat{R}_{t,i})$ is small. By linearity of expectation and the law of total probability:

$$\begin{aligned}
\mathbf{E}\left[|H| \mid \mathsf{Good}_{L,R}\right] &= \sum_{i\in[n]} \mathbf{Pr}[i \in H \mid \mathsf{Good}_{L,R}] \\
&= \sum_{i\in[n]} \mathbf{Pr}[i \in H \wedge \neg\mathsf{Good}_{L,\widehat{R}_{t,i}} \mid \mathsf{Good}_{L,R}] + \\
&\quad \sum_{i\in[n]} \mathbf{Pr}[i \in H \wedge \mathsf{Good}_{L,\widehat{R}_{t,i}} \mid \mathsf{Good}_{L,R}].
\end{aligned}$$

We now bound each of these terms separately. It is rather easy (though a bit technical) to bound the second term. Specifically, we show that $\sum_{i\in[n]} \mathbf{Pr}[i \in H \wedge \mathsf{Good}_{L,\widehat{R}_{t,i}} \mid \mathsf{Good}_{L,R}] \leq \epsilon n/(1-\beta)$. Indeed, for each $i \in [n]$, $\mathbf{Pr}[i \in H \wedge \mathsf{Good}_{L,\widehat{R}_{t,i}} \mid \mathsf{Good}_{L,R}] \leq \mathbf{Pr}[i \in H \wedge \mathsf{Good}_{L,\widehat{R}_{t,i}}]/\,\mathbf{Pr}[\mathsf{Good}_{L,R}]$. So, the denominator is exactly $1-\beta$. The fact that the nominator is bounded by ϵ follows from the definition of $\mathsf{Good}_{L,\widehat{R}_{t,i}}$.

The bound on the first term is much more interesting. In words, the event we are trying to bound corresponds to sampling the sequences L and R and then $\widehat{R}_{t,i}$ and asking what is the probability that $\mathsf{Good}_{L,\widehat{R}_{t,i}}$ occurs *conditioned* on $\mathsf{Good}_{L,R}$ occurring (ignoring event $i \in H$). To analyze this event, we recall that $\widehat{R}_{t,i}$ is obtained by *resampling* the tth operation in R. So, what is the probability that by resampling only one read operation in R we suddenly do not satisfy the event Good? We prove a general lemma that *partial resampling* cannot reduce the probability beyond a certain point! Here is a simple variant of the lemma (we state and prove a more general version in the full version [29]):

Partial Resampling Lemma: Consider two independent random variables X and Y. Let Y^* be an independent random variable distributed identically to Y. Let f be an arbitrary Boolean function. Then,

$$\mathbf{Pr}[f(X, Y^*) = 1 \mid f(X, Y) = 1] \geq \mathbf{Pr}[f(X, Y) = 1].$$

This means that if the event $\mathsf{Good}_{L,R}$ occurs, then it must also occur in $\mathsf{Good}_{L,\widehat{R}_{t,i}}$ with good probability. Plugging in the assumption, we can bound the second term by βn.

Together, the two bounds imply that $\mathbf{E}\left[|H| \mid \mathsf{Good}_{L,R}\right] < (\beta + \epsilon/(1-\beta))n$, as needed.

3 The Model

This section introduces the model in which our lower bound is proven. As in previous works [27,33,43], we start-off with the cell probe model, first described by Yao [60]. Traditionally, this model is used to prove lower bounds for word-RAM data structures and is extremely powerful in the sense that it allows arbitrary computations and only charges for memory accesses.

In a high-level, the cell probe model models the interaction between a CPU and a memory. The memory is modeled as a word-RAM, that is, an array of cells such that each cell can contain at most $\boldsymbol{b}$ bits. The CPU can perform operations on the memory, namely, either reading the content of some cell or overwriting the content of some cell. An algorithm executed in this setting is charged one unit of cost on every operation it makes (read or write) and all computation based on the contents of probed cells is free of charge.

Whereas this model captures traditional data structures, it does not capture data structures that have privacy requirements for the stored data and/or the operations performed. Indeed, the latter are usually modeled in the client-server model, where a client wishes to outsource data to server while retaining the ability to perform computation over the data. At the same time, the client wishes to hide the performed operations as well as the contents of its data cells from the server who sees the entire memory and the memory accesses. To address this gap, Larsen and Nielsen [33] introduced the *Oblivious Cell Probe Model*, an augmented version of the cell probe model. We briefly introduce this model next, mostly following Larsen and Nielsen.

Data structure problems. A data structure problem in the oblivious cell probe model is defined by a tuple $(\mathcal{U}, \mathcal{Q}, \mathcal{O}, f)$, where $\mathcal{U}$ is a universe of update operations, $\mathcal{Q}$ is a universe of queries, and $\mathcal{O}$ is an output domain. Furthermore, there is a query function $f\colon \mathcal{U}^* \times \mathcal{Q} \to \mathcal{O}$. For a sequence of updates $u_1, \dots, u_M \in \mathcal{U}$ and a query $q \in \mathcal{Q}$, we say that the answer to the query q after updates $u_1, \dots, u_M$ is $f(u_1, \dots, u_M, q)$.

Oblivious Cell Probe Data Structures. An oblivious cell probe data structure for a given data structure problem $\mathcal{P} = (\mathcal{U}, \mathcal{Q}, \mathcal{O}, f)$, consists of a randomized algorithm implementing the update and query operations for $\mathcal{P}$. The data structure is parametrized by three integers m, $\boldsymbol{b}$, and N', denoting the client storage and cell size in bits, and the number of cells respectively. We follow the standard assumption $\log N' \leq \boldsymbol{b}$ so that any cell can store the address of any other cell. We further assume that the data structure has access to a finite string of randomness ρ of length ℓ. The parameter ℓ can be arbitrary large and so ρ can contain a random oracle. Fixing ρ, the algorithm DS is deterministic. As such, the data structure can be described by a decision tree T_{op} for every operation $\mathsf{op} \in \mathcal{U} \cup \mathcal{Q}$, i.e., it has one decision tree for every possible operation in the data structure problem. Each node in the decision tree is labelled by an index indicating the location to probe in the memory (held by the server). The decision of which path to continue to in the tree depends on the answer to the probe to the memory and small local information stored by the client.

More precisely, each node in the decision tree T_{op}, where $\mathsf{op} \in \mathcal{U} \cup \mathcal{Q}$, is labeled by an address $i \in [N']$ and it has one child for every triple of the form $(m_0, c_0, \rho) \in \{0,1\}^m \times \{0,1\}^{\boldsymbol{b}} \times \{0,1\}^{\ell}$. Each edge to a child is further labeled by $(j, m_1, c_1) \in [N'] \times \{0,1\}^m \times \{0,1\}^{\boldsymbol{b}}$. To process an operation op, the oblivious cell probe data structure starts its execution at the root of the tree and traverses from root to leaf. When visiting a node v in this traversal, labelled with some address $i_v \in [N']$, it probes the memory cell i_v. If C denotes its content, M denotes the current contents of the client memory and ρ denotes the random bit-string, the process continues by descending to the child of v corresponding to the tuple (M, C, ρ). If the edge to the child is labelled (j, m_1, c_1), then the memory cell of address j has its contents updated to c_1 and the client memory is updated to m_1. We say that memory cell j is *probed*. The execution stops when reaching a leaf. Each leaf v of the decision tree T_{op}, where $\mathsf{op} \in \mathcal{Q}$, is labeled with an element ans_v in $\mathcal{O}$ (the answer to the query). We say that the oblivious cell probe data structure returns ans_v as its answer to the query op.

I/O efficiency. The I/O efficiency of an oblivious data structure is related to the depth of the decision tree as each edge corresponds to a cell probe. Furthermore, our model assumes that the server is passive, i.e., it can only update or retrieve a cell for the client.

Definition 1 (Expected amortized I/O efficiency). *An oblivious cell probe data structure has expected amortized I/O efficiency $t(M)$ on a sequence y of M operations from $\mathcal{U} \cup \mathcal{Q}$ if the total number of memory probes is no more than $t(M) \cdot M$ in expectation. The expectation is taken over the random choice of the randomness $\rho \in \{0,1\}^{\ell}$. An oblivious cell probe data structure has expected amortized I/O efficiency $t(M)$ if it has expected amortized I/O efficiency $t(M)$ on all sequences y of operations from $\mathcal{U} \cup \mathcal{Q}$.*

Remark 2 (Other efficiency notions). There are few other metrics of efficiency of interest in the context of ORAM constructions. It is common to consider the *bandwidth* efficiency of a construction, namely, the communication complexity consumed by the construction when processing a sequence of operations, amortized per operation. This is equal to $\boldsymbol{b}$ times the I/O efficiency. Vice versa, if the amortized bandwidth of a construction is $t(\cdot)$, then the I/O efficiency of that construction is $t/\boldsymbol{b}$.

Thus, there is a $Q = Q(N, \boldsymbol{b}, \boldsymbol{w})$ lower bound on I/O efficiency if and only if there is a $\boldsymbol{b} \cdot Q$ lower bound on bandwidth. For example, suppose that $\boldsymbol{w} = \log N$ and $\boldsymbol{b}, m \in \Theta(\log^2 N)$. Then, the previously known lower bound [33] says that $\Omega(\log^2 N)$ amortized bandwidth is necessary (that is $\Omega(1)$ I/O efficiency), but our improved lower bound says that $\Omega(\log^3 N/\log\log N)$ bandwidth is necessary (that is $\Omega(\log N/\log\log N)$ I/O efficiency).

It is also common to measure the complexity of an ORAM construction in the language of efficiency *overhead* (either I/O or bandwidth) where we compare the ratio between the efficiency of the ORAM and the efficiency of the insecure RAM. This makes complete sense when $\boldsymbol{b} = \boldsymbol{w}$, but when $\boldsymbol{b} \in \omega(\boldsymbol{w})$ it is more confusing since the basic unit of cost (cell size) is different between the two settings. Some papers do explicitly distinguish between $\boldsymbol{w}$ and $\boldsymbol{b}$ [49, 50, 53, 58] and measure complexity correctly. For clarity, we will avoid the term *overhead*.

Correctness and security. Let $y = (\mathsf{op}_1, \ldots, \mathsf{op}_M)$ be a sequence of M operations to the given data structure problem, where each $\mathsf{op}_i \in \mathcal{U} \cap \mathcal{Q}$. For an oblivious cell probe data structure, define the (possibly randomized) probe sequence on y as the tuple:

$$\mathsf{Access}(y) = (\mathsf{Access}(\mathsf{op}_1), \ldots, \mathsf{Access}(\mathsf{op}_M)),$$

where $\mathsf{Access}(\mathsf{op}_i)$ is the sequence of memory addresses probed while processing op_i. More precisely, let $\mathsf{Access}(y; \rho) := (\mathsf{Access}(\mathsf{op}_1; \rho), \ldots, \mathsf{Access}(\mathsf{op}_M; \rho))$ be the deterministic sequence of operations when the random bit-string fixed to ρ and let $\mathsf{Access}(y)$ be the random variable describing $\mathsf{Access}(y; \rho)$ for a random $\rho \in \{0,1\}^\ell$.

Definition 2 (Correctness and security). *An oblivious cell probe data structure is said to be δ-correct and ϵ-secure if the following two properties hold:*

– ***Security****: For any two data request sequences y and z of the same length M, their probe sequences $\mathsf{Access}(y)$ and $\mathsf{Access}(z)$ cannot be distinguished with probability better than ϵ by an algorithm which is polynomial time in $M + \log|\mathcal{U}| + \log|\mathcal{Q}| + \boldsymbol{b}$.*
– ***Correctness****: The oblivious cell probe data structure has failure probability at most δ, namely, for every sequence and any operation op in the sequence, the data structure answers op correctly with probability at least $1 - \delta$.*

ORAM is array maintenance. As observed in previous work [33], the definition of an online ORAM coincides with the definition of an oblivious data structure (see [57]) solving the *array maintenance problem.* In this problem, the goal is to maintain an array of N entries, each of size $\boldsymbol{w}$ bits, while allowing write and read operations, where (write, i, a) sets the content of the ith cell to the value a and (read, i) return the content of the ith cell (for $i \in [N]$ and $a \in \{0,1\}^{\boldsymbol{w}}$).

Therefore, in order to prove a lower bound on the I/O efficiency of an ORAM scheme, it suffices to prove a lower bound on the I/O efficiency of any correct and secure data structure for the array maintenance problem in the oblivious cell probe model.

Remark 3 (Operation boundaries). We follow Larsen and Nielsen [33] and assume that the adversary sees which cell access belongs to which operation from y. Hubáček et al. [26] were able to extend the lower bound of Larsen and Nielsen [33] to account for this gap. We suspect that our techniques and lower bound could be extended to capture this stronger setting, but it is left for future work.

4 An ORAM Lower Bound

This section is devoted to the proof of our lower bound on the I/O efficiency of oblivious cell probe data structures solving the array maintenance problem. As mentioned, such a lower bound directly implies an I/O efficiency lower bound for online ORAMs. Our main theorem is stated next.

Theorem 3 (Main theorem). *Let $\mathcal{DS}$ be an oblivious cell probe data structure for the array maintenance problem on arrays of N entries, each of size $\boldsymbol{w}$ bits. Let N' denote the number of cells in $\mathcal{DS}$, $\boldsymbol{b}$ denote the cell size in bits, and m denote the number of bits of client memory. Assume that $16 \leq \boldsymbol{w} \leq \boldsymbol{b}$ and $\boldsymbol{w} \leq m \leq N\boldsymbol{w}$.*

If $\mathcal{DS}$ is $(1/128)$-correct and $(1/4)$-secure, then there is a sequence of $\ell \in (N/(2\lceil \boldsymbol{b}/\boldsymbol{w} \rceil), N]$ operations such that the expected amortized I/O efficiency of $\mathcal{DS}$ on this sequence is

$$\Omega\left(\frac{\log(N\boldsymbol{w}/m)}{1+\log\lceil \boldsymbol{b}/\boldsymbol{w} \rceil}\right).$$

In particular, when $\boldsymbol{w} \leq m \leq N^{1-\epsilon}$ for $\epsilon > 0$, $\boldsymbol{b} = \log^c N$ for $c > 1$, and $\boldsymbol{w} = \log N$, the I/O efficiency is $\Omega\left(\frac{\log N}{\log\log N}\right)$. The rest of this section is devoted to the proof of Theorem 3.

Proof (Proof of Theorem 3). We start with the following definition.

Definition 3 (Set of probed cells). *Given a length M sequence of operations, $\mathsf{seq} = (\mathsf{op}_1, \ldots, \mathsf{op}_M)$, define $\mathsf{Cells}(\mathsf{op}_i \mid \mathsf{op}_1, \ldots, \mathsf{op}_{i-1})$ as the* set *of addresses of (physical) cells accessed by $\mathcal{DS}$ during its execution of operation op_i* after *executing the sequence $(\mathsf{op}_1, \ldots, \mathsf{op}_{i-1})$. Similarly, given seq and $i, j \in [M]$ such that $i < j$, $\mathsf{Cells}(\mathsf{op}_i, \mathsf{op}_{i+1}, \ldots, \mathsf{op}_j \mid \mathsf{op}_1, \ldots, \mathsf{op}_{i-1})$ is defined as the set of addresses of cells accessed by $\mathcal{DS}$ during its execution of operations $(\mathsf{op}_i, \ldots, \mathsf{op}_j)$* after *executing the sequence $(\mathsf{op}_1, \ldots, \mathsf{op}_{i-1})$.*

Notice that we define $\mathsf{Cells}(\mathsf{op}_i \mid \mathsf{op}_1, \ldots, \mathsf{op}_{i-1})$ as a set and so its cardinality does not account for multiplicities. Therefore, we will use the sum of cardinalities $\sum_{i\in[M]} \left|\mathsf{Cells}(\mathsf{op}_i \mid \mathsf{op}_1, \ldots, \mathsf{op}_{i-1})\right|$ as a lower bound on the total number of accesses made by $\mathcal{DS}$.

We now construct the information transfer tree. Fix ℓ to be a power of $\chi := 2\lceil \boldsymbol{b}/\boldsymbol{w} \rceil$ in the range $(N/(2\lceil \boldsymbol{b}/\boldsymbol{w} \rceil), N]$. Let T be the complete χ-ary tree consisting of ℓ leaves (see Figure 2 for visualization). For any sequence of operations $\mathsf{seq} = (\mathsf{op}_1, \ldots, \mathsf{op}_\ell)$, for each $i \in [\ell]$, we associate op_i to the ith leaf of T. Additionally, $\mathsf{Cells}(\mathsf{op}_i \mid \mathsf{op}_1, \ldots, \mathsf{op}_{i-1})$ (i.e., the addresses of cells accessed by $\mathcal{DS}$ during its execution of the ith operation in the sequence) are associated to the same ith leaf. For each accessed cell q that is associated with a leaf i, we map q to at most one internal node v of T, where v is an ancestor of i. This is described next.

First, for each internal $v \in \mathsf{T}$, we define a set of index-cell pairs, $P_v(\mathsf{seq})$, as follows. A pair of index-cell $(i, q) \in [\ell] \times [N']$ is in $P_v(\mathsf{seq})$ if and only if

- i is a leaf in the subtree induced by v and $q \in \mathsf{Cells}(\mathsf{op}_i \mid \mathsf{op}_1, \ldots, \mathsf{op}_{i-1})$,
- There exists $j < i$ such that $q \in \mathsf{Cells}(\mathsf{op}_j \mid \mathsf{op}_1, \ldots, \mathsf{op}_{j-1})$,
- For all $j' \in \{j+1, \ldots, i-1\}$, it holds that $q \notin \mathsf{Cells}(\mathsf{op}_{j'} \mid \mathsf{op}_1, \ldots, \mathsf{op}_{j'-1})$, and
- The lowest common ancestor of i and j is v.

Notice that each cell access $q \in \mathsf{Cells}(\mathsf{op}_i \mid \mathsf{op}_1, \ldots, \mathsf{op}_{i-1})$ during the execution of op_i is assigned to at most one $v \in \mathsf{T}$. Hence, for any seq and execution of $\mathcal{DS}$, we have that

$$\sum_{i \in [\ell]} \left| \mathsf{Cells}(\mathsf{op}_i \mid \mathsf{op}_1, \ldots, \mathsf{op}_{i-1}) \right| \geq \sum_{v \in \mathsf{T}} |P_v(\mathsf{seq})|.$$

We conclude the proof of the theorem using the following lemma whose proof is given below.

Lemma 1. *Let $\epsilon := 1/128$. Fix any sequence seq consisting of ℓ operations. Let $v \in \mathsf{T}$ be an internal node whose subtree consists of at least $2 \cdot \max\{8, m/(\epsilon \boldsymbol{w})\}$ leaves. For any $(1/4)$-secure and $(1/128)$-correct $\mathcal{DS}$ against ℓ operations, it holds that*

$$\mathbf{E}\left[|P_v(\mathsf{seq})|\right] \geq \epsilon \cdot \ell/(4\chi^{d(v)}),$$

where $d(v)$ is the depth of v (i.e. the distance from v to the root).

Let us first explain why Lemma 1 implies Theorem 3. Let d^* be the maximum depth for which Lemma 1 applies. Summing over all nodes in T, by linearity of expectation, we have that

$$\mathbf{E}\left[\sum_{v \in \mathsf{T}} |P_v(\mathsf{seq})|\right] = \sum_{v \in \mathsf{T}} \mathbf{E}\left[|P_v(\mathsf{seq})|\right] \geq \sum_{v \in \mathsf{T}, d(v) \in [0, d^*]} \mathbf{E}\left[|P_v(\mathsf{seq})|\right] \geq (d^*+1) \cdot \epsilon \ell/4,$$

where the last inequality follows by Lemma 1. Since Lemma 1 applies to any node v that has at least $2 \cdot \max\{8, m/(\epsilon \boldsymbol{w})\}$ leaves in its induced subtree, we have

$$d^* := \left\lfloor \log_\chi \left\lceil \frac{\ell}{2 \cdot \max\{8, m/(\epsilon \boldsymbol{w})\}} \right\rceil \right\rfloor \in \Omega\left(\frac{\log(N\boldsymbol{w}/m)}{1 + \log(\boldsymbol{b}/\boldsymbol{w})}\right)$$

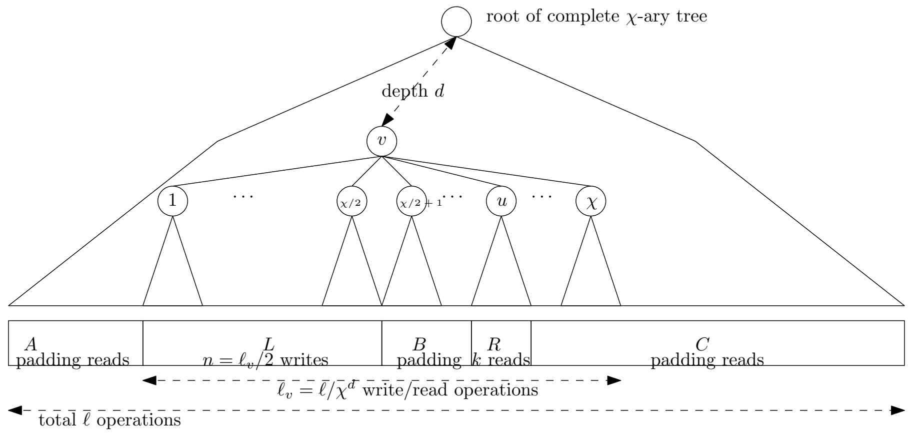

Fig. 2: The ditribution $\mathcal{D}(v,u)$ of hard sequences for the parent-child pair (v,u) in the complete χ-ary tree of ℓ leaves. Each leaf is associated with a read or write operation, and the hard sequence is the operations from the left-most to the right-most leaves. Given the internal node v and its child u where u is in the right-side of the subtree induced by v, we focus on the operations in the Left-side of the subtree of v, i.e., L part, and on the operations in the subtree induced by u, i.e., R (for right-side) part. The L part is $n = \ell_v/2$ write operations to fixed locations with random contents (where $\ell_v = \ell/\chi^d$ is the number of leaves in the subtree of v), and the R part is $k = \ell_v/\chi$ read operations from random locations that were written in L part. The remaining parts A, B, C are all padding operations that just read the fixed location 1. The overall hard sequence is then (A, L, B, R, C).

for all $m, \boldsymbol{b}, \boldsymbol{w}, N$ such that $\boldsymbol{b} \geq \boldsymbol{w} \geq 16$ and $\boldsymbol{w} \leq m \leq N\boldsymbol{w}$ (which ensure that the logs are nonnegative). Hence, for any seq of ℓ operations, the expected number of accesses is lower bounded by $\ell \cdot \Omega\left(\frac{\log(N\boldsymbol{w}/m)}{1+\log(\boldsymbol{b}/\boldsymbol{w})}\right)$, which concludes the proof of Theorem 3.

We conclude this section with the proof of Lemma 1. Note that this proof will rely on Theorem 5 which is stated and proved in Section 5.

Proof (Proof of Lemma 1). Recall that $P_v(\mathsf{seq})$ consists of pairs of index-cell pairs (i,q) such that during the ith operation $\mathcal{DS}$ accesses physical cell q and also the most recent access to q was made at some operation $j < i$ such that j is a leaf in the induced subtree of v and v is the the lowest common ancestor of i and j. Denote $P_{v,u}(\mathsf{seq})$ the subset of (i,q) in $P_v(\mathsf{seq})$ that result from an operation i that happens in the subtree induced by u. It holds that

$$|P_v(\mathsf{seq})| = \sum_{u \text{ is a child of } v} |P_{v,u}(\mathsf{seq})|. \tag{3}$$

We therefore prove a lower bound on each $|P_{v,u}(\mathsf{seq})|$. To this end, for a given pair of parent-child, (v,u), in the tree, we design a distribution of access

$\mathsf{seq}_{\mathsf{hard}}$ which causes $|P_{v,u}(\mathsf{seq}_{\mathsf{hard}})|$ to be large with high probability. We then use the security guarantee of $\mathcal{DS}$, ensuring that the access pattern resulting from executing any seq must be indistinguishable, and therefore the same large number of probes must occur on any input sequence. That is, $|P_{v,u}(\mathsf{seq})|$ is large with high probability. We give the hard distribution next.

The hard distribution. To describe the distribution of hard sequences, we set up some notation. Specifically, we will explain how to "split" a given length ℓ sequence of operations w.r.t a given internal node $v \in \mathsf{T}$.

- Let $d := d(v)$ be the depth of the node v, and let $l := l(v) \in \left[\chi^d\right]$ be the index of v in the dth level.
- Let $\ell_v := \ell/\chi^d$ be the number of leaves in the subtree induced by v. Set $n := \ell_v/2$, and $k := \ell_v/\chi$.
- Recall that v has χ children. Let $U := \{\chi/2+1, \chi/2+2, \ldots, \chi\}$ be the set of indices of second half children of v (i.e., the right half of children). Given $u \in U$, we slightly abuse notation and say that the uth child of v is u.

Because our goal is to bound the number of probes during the subtree of u, we choose to perform n writes during the first n leaves of v, and then perform k reads during the k leaves of $u \in U$ (Figure 2). The remaining parts are just padding to ℓ operations. Formally, the distribution of hard sequence $\mathcal{D}(v,u)$, with induced parameters l, ℓ_v, n, k as above, is sampled as follow:

1. Let A be the sequence consisting of $(l-1)\cdot\ell_v$ dummy reads, i.e., repeating $(\mathsf{read}, 1)$ for $(l-1)\cdot\ell_v$ times.
$$A := \underbrace{(\mathsf{read}, 1), \ldots, (\mathsf{read}, 1)}_{(l-1)\cdot\ell_v \text{ times}},$$
2. Let L (for left-side) be the sequence of n writes to fixed locations with random words, i.e.,
$$L := (\mathsf{write}, 1, x_1), (\mathsf{write}, 2, x_2), \ldots, (\mathsf{write}, n, x_n),$$
where $x_1, \ldots, x_n \leftarrow \{0,1\}^{\boldsymbol{w}}$ are chosen independently uniformly at random.
3. Let B be the sequence consisting of $k\cdot(u-1)-n$ dummy reads,
$$B := \underbrace{(\mathsf{read}, 1), \ldots, (\mathsf{read}, 1)}_{k\cdot(u-1)-n \text{ times}},$$
4. Let R (for right-side) be the sequence of k reads from random addresses in $[n]$, i.e.,
$$R := (\mathsf{read}, a_1), (\mathsf{read}, a_2), \ldots, (\mathsf{read}, a_k),$$
where $a_1, \ldots, a_k \leftarrow [n]$ are chosen independently uniformly at random.
5. Let C be the sequence of dummy reads whose goal is to pad the whole sequence to length ℓ,
$$C := \underbrace{(\mathsf{read}, 1), \ldots, (\mathsf{read}, 1)}_{\ell-(l-1)\cdot\ell_v-u\cdot k \text{ times}},$$

* **Output** the concatenated length ℓ sequence

$$\mathsf{seq}_{\mathsf{hard}} = A, L, B, R, C.$$

We are interested in the set of cells that are touched both during the L, B sequence and during the R sequence, i.e., the set $\mathsf{Cells}(L, B \mid A) \cap \mathsf{Cells}(R \mid A, L, B)$ (see Definition 3 for $\mathsf{Cells}(\dots)$ notation). By definition, it holds that

$$|P_{v,u}(\mathsf{seq}_{\mathsf{hard}})| \geq |\mathsf{Cells}(L, B \mid A) \cap \mathsf{Cells}(R \mid A, L, B)|.$$

In Theorem 5 we prove the following.

Theorem 4 (See Theorem 5). *Let $\delta := 1/128$ and $\epsilon := 1/128$. If $\mathcal{DS}$ is δ-correct (for the array maintenance problem), then as long as $n \in [\max\{8, m/(\epsilon \boldsymbol{w})\}, N]$ and $k \leq n \cdot \boldsymbol{w}/\boldsymbol{b}$, it holds that*

$$\mathbf{Pr}\left[|\mathsf{Cells}(L, B \mid A) \cap \mathsf{Cells}(R \mid A, L, B)| \geq \epsilon k\right] > 3/4.$$

Indeed, observe that the conditions to apply this theorem are met since $k \leq n\boldsymbol{w}/\boldsymbol{b}$ as $n = \ell_v/2$, $k = \ell_v/\chi$, and $\chi = 2\lceil \boldsymbol{b}/\boldsymbol{w} \rceil$. Also, since v is an internal node whose induced subtree consists of $\ell_v \geq 2 \cdot \max\{8, m/(\epsilon \boldsymbol{w})\}$ leaves, we also have $n \in [\max\{8, m/(\epsilon \boldsymbol{w})\}, N]$. Therefore, $\mathbf{Pr}\left[|P_{v,u}(\mathsf{seq}_{\mathsf{hard}})| \geq \epsilon k\right] > 3/4$.

Due to the security guarantee of $\mathcal{DS}$, we deduce that for any (equal-length) sequence seq the above should hold. Namely, denoting the randomness of the $\mathcal{DS}$ by ρ, we have

$$\mathop{\mathbf{Pr}}_{\mathsf{seq}_{\mathsf{hard}}, \rho}[|P_{v,u}(\mathsf{seq}_{\mathsf{hard}})| \geq \epsilon k] - \mathop{\mathbf{Pr}}_{\rho}[|P_{v,u}(\mathsf{seq})| \geq \epsilon k] \leq 1/4.$$

Therefore, we obtain that $\mathbf{Pr}\left[|P_{v,u}(\mathsf{seq})| \geq \epsilon k\right] > 1/2$ and so $\mathbf{E}\left[|P_{v,u}(\mathsf{seq})|\right] > \epsilon k/2$. Using Eq. (3) and linearity of expectation we obtain that

$$\begin{aligned}
\mathbf{E}[|P_v(\mathsf{seq})|] &= \mathbf{E}\left[\sum_{u \text{ is a child of } v} |P_{v,u}(\mathsf{seq})|\right] \\
&= \sum_{u \text{ is a child of } v} \mathbf{E}\left[|P_{v,u}(\mathsf{seq})|\right] \\
&> (\chi/2) \cdot (\epsilon k/2) = \epsilon \ell/(4\chi^d).
\end{aligned}$$

5 The Compression Argument

Let $\mathcal{DS}$ be an oblivious cell probe data structure for the array maintenance problem on arrays of N entries, each of $\boldsymbol{w}$ bits. Let N' denote the number of cells in $\mathcal{DS}$, let $\boldsymbol{b}$ denote the bit-length of each cell, and let m denote the number of bits of client memory.

Consider the following distribution over sequences of operations given to $\mathcal{DS}$. The distribution is denoted $\mathcal{D}_{A,B,n,k}$ and it is parametrized by two sequences of operations A and B, and by two positive integers $n, k \leq N$. The sequence

A consist of arbitrary reads and writes (A is going to be a prefix sequence) and B consist of arbitrary reads but *no* writes (B is going to be a padding sequence). Each sequence of operations sampled from $\mathcal{D}_{A,B,n,k}$ consists of 4 parts, A, L, B, R, in this order, where L (for left-side) is a sequence of n writes to fixed addresses $1, \ldots, n$ with uniformly random data, and R (for right-side) is a sequence of k reads from uniformly random indices in $[n]$. The full sequence (A, L, B, R) looks as follows:

$A:$ Fixed sequence of reads and writes;

$L:$ $(\mathsf{write}, 1, x_1), (\mathsf{write}, 2, x_2), \ldots, (\mathsf{write}, n, x_n)$,
where $x_1, \ldots, x_n \leftarrow \{0,1\}^{\boldsymbol{w}}$, chosen uniformly at random;

$B:$ Fixed sequence of reads;

$R:$ $(\mathsf{read}, a_1), (\mathsf{read}, a_2), \ldots, (\mathsf{read}, a_k)$,
where $a_1, \ldots, a_k \leftarrow [n]$, chosen uniformly at random.

Recall that in Definition 3, given a sequence of operations (X, Y) and randomness ρ, we let $\mathsf{Cells}(Y \mid X)$ be the set of addresses of (physical) cells probed by $\mathcal{DS}$ during its execution of the Y sequence *after* executing the X sequence.[11] For example, in an instance of sequence (A, L, B, R) sampled from our distribution, (1) $\mathsf{Cells}(L, B \mid A)$ contains the (physical) addresses of cells probed by $\mathcal{DS}$ during the execution of the L and B parts after executing the A sequence, and (2) $\mathsf{Cells}(R \mid A, L, B)$ contains the (physical) addresses of cells probed by $\mathcal{DS}$ during the execution of the R sequence after executing the A, L, and B sequences. We prove the following theorem.

Theorem 5. *Let $\delta := 1/128$, $\epsilon := 1/128$ and $\alpha := 3/4$. Further, fix integers $n \in [\max\{8, m/(\epsilon\boldsymbol{w})\}, N]$, $\boldsymbol{w} \geq 16$, and $k \leq n \cdot \boldsymbol{w}/\boldsymbol{b}$. Lastly, fix arbitrary sequences A and B as above. Then, if $\mathcal{DS}$ is δ-correct (for the array maintenance problem), then it holds that*

$$\mathbf{Pr}\left[|\mathsf{Cells}(L, B \mid A) \cap \mathsf{Cells}(R \mid A, L, B)| \geq \epsilon k\right] > \alpha,$$

where the probability is taken over the choice of L and R (i.e., over the choice of (A, L, B, R) from $\mathcal{D}_{A,B,n,k}$), and over the internal randomness of $\mathcal{DS}$.

In order to prove Theorem 5, we assume for contradiction that the statement is false, namely that there are A, B, n, k as in the theorem statement and a $\beta \leq \alpha$ for which

$$\mathbf{Pr}\left[|\mathsf{Cells}(L, B \mid A) \cap \mathsf{Cells}(R \mid A, L, B)| \geq \epsilon k\right] = \beta. \tag{4}$$

To reach a contradiction, we construct a randomized compression scheme that encodes $n\boldsymbol{w}$ uniformly random bits into a message that is less than $n\boldsymbol{w}$ bits. Section 5.1 describes the encoding and decoding procedure of such compression,

[11] Notice that $\mathsf{Cells}(Y \mid X)$ is a *set* of addresses, whereas $\mathsf{Access}(X \| Y)$ is a *sequence* of addresses.

and it also shows the compression is correct. We then in Section 5.2 prove that the expected size of the encoding is less then $n\boldsymbol{w}$ bits, which is a contradiction to Shannon's source coding theorem and concludes the proof of Theorem 5.

The reader may find it helpful to first read the full version [29] where we prove a weaker version of Theorem 5. Specifically, we show that the *expected* size of the intersection of both sets from Theorem 5 is $\Omega(k)$ (rather than that it holds with high probability).

5.1 The Encoding and Decoding Procedures

The encoder, Alice, gets as input the $n\boldsymbol{w}$ random bits interpreted as $x_1, \ldots, x_n \in \{0,1\}^{\boldsymbol{w}}$, and the decoder, Bob, aims to recover $x_1, \ldots, x_n$. Our compression scheme uses a long string which is shared by Alice and Bob but is completely independent of $x_1, \ldots, x_n$. This shared string consists of

- Fixed $\mathsf{read}/\mathsf{write}$ sequence A and read-only sequence B;
- A sequence R of k reads where the indices are sampled uniformly at random (i.e., (read, a_1), $(\mathsf{read}, a_2), \ldots, (\mathsf{read}, a_k)$, where $a_1, \ldots, a_k \leftarrow [n]$);
- An integer $t \leftarrow [k]$ sampled uniformly at random; and
- A random tape ρ used by $\mathcal{DS}$.

Since $x_1, \ldots, x_n$ are sampled independently and uniformly, their entropy conditioned on the shared string is $n\boldsymbol{w}$. Therefore, by Shannon's source coding theorem, the only way for Alice to correctly transmit them to Bob is by sending at least $n\boldsymbol{w}$ bits.

Alice's encoding:

- Input: $n\boldsymbol{w}$ bits interpreted as $x_1, \ldots, x_n \in \{0,1\}^{\boldsymbol{w}}$.
- Procedure:
  1. Using ρ and $\mathcal{DS}$, execute the sequence of requests
  $$A, L, B, R,$$
  where A, B, and R are taken from the shared string, and $L := (\mathsf{write}, 1, x_1), (\mathsf{write}, 2, x_2), \ldots, (\mathsf{write}, n, x_n)$. Define the following collections of cells' indices that are physically probed during the execution:
     - $C_0 := \mathsf{Cells}(L, B \mid A)$. That is, the cells probed during the execution of the L, B sequences.
     - $C := C_0 \cap \mathsf{Cells}(R \mid A, L, B)$. That is, the cells probed during the execution of the L, B sequences which are also probed during the execution of the R sequence.

     Right after executing A, L, B using ρ, let σ be the local state of $\mathcal{DS}$, and let $\mathsf{content}(C)$ be the contents of the cells in C.
  2. Define $R[1 \ldots t-1] := (\mathsf{read}, a_1), \ldots, (\mathsf{read}, a_{t-1})$ to be the sequence of operations that consists of the first $t-1$ reads from R. For each $i \in [n]$, define $\widehat{R}_{t,i}$ to be a sequence of operations that consists of $R[1 \ldots t-1]$ and then, as its tth operation, it performs a read from index i. That is,
  $$\widehat{R}_{t,i} := (\mathsf{read}, a_1), \ldots, (\mathsf{read}, a_{t-1}), (\mathsf{read}, i).$$

3. For each $i \in [n]$, using ρ and $\mathcal{DS}$, execute the sequence of operations

$$A, L, B, \widehat{R}_{t,i}.$$

   - We say that $i \in [n]$ (or $\widehat{R}_{t,i}$ correspondingly) is *easy* iff

$$\mathsf{Cells}((\mathsf{read}, i) \mid A, L, B, R[1 \ldots t-1]) \cap C_0 = \emptyset,$$

     and *hard* otherwise. Let $H \subset [n]$ be the set of hard i's and $h := (x_i)_{i \in H}$ (written in increasing order w.r.t. i).
   - For each $i \in [n]$, add i into set H_0 iff $\mathcal{DS}$ answers operation (read, i) *incorrectly* (after the execution of $A, L, B, R[1 \ldots t-1]$). That is, let $i \in H_0$ iff the answer to (read, i) is not x_i. Let $h_0 := (x_i)_{i \in H_0}$ (written in increasing order w.r.t. i).

– Output:
   - If $|C| \geq \epsilon k$, output a bit 0, followed by $\mathsf{msg}_0 := (x_1, \ldots, x_n)$.
   - Else (i.e., $|C| < \epsilon k$), output a bit 1, followed by $\mathsf{msg}_1 := (\sigma,\ C,\ \mathsf{content}(C),\ H, h, H_0, h_0)$.

Bob's decoding:

– Input from Alice is either
   - the first bit is 0, followed by $\mathsf{msg}_0 := (x'_1, \ldots, x'_n)$, or
   - the first bit is 1, followed by $\mathsf{msg}_1 := (\sigma, C, \mathsf{content}(C), H, h, H_0, h_0)$.

– Procedure:
   1. If the first bit is 0, output the received $x'_1, \ldots, x'_n$ directly. Otherwise, continue as follows.
   2. For each hard $i \in [n]$, i.e., $i \in H$, recover x'_i by reading it from h (recall that elements in h are ordered in increasing i).
   3. For each incorrect index $i \in H_0$, recover x'_i by reading it from h_0 (recall that elements in h_0 are ordered in increasing i).
   4. For each easy and correct $i \in [n]$, i.e., $i \notin H \cup H_0$, recover x'_i using the following steps:
      (a) Using $\mathcal{DS}$ and randomness ρ, execute the sequence of operations A. Then, replace the content of cells in C with $\mathsf{content}(C)$ and replace the local state of $\mathcal{DS}$ with σ.
      (b) Using this configuration, randomness ρ, and $\mathcal{DS}$, execute $\widehat{R}_{t,i}$ and let x'_i be the result of the tth operation in $\widehat{R}_{t,i}$, i.e., (read, i).

– Output: $x'_1, \ldots, x'_n$.

Correctness of compression. For correctness, we show that Bob always outputs values $x'_1, \ldots, x'_n$ such that $x'_i = x_i$ for all $i \in [n]$, where $x_1, \ldots, x_n$ are the inputs of Alice. Whenever $|C| \geq \epsilon k$, correctness holds immediately since Alice just sends $x_1, \ldots, x_n$ explicitly to Bob. We therefore consider the case where $|C| < \epsilon k$. For every hard $i \in H$ or incorrect $i \in H_0$, we have $x'_i = x_i$ by construction (since it is transmitted explicitly as part of h or h_0). For each easy and correct $i \in [n]$, executing $\widehat{R}_{t,i}$ (using $\mathcal{DS}$, local state σ, and random tape ρ) needs only the contents of cells either in C or not in C_0 (observe that $R[1 \ldots t-1]$ needs both and then easy (read, i) needs only those not in C_0). Bob can obtain the content of these cells not in C_0 by executing the sequence of operations A. Hence, all the needed information can be obtained by Bob and it is identical to that of Alice. Recall that sequence B is read-only so the output is indeed x_i written by L. Therefore, by correctness of $\mathcal{DS}$ (as writes to and reads from $i \in [n] \subseteq [N]$ are valid operations), Bob indeed obtains $x'_i = x_i$ for all $i \in [n]$.

5.2 Encoding Size Analysis

We upper bound the expected size of the encoding outputted by Alice. We follow the conventions that i) $|s|$ denotes the *number of bits* of s for any *sequence* s, and ii) $|S|$ denotes the *cardinality* of S for any *set* S.

The encoding consists of a bit j and the message msg_j, where j depends on whether $|C| \geq \epsilon k$. Let Good be the indicator for the event that $|C| < \epsilon k$. By the law of total expectation, the expected size is the sum of two cases,

$$\mathbf{E}\left[\left|j, \mathsf{msg}_j\right|\right] = 1 + \mathbf{E}\left[|\mathsf{msg}_0| \mid \neg\mathsf{Good}\right] \cdot \mathbf{Pr}\left[\neg\mathsf{Good}\right] + \mathbf{E}\left[|\mathsf{msg}_1| \mid \mathsf{Good}\right] \cdot \mathbf{Pr}\left[\mathsf{Good}\right].$$

By Eq. (4), we have

$$\mathbf{Pr}[\mathsf{Good}] = 1 - \beta \quad \text{and} \quad \mathbf{Pr}[\neg\mathsf{Good}] = \beta, \tag{5}$$

and by construction, $|\mathsf{msg}_0|$ is always $n\boldsymbol{w}$ bits. We thus focus on proving an upper bound on the second conditional expectation, namely on $\mathbf{E}[|\mathsf{msg}_1| \mid \mathsf{Good}]$.

Recall that the encoding msg_1 consists of $\sigma, C, \mathsf{content}(C), H, h, H_0, h_0$ and so by linearity of expectation, it suffices to bound the expected size of each component marginally. First, since the local state of $\mathcal{DS}$ is m bits, we know that $|\sigma| \leq m$. Second, by the definition of the event Good, we have that

$$\mathbf{E}\left[|C| \mid \mathsf{Good}\right] < \epsilon k \text{ and } \mathbf{E}\left[|\mathsf{content}(C)| \mid \mathsf{Good}\right] < \epsilon k \boldsymbol{b},$$

where the latter inequality follows since each cell consists of $\boldsymbol{b}$ bits. Third, for H_0 and h_0, we have $\mathbf{E}[|H_0|] \leq \delta n$ by δ-correctness of $\mathcal{DS}$ and then linearity of expectation. Hence, we have $\mathbf{E}[|h_0|] \leq \delta n \boldsymbol{w}$ without conditioning on Good. That is, it takes just $\delta n \boldsymbol{w}$ bits even if Alice had always sent h_0.

We are therefore left with upper bounding the number of hard read requests $\widehat{R}_{t,i}$, namely, the cardinality of H. For this, we use the fact that the tth read request is online and is made after the previous $t-1$ requests are executed. That is, after executed $t-1$ requests where $\mathcal{DS}$ reads cells in C, the set C is fixed. Then, when given the tth request, $\mathcal{DS}$ must touch a new cell not in C (unless it got lucky and it was already in C). Intuitively, this means that $\mathcal{DS}$ must spend probes in order to answer the tth random read request (no matter how many probes were spent on write requests and on previous read requests). Formalizing this intuition into a bound on $|H|$ is done in the following Lemma (see the full version [29] for the proof).

Lemma 2. *Assuming Eq.* (4)*, then* $\mathbf{E}\left[|H| \mid \mathsf{Good}\right] < (\beta + \epsilon/(1-\beta))n$.

Expected size of encoding. We now sum up the expected size of msg_1 sent by Alice conditioned on the case Good. Recall that σ takes m bits, and C and $\mathsf{content}(C)$ consume together at most $2\epsilon k\boldsymbol{b}$ bits conditioned on Good. The set H can be described simply using a binary string of n bits, where the ith bit indicates whether $i \in H$ or not. To describe h, by Lemma 2, h can be described with at most $(\beta + \epsilon/(1-\beta))n\boldsymbol{w}$ bits in expectation. The set H_0 is described using n bits as well, but we defer h_0 since its expectation is not conditional. So, the expected size conditioned on Good is

$$\begin{aligned} m + 2\epsilon k\boldsymbol{b} + n + (\beta + \epsilon/(1-\beta)) \cdot n\boldsymbol{w} + n &\le m + 2\epsilon n\boldsymbol{w} + n + (\beta + \epsilon/(1-\beta)) \cdot n\boldsymbol{w} + n \\ &\le (3\epsilon + \epsilon/(1-\beta) + 1/8 + \beta) \cdot n\boldsymbol{w}, \end{aligned}$$

where the first inequality follows since $k \le n\boldsymbol{w}/\boldsymbol{b}$, and the second is since $m \le \epsilon n\boldsymbol{w}$ and $\boldsymbol{w} \ge 16$. The total expected size is then

$$\begin{aligned} &1 + \mathbf{E}[|h_0|] + \mathbf{E}[|\mathsf{msg}_0| \mid \neg\mathsf{Good}] \cdot \mathbf{Pr}[\neg\mathsf{Good}] + \mathbf{E}[|\mathsf{msg}_1| \mid \mathsf{Good}] \cdot \mathbf{Pr}[\mathsf{Good}] \\ &\le 1 + \delta n\boldsymbol{w} + n\boldsymbol{w} \cdot \mathbf{Pr}[\neg\mathsf{Good}] + (3\epsilon + \epsilon/(1-\beta) + 1/8 + \beta) \cdot n\boldsymbol{w} \cdot \mathbf{Pr}[\mathsf{Good}] \\ &= 1 + \delta n\boldsymbol{w} + n\boldsymbol{w}\beta + (3\epsilon + \epsilon/(1-\beta) + 1/8 + \beta) \cdot (1-\beta) \cdot n\boldsymbol{w}, \end{aligned}$$

where the first term 1 is the bit indicating if the case is Good in Alice's encoding, and the last equality follows since $\mathbf{Pr}[\mathsf{Good}] = 1 - \beta$ (Eq. (5)). Plugging in $\delta = 1/128, \epsilon = 1/128$ and $\beta \le \alpha = 3/4$, we obtain that the expected encoding size is strictly smaller than

$$1 + (1/128 + 3/4 + (1/4)(1/16 + 1/8 + 3/4)) \cdot n\boldsymbol{w} < n\boldsymbol{w}$$

in bits, where the inequality follows since $n \ge 8$ and $w \ge 16$. By Shannon's source coding theorem, we thus reached a contradiction which completes the proof of Theorem 5.

6 Separating Offline and Online ORAM

In this section we prove a separation between the offline and online ORAM models. Concretely, we prove the following result.

Theorem 6. *Consider the task of obliviously simulating a RAM with N cells each of size $\boldsymbol{w} = \log N$ bits using a RAM of N' cells each of size $\boldsymbol{b}$ bits and using local memory of size m bits for $\boldsymbol{b}, m \in \mathsf{poly}\log N$. There exists an* offline *ORAM scheme with $N' \in O(N)$ for this task with $o(1)$ I/O efficiency, while every* online *ORAM scheme for this task must have $\Omega(\log N/\log\log N)$ I/O efficiency (no matter how large N' is).*

Proof. The lower bound follows directly from Theorem 3. Plugging in the values of $\boldsymbol{w}, \boldsymbol{b}, m$, we get that every *online* ORAM scheme for this task must have $\Omega(\log N/\log\log N)$ I/O efficiency. The upper bound follows from existing results [7, 11] and is deferred to the full version [29].

Acknowledgements

The first author thanks Paul Grubbs for a discussion that motivated him to look into the lower bound of [33] more closely. We also thank Elaine Shi for useful discussions. I. K. is supported in part by an Alon Young Faculty Fellowship and by an ISF grant (No. 1774/20). W.-K. L. is supported in part by a DARPA Brandeis award. This work was done partly while W.-K. L. worked at NTT Research.

References

1. Abraham, I., Fletcher, C.W., Nayak, K., Pinkas, B., Ren, L.: Asymptotically tight bounds for composing ORAM with PIR. In: PKC (2017)
2. Aggarwal, A., Vitter, Jeffrey, S.: The Input/Output Complexity of Sorting and Related Problems. Commun. ACM **31**(9), 1116–1127 (Sep 1988)
3. Asharov, G., Komargodski, I., Lin, W., Nayak, K., Peserico, E., Shi, E.: Optorama: Optimal oblivious RAM. In: EUROCRYPT (2020)
4. Bindschaedler, V., Naveed, M., Pan, X., Wang, X., Huang, Y.: Practicing oblivious access on cloud storage: the gap, the fallacy, and the new way forward. In: ACM CCS. pp. 837–849 (2015)
5. Boyle, E., Chung, K., Pass, R.: Large-scale secure computation: Multi-party computation for (parallel) RAM programs. In: CRYPTO (2015)
6. Boyle, E., Chung, K., Pass, R.: Oblivious parallel RAM and applications. In: TCC (2016)
7. Boyle, E., Naor, M.: Is there an oblivious RAM lower bound? In: ITCS (2016)
8. Cash, D., Drucker, A., Hoover, A.: A lower bound for one-round oblivious RAM. In: TCC (2020)
9. Cash, D., Küpçü, A., Wichs, D.: Dynamic proofs of retrievability via oblivious RAM. J. Cryptology **30**(1), 22–57 (2017)
10. Chan, T.H., Guo, Y., Lin, W., Shi, E.: Oblivious hashing revisited, and applications to asymptotically efficient ORAM and OPRAM. In: ASIACRYPT (2017)
11. Chan, T.H., Guo, Y., Lin, W., Shi, E.: Cache-oblivious and data-oblivious sorting and applications. In: SODA. pp. 2201–2220 (2018)
12. Damgård, I., Meldgaard, S., Nielsen, J.B.: Perfectly secure oblivious RAM without random oracles. In: TCC (2011)
13. Devadas, S., van Dijk, M., Fletcher, C.W., Ren, L., Shi, E., Wichs, D.: Onion ORAM: A constant bandwidth blowup oblivious RAM. In: TCC (2016)
14. Farhadi, A., Hajiaghayi, M., Larsen, K.G., Shi, E.: Lower bounds for external memory integer sorting via network coding. In: STOC (2019)
15. Fletcher, C.W., Dijk, M.v., Devadas, S.: A secure processor architecture for encrypted computation on untrusted programs. In: Proceedings of the seventh ACM workshop on Scalable trusted computing. pp. 3–8. ACM (2012)
16. Fletcher, C.W., Ren, L., Kwon, A., van Dijk, M., Devadas, S.: Freecursive ORAM: [nearly] free recursion and integrity verification for position-based oblivious RAM. In: ASPLOS (2015)
17. Floyd, R.W.: Permuting Information in Idealized Two-Level Storage. In: Complexity of Computer Computations, pp. 105–109. The IBM Research Symposia Series, Springer US (1972)

18. Fredman, M.L., Saks, M.E.: The cell probe complexity of dynamic data structures. In: STOC. ACM (1989)
19. Gentry, C., Goldman, K.A., Halevi, S., Jutla, C.S., Raykova, M., Wichs, D.: Optimizing ORAM and using it efficiently for secure computation. In: PETS (2013)
20. Gentry, C., Halevi, S., Jutla, C., Raykova, M.: Private database access with HE-over-ORAM architecture. In: International Conference on Applied Cryptography and Network Security. pp. 172–191. Springer (2015)
21. Gentry, C., Halevi, S., Raykova, M., Wichs, D.: Outsourcing private RAM computation. In: FOCS (2014)
22. Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious RAMs. J. ACM (1996)
23. Goodrich, M.T.: Data-oblivious external-memory algorithms for the compaction, selection, and sorting of outsourced data. In: SPAA (2011)
24. Goodrich, M.T., Mitzenmacher, M.: Privacy-preserving access of outsourced data via oblivious RAM simulation. In: ICALP (2011)
25. Gordon, S.D., Katz, J., Kolesnikov, V., Krell, F., Malkin, T., Raykova, M., Vahlis, Y.: Secure two-party computation in sublinear (amortized) time. In: CCS (2012)
26. Hubácek, P., Koucký, M., Král, K., Slívová, V.: Stronger lower bounds for online ORAM. In: TCC (2019)
27. Jacob, R., Larsen, K.G., Nielsen, J.B.: Lower bounds for oblivious data structures. In: SODA (2019)
28. Jafargholi, Z., Larsen, K.G., Simkin, M.: Optimal oblivious priority queues. In: SODA (2021)
29. Komargodski, I., Lin, W.K.: A logarithmic lower bound for oblivious RAM (for all parameters). Cryptology ePrint Archive, Report 2020/1132 (2020)
30. Kushilevitz, E., Lu, S., Ostrovsky, R.: On the (in)security of hash-based oblivious RAM and a new balancing scheme. In: SODA (2012)
31. Larsen, K.G.: The cell probe complexity of dynamic range counting. In: STOC (2012)
32. Larsen, K.G., Malkin, T., Weinstein, O., Yeo, K.: Lower bounds for oblivious near-neighbor search. In: SODA (2020)
33. Larsen, K.G., Nielsen, J.B.: Yes, there is an oblivious RAM lower bound! In: CRYPTO (2018)
34. Larsen, K.G., Simkin, M., Yeo, K.: Lower bounds for multi-server oblivious RAMs. In: TCC (2020)
35. Larsen, K.G., Weinstein, O., Yu, H.: Crossing the logarithmic barrier for dynamic boolean data structure lower bounds. In: 2018 Information Theory and Applications Workshop, ITA. pp. 1–40 (2018)
36. Lin, W., Shi, E., Xie, T.: Can we overcome the n log n barrier for oblivious sorting? In: SODA (2019)
37. Liu, C., Wang, X.S., Nayak, K., Huang, Y., Shi, E.: ObliVM: A programming framework for secure computation. In: IEEE S&P (2015)
38. Lu, S., Ostrovsky, R.: Distributed oblivious RAM for secure two-party computation. In: TCC. pp. 377–396 (2013)
39. Maas, M., Love, E., Stefanov, E., Tiwari, M., Shi, E., Asanovic, K., Kubiatowicz, J., Song, D.: PHANTOM: practical oblivious computation in a secure processor. In: ACM CCS (2013)
40. Ostrovsky, R., Shoup, V.: Private information storage (extended abstract). In: STOC (1997)
41. Patel, S., Persiano, G., Raykova, M., Yeo, K.: PanORAMa: Oblivious RAM with logarithmic overhead. In: FOCS (2018)

42. Patel, S., Persiano, G., Yeo, K.: Lower bounds for encrypted multi-maps and searchable encryption in the leakage cell probe model. In: CRYPTO (2020)
43. Persiano, G., Yeo, K.: Lower bounds for differentially private RAMs. In: EUROCRYPT (2019)
44. Pătraşcu, M., Demaine, E.D.: Logarithmic lower bounds in the cell-probe model. SIAM Journal on Computing **35**(4), 932–963 (2006)
45. Ren, L., Fletcher, C.W., Kwon, A., Stefanov, E., Shi, E., van Dijk, M., Devadas, S.: Constants count: Practical improvements to oblivious RAM. In: USENIX Security (2015)
46. Ren, L., Yu, X., Fletcher, C.W., van Dijk, M., Devadas, S.: Design space exploration and optimization of path oblivious RAM in secure processors. In: ISCA (2013)
47. Shi, E.: Path oblivious heap: Optimal and practical oblivious priority queue. In: S&P (2020)
48. Shi, E., Chan, T.H., Stefanov, E., Li, M.: Oblivious RAM with $O((\log N)^3)$ worst-case cost. In: ASIACRYPT (2011)
49. Shi, E., Chan, T.H., Stefanov, E., Li, M.: Oblivious RAM with o((logn)3) worst-case cost. In: Advances in Cryptology - ASIACRYPT. pp. 197–214 (2011)
50. Stefanov, E., van Dijk, M., Shi, E., Chan, T.H., Fletcher, C.W., Ren, L., Yu, X., Devadas, S.: Path ORAM: an extremely simple oblivious RAM protocol. J. ACM **65**(4), 18:1–18:26 (2018)
51. Stefanov, E., van Dijk, M., Shi, E., Fletcher, C.W., Ren, L., Yu, X., Devadas, S.: Path ORAM: an extremely simple oblivious RAM protocol. In: CCS (2013)
52. Stefanov, E., Shi, E.: Oblivistore: High performance oblivious cloud storage. In: IEEE S&P (2013)
53. Stefanov, E., Shi, E., Song, D.X.: Towards practical oblivious RAM. In: NDSS (2012)
54. Vitter, J.S.: External Memory Algorithms and Data Structures: Dealing with Massive Data. ACM Comput. Surv. **33**(2), 209–271 (Jun 2001)
55. Wang, X., Chan, T.H., Shi, E.: Circuit ORAM: on tightness of the goldreich-ostrovsky lower bound. In: CCS (2015)
56. Wang, X.S., Huang, Y., Chan, T.H., Shelat, A., Shi, E.: SCORAM: oblivious RAM for secure computation. In: ACM CCS. pp. 191–202 (2014)
57. Wang, X.S., Nayak, K., Liu, C., Chan, T.H., Shi, E., Stefanov, E., Huang, Y.: Oblivious data structures. In: CCS (2014)
58. Weiss, M., Wichs, D.: Is there an oblivious RAM lower bound for online reads? J. Cryptol. **34**(3), 18 (2021)
59. Williams, P., Sion, R., Tomescu, A.: Privatefs: A parallel oblivious file system. In: ACM CCS (2012)
60. Yao, A.C.: Should tables be sorted? J. ACM **28**(3), 615–628 (1981)
61. Zahur, S., Wang, X.S., Raykova, M., Gascón, A., Doerner, J., Evans, D., Katz, J.: Revisiting square-root ORAM: efficient random access in multi-party computation. In: IEEE S&P. pp. 218–234 (2016)