Yes, There is an Oblivious RAM Lower Bound!

Kasper Green Larsen[1,2(✉)] and Jesper Buus Nielsen[1,2]

[1] Computer Science, Aarhus University, Aarhus, Denmark
larsen@cs.au.dk
[2] Computer Science and DIGIT, Aarhus University, Aarhus, Denmark

Abstract. An Oblivious RAM (ORAM) introduced by Goldreich and Ostrovsky [JACM'96] is a (possibly randomized) RAM, for which the memory access pattern reveals no information about the operations performed. The main performance metric of an ORAM is the bandwidth overhead, i.e., the multiplicative factor extra memory blocks that must be accessed to hide the operation sequence. In their seminal paper introducing the ORAM, Goldreich and Ostrovsky proved an amortized $\Omega(\lg n)$ bandwidth overhead lower bound for ORAMs with memory size n. Their lower bound is very strong in the sense that it applies to the "offline" setting in which the ORAM knows the entire sequence of operations ahead of time.

However, as pointed out by Boyle and Naor [ITCS'16] in the paper "Is there an oblivious RAM lower bound?", there are two caveats with the lower bound of Goldreich and Ostrovsky: (1) it only applies to "balls in bins" algorithms, i.e., algorithms where the ORAM may only shuffle blocks around and not apply any sophisticated encoding of the data, and (2), it only applies to statistically secure constructions. Boyle and Naor showed that removing the "balls in bins" assumption would result in super linear lower bounds for sorting circuits, a long standing open problem in circuit complexity. As a way to circumventing this barrier, they also proposed a notion of an "online" ORAM, which is an ORAM that remains secure even if the operations arrive in an online manner. They argued that most known ORAM constructions work in the online setting as well.

Our contribution is an $\Omega(\lg n)$ lower bound on the bandwidth overhead of any online ORAM, even if we require only computational security and allow arbitrary representations of data, thus greatly strengthening the lower bound of Goldreich and Ostrovsky in the online setting. Our lower bound applies to ORAMs with memory size n and any word size $r \geq 1$. The bound therefore asymptotically matches the known upper bounds when $r = \Omega(\lg^2 n)$.

K. G. Larsen—Supported by a Villum Young Investigator grant 13163 and an AUFF starting grant.

J. B. Nielsen—Supported by the European Union's Horizon 2020 research and innovation programme under grant agreement #731583 (SODA).

H. Shacham and A. Boldyreva (Eds.): CRYPTO 2018, LNCS 10992, pp. 523–542, 2018.
https://doi.org/10.1007/978-3-319-96881-0_18

1 Introduction

It is often attractive to store data at an untrusted party, and only retrieve the needed parts of it. Encryption can help ensure that the party storing the data has no idea of what it is storing, but still it is possible to get information about the stored data by analyzing the access pattern.

Goldreich and Ostrovsky [GO96] solved this problem in a model with a client that is equipped with a random oracle and small (constant size) memory. The client runs a program while using a (larger) RAM stored on a server, where the access pattern is observed by the adversary. The results from [GO96] shows that any program in the standard RAM model can be transformed using an "oblivious RAM simulator" into a program for the oblivious RAM model, where the access pattern is information theoretically hidden. Whereas it is not reasonable to assume a random oracle in a real implementation, Goldreich and Ostrovsky point out that one can replace it by a pseudorandom function (PRF) that only depends on a short key stored by the client. This way, one obtains a solution that is only computationally secure. The construction in [GO96] had an overhead of polylog(n), where the overhead is defined to be the number of memory blocks communicated per operation and n is defined as the number of memory blocks of the ORAM. The paper at the same time showed a lower bound on the overhead of $\Omega(\lg n)$.

There has been a surge in research on ORAMs in recent years, both on practical efficiency, asymptotic efficiency, practical applications and theoretical applications. There are literally hundreds of papers on the subject and any list will leave out important results. However, a good starting point for getting an overview of the breadth of the research is [PR10, DMN11, GM11, GMOT12, KLO12, WST12, SS13, CLP14, GHL+14, GLO15, BCP16, LO17, Goo17, Goo18], their references and the papers citing them.

A seminal result was the Path ORAM [SvDS+13], which has an amortized $O(\lg n)$ bandwidth cost (measured in blocks communicated) for blocks of size $\Omega(\lg^2 n)$ bits. It was the first to achieve this overhead. Since the lower bound in [GO96] applies to any block size, this seems to have finished the story by giving matching lower and upper bounds. However, as pointed out by Boyle and Naor [BN16], there are two caveats with the lower bound of Goldreich and Ostrovsky: (1) it only applies to "balls in bins" algorithms, i.e., algorithms where the ORAM may only shuffle blocks around and not apply any sophisticated encoding of the data, and (2), it only applies to statistically secure constructions. This leaves open the question whether a non-"balls in bins" ORAM construction or an inherently computationally secure only ORAM construction could beat the $\lg n$ lower bound. In this work we show that this is not the case.

1.1 Our Contributions

Before we state our result we present the class of ORAM schemes our new lower bound applies to.

Online ORAMs. Boyle and Naor showed that proving lower bounds without the "balls in bins" assumption would result in super-linear lower bounds for sorting circuits, a long standing open problem in circuit complexity. As a way to circumventing this barrier, they proposed a notion of an "online" ORAM, which is an ORAM that remains secure even if the operations arrive in an online manner. They argued that most known ORAM constructions work in the online setting as well. Also, most applications of ORAM schemes require that the scheme is online.

Passive ORAMs. It is implicit in the original definition of ORAMs that the server is passive storage. There are also ORAM constructions (for instance, Onion ORAM [DvDF+16] and the recent proposal in [AFN+16]), which allow the server to perform untrusted computation on behalf of the client. Our lower bound does not apply to such ORAMs. And indeed most of these schemes achieves sub-logarithmic overhead.

Problem Statement. To be a bit more precise, the purpose of the online ORAM is to allow a client to store data on an untrusted server. The online ORAM provides security in the sense that it hides the data access pattern from the server (which blocks are read/written). More formally, an online ORAM supports the following two operations:

- write(a, data): Store data in the block of address a, where a $\in [n]$ and data $\in \{0,1\}^r$.
- read(a): Return the contents of the block of address a.

During operation, the ORAM maybe perform the same type of operations on a memory stored on a server. The server memory space may be larger than that of the ORAM and the block size need not be the same. To distinguish the two, we refer to blocks at the server as *memory cells*, we use w to denote the number of bits in a memory cell and we use r to denote the number of bits in a block, i.e., r is the number of bits in the data arguments of the write(a, data) operations.

An online ORAM is secure in the following sense: Let y be a sequence of operations for the ORAM:

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

where each op_i is either a write(a, data) or read(a) operation. Let

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

denote the memory access pattern to the server from the online ORAM, i.e., each $A(\mathrm{op}_j)$ is the list of addresses of the cells accessed at the server while processing op_j. For a randomized construction $A(y)$ is a random variable. Security is defined as follows: for two distinct sequences of operations y and z with the same number of operations, $A(y)$ and $A(z)$ are computationally indistinguishable. In the main text we will give a more formal definition. The present informal definition is only to be able to present our result.

We prove the following theorem.

Theorem 1 (informal). *Any online ORAM with n blocks of memory, consisting of $r \geq 1$ bits each, must have an expected amortized bandwidth overhead of $\Omega(\lg(nr/m))$ on sequences of $\Theta(n)$ operations. Here m denotes the client memory in bits. This holds in the random oracle model, requiring only computational indistinguishability, holds for any server cell size w and allows for arbitrary representations of the data in memory. For the natural setting of parameters $r \leq m \leq n^{1-\varepsilon}$ for an arbitrarily small constant $\varepsilon > 0$, the lower bound simplifies to $\Omega(\lg n)$.*

Discussion 1. Comparing our definition of an online ORAM to that of Boyle and Naor [BN16], our security definition is slightly stricter in the sense that for us, $A(y)$ also lets the adversary see which block accesses belong to which operations. Boyle and Naor simply define $A(y)$ as $A(\mathrm{op}_1) \cdots A(\mathrm{op}_M)$ (without the comma separation). We believe our stricter definition is justifiable as it seems questionable to base security on not knowing when one operation has finished processing, at least in an online setting where operations arrive one at a time and we don't know before hand how many operations we have to process. To the best of our knowledge, all online ORAM implementations also satisfy our stricter security definition.

Discussion 2. Most ORAM constructions have $w = \Theta(r)$, i.e., the server memory cells have the same asymptotic number of bits as the block size of the ORAM. However, this is not a strict requirement, and the Path ORAM [SvDS+13] in fact has $r = \Theta(\lg^2 n)$ and $w = \Theta(\lg n)$ in order to achieve their $O(\lg n)$ amortized bandwidth overhead, i.e., the ORAM block size and the server memory cells have very different sizes. When dealing with w and r that are not asymptotically the same, one defines the bandwidth overhead as the multiplicative factor extra bits that must be accessed from the server compared to just reading the r bits comprising a block. Thus if an ORAM accesses t memory cells per operation, its bandwidth overhead is tw/r. The Path ORAM accesses an amortized $\Theta(\lg^2 n)$ memory cells per operation. This is $\Theta(w \lg^2 n) = \Theta(\lg^3 n)$ bits, which is a multiplicative factor $\Theta((\lg^3 n)/r) = \Theta(\lg n)$ overhead, i.e., its bandwidth overhead is $\Theta(\lg n)$. Our lower bound holds regardless of the memory cell size w.

1.2 Proof Strategy

In the following, we give a brief overview of the ideas in our lower bound proof. Our first observation is that the definition of the online ORAM coincides with the definition of an oblivious data structure, as defined in [WNL+14], solving the following array maintenance problem:

Definition 1. *In the array maintenance problem, we must maintain an array B of n r-bit entries under the following two operations:*

- *write$(a, data)$: Set the contents of $B[a]$ to data, where $a \in [n]$ and data $\in \{0,1\}^r$.*
- *read(a): Return the contents of $B[a]$.*

This data structure view allows us to re-use techniques for proving data structure lower bounds. More concretely, we prove a lower bound for oblivious data structures solving the array maintenance problem and then use the argument above to conclude the same lower bound for online ORAMs.

Data structure lower bounds are typically proved in the cell probe model of Yao [Yao81]. Intuitively, this model is the same as the standard RAM, except that computation is free of charge and we only pay for memory accesses. This matches the ORAM performance metrics perfectly as we care about the bandwidth overhead and the memory accesses revealing no information. We thus tweak the definition of the cell probe model such that it captures client memory and other technical details of the online ORAM not normally found in data structures. We will define our model, which we term the *oblivious cell probe model*, formally in Sect. 2. Another advantage of this data structure view is that it accurately captures the *online* setting of online ORAMs and thus allow us to circumvent the circuit complexity barrier demonstrated by Boyle and Naor [BN16].

The strongest current techniques for proving lower bounds in the cell probe model, can prove lower bounds of the form $\tilde{\Omega}(\lg^2 n)$ [Lar12] for problems with a $\lg n$-bit output, and $\tilde{\Omega}(\lg^{1.5} n)$ for decision problems [LWY18], i.e., one-bit answers to queries. Here $\tilde{\Omega}$ hides polyloglog factors. We did not manage to use these techniques to prove lower bounds for ORAMs, but instead took inspiration from the so-called *information transfer* method of Pătraşcu and Demaine [PD06], which can prove lower bounds of $\Omega(\lg n)$. It would be quite exciting if the techniques in [Lar12,LWY18] could be tweaked to prove $\omega(\lg n)$ lower bounds for e.g. the *worst case* bandwidth overhead of ORAMs. We leave this as intriguing future work.

The basic idea in the *information transfer* method, is to consider a distribution over sequences of M operations on a data structure. One then considers a binary tree $\mathcal{T}$ on top of such a random sequence, having an operation in each leaf. The next step is to consider the memory accesses arising from processing the M operations. Each such memory access is assigned to a node $v \in \mathcal{T}$ as follows: For a memory access p to a memory cell c, let ℓ_i be the leaf of $\mathcal{T}$ containing the operation that caused the memory access p. Let ℓ_j, with $j < i$, be the leaf corresponding to the last time c was accessed prior to p. We associate p with the lowest common ancestor of ℓ_i and ℓ_j. The next step is to prove that for every node $v \in \mathcal{T}$, there has to be many memory accesses assigned to v. Since each memory access is assigned to only one node in $\mathcal{T}$, we can sum up the number of memory accesses assigned to all the nodes of $\mathcal{T}$ and get a lower bound on the total number of accesses.

Now to lower bound the number of memory accesses assigned to a node v, observe that such memory accesses correspond precisely to operations in the right subtree of v accessing memory cells last accessed during the operations in the left subtree. To prove that there must be many such memory accesses, one proves that the answers to the queries (read operations) in the right subtree depends heavily on the updates (write operations) in the left subtree. In this way, one basically shows that every leaf must make a memory access for every ancestor in $\mathcal{T}$, resulting in an $\Omega(\lg M)$ lower bound.

The problem for us, is that the array maintenance problem is trivial for standard data structures. Thus the above approach fails utterly without more ideas. The issue is that for any distribution over read and write operations, we cannot prove that a read operation in some leaf of $\mathcal{T}$ has to make memory accesses for every ancestor in $\mathcal{T}$. Basically, for most nodes, the read operations in the right subtree will request array entries not written to in the left subtree and thus will not need to access anything written there. Our key idea for exploiting the security requirement is that, if we *change* the distribution over operations, then the number of memory accesses assigned to the nodes of $\mathcal{T}$ cannot change drastically as this would be observable by an adversary who can simply construct $\mathcal{T}$ and assign the memory accesses. We can therefore examine the nodes v of $\mathcal{T}$, and for each one, change the distribution over operations such that the read operations in the right subtree requests precisely the array entries written to in the left subtree. By an entropy argument, there has to be many memory accesses under such a distribution. And by our security requirement, this translates back to many memory accesses under the original distribution. We refer the reader to Sect. 3 for the full details.

2 Oblivious Cell Probe Model

In this section, we formally define a lower bound model for proving lower bounds for oblivious data structures. As mentioned earlier, an ORAM immediately gives an oblivious data structure for array maintenance. Hence we set out to prove lower bounds for such data structures.

Our new model is an extension of the cell probe model of Yao [Yao81]. The cell probe model is traditionally used to prove lower bounds for word-RAM data structures and is extremely powerful in the sense that it allows arbitrary computations and only charge for memory accesses. We augment the cell probe model to capture the client side memory of an ORAM. To make clear the distinction between our lower bound model and traditional upper bound models, we call ours the *oblivious cell probe model*. The reason why we introduce this model, is that it allows for a clean proof and definition, plus it brings in all the techniques developed for proving data structure lower bounds. Moreover, we hope that our work inspires other lower bound proofs for oblivious data structures, and thus our thorough definition may serve as a reference.

Problems. A data structure problem in the oblivious cell probe model is defined by a universe $\mathcal{U}$ of update operations, a universe $\mathcal{Q}$ of queries and an output domain $\mathcal{O}$. Furthermore, there is a query function $f : \mathcal{U}^* \times \mathcal{Q} \to \mathcal{O}$. 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 $f(u_1 \dots u_M, q)$.

As an example, consider the array maintenance problem (Definition 1). Here $\mathcal{U}$ is the set of all write(a, data) operations, $\mathcal{Q}$ is the set of all read(a) operations and $\mathcal{O}$ is $\{0,1\}^r$.

Oblivious Cell Probe Data Structure. An oblivious cell probe data structure with client memory m bits for a problem $\mathcal{P} = (\mathcal{U}, \mathcal{Q}, \mathcal{O}, f)$ consists of a random access memory of w-bit cells, a client memory of m bits and a random bit string R of some finite length ℓ. We make no restrictions on ℓ, only that it is finite. Note in particular that R can be exponentially long and hence contain all the randomness needed by the oblivious RAM and/or a random oracle. We will call R the *random-oracle bit-string*. Each cell of the memory has an integer address amongst $[K]$ and we typically assume $w \geq \max\{\lg K, \lg M\}$ such that any cell can store the address of any other cell and the index of any operation performed on it.

When processing an operation, an oblivious cell probe data structure may read or write memory cells. The cell to read or write in each step may depend arbitrarily on the client memory contents and all contents of cells read so far while processing the operation. Moreover, after each read or write, the oblivious cell probe data structure may change the contents of the client memory. The performance measure is defined solely as the number of memory cells read/written to while processing an operation, i.e., computation is free of charge. To capture this formally, an oblivious cell probe data structure is defined by a decision tree T_{op} for every operation $\text{op} \in \mathcal{U} \cup \mathcal{Q}$, i.e., it has one decision tree for every possible operation in the data structure problem. The tree is meant to capture in a crisp way the operation of the oblivious data structure. Each node represents a "step" of computation which might depend on the local client memory of the oblivious data structure, the randomness R and all server memory positions read so far while processing the operation. It may also read a memory position on the server or write a memory position on the server. The "implementation" of the operation can therefore depend on previous operations to the extent that information about them is stored in the local memory.

More formally, each decision tree T_{op} is a rooted finite tree. Each node v of T_{op} is labelled with an address $i \in [K]$ and it has one child for every triple of the form (m_0, c_0, r) where $m_0 \in \{0,1\}^m$, $c_0 \in \{0,1\}^w$ and $r \in \{0,1\}^\ell$. Each edge to a child is furthermore labelled with a triple (j, m_1, c_1) with $j \in [K]$, $m_1 \in \{0,1\}^m$ and $c_1 \in \{0,1\}^w$. To process an operation op, the oblivious cell probe data structure starts its execution at the root of the corresponding tree T_{op} and traverses a root to leaf path in T_{op}. When visiting a node v in this traversal, labelled with some address $i_v \in [K]$ it *probes* the memory cell of address i_v. If C denotes its contents, M denotes the current contents of the client memory and R denotes the random-oracle bit-string, the process continues by descending to the child of v corresponding to the tuple (M, C, R). 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*. We make no requirements that $m_1 \neq M$, $c_1 \neq C$ or $j \neq i$. The execution stops when reaching a leaf of T_{op}.

Finally, each leaf v of a tree T_{op}, where op is in $\mathcal{Q}$, is labelled with a w-bit string L_v (the answer to the query). We say that the oblivious cell probe data structure returns L_v as its answer to the query op.

Definition 2 (Expected Amortized Running Time). *We say that an oblivious cell probe data structure has expected amortized running time $t(M)$ 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 a uniformly random random-oracle string $r \in \{0,1\}^\ell$. We say that an oblivious cell probe data structure has expected amortized running time $t(M)$ if it has expected amortized running time $t(M)$ on all sequences y of operations from $\mathcal{U} \cup \mathcal{Q}$.*

We proceed to define security. Let

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

denote a sequence of M operations to the data structure problem, where each $\mathrm{op}_i \in \mathcal{U} \cup \mathcal{Q}$. For an oblivious cell probe data structure, define the (possibly randomized) *probe sequence* on y as the tuple:

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

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

Definition 3 (Correctness). *We say that an oblivious cell probe data structure has failure probability δ if, for every sequence and any operation op in the sequence, the data structure answers op correctly with probability at least $1 - \delta$.*

Definition 4 (Security). *An oblivious cell probe data structure is said to be secure if the following two properties hold:*

Indistinguishability: *For any two data request sequences y and z of the same length M, their probe sequences $A(y)$ and $A(z)$ cannot be distinguished with probability better than $\frac{1}{4}$ by an algorithm which is polynomial time in $M + \lg|\mathcal{U}| + \lg|\mathcal{Q}| + w$.*

Correctness: *The oblivious cell probe data structure has failure probability at most $1/3$.*

Discussion 1. It is clear that for most uses of an ORAM, having indistinguishability of $1/4$ and failure probability $1/3$ is not satisfactory. However, for the sake of a lower bound, allowing these large constant slack parameters just gives a stronger bound. In particular, when $M, \lg|\mathcal{U}|, \lg|\mathcal{Q}|, w \in \mathrm{poly}(k)$ for a security parameter k, then our bound applies to computational indistinguishability by an adversary running in time $\mathrm{poly}(k)$.

Discussion 2. Since the random-oracle bit-string and the decision trees are finite, the model does not capture algorithms which might potentially run for arbitrary many steps with vanishing probability. However, any such algorithm might at

the price of an error probability on the output be pruned to a finite decision tree consuming only a finite amount of randomness. By pruning at a sufficiently high depth, an arbitrarily small error probability in $O(2^{-n})$ may be introduced. Since we allow a large constant error probability of $1/3$ our lower bound also applies to algorithms which might potentially run for arbitrary many step with vanishing probability on sequences of length $\text{poly}(n)$.

Discussion 3. For online ORAMs, we are typically interested in the *bandwidth overhead*, which is the multiplicative factor extra bits that must be accessed compared to the underlying RAM being simulated. If the underlying RAM/array has r-bit entries, we have that a sequence of M operations can be processed by accessing Mr bits. Thus for ORAMs with (server) cell size w bits, this translates into the minimum number of probes being Mr/w. Thus if an oblivious data structure for the array maintenance problem has expected amortized running time $t(M)$, then the corresponding ORAM has an expected amortized bandwidth overhead of $t(M)w/r$.

3 Lower Bound

In this section, we prove our lower bound for oblivious cell probe data structures solving the array maintenance problem and thus indirectly also prove a lower bound for online ORAMs. The model is recapped in Fig. 1. The formal statement of our result is as follows:

Theorem 2. *Let $\mathcal{D}$ be an oblivious cell probe data structure for the array maintenance problem on arrays of n r-bit entries where $r \geq 1$. Let w denote the cell size of $\mathcal{D}$, let m denote the number of bits of client memory. If $\mathcal{D}$ is secure according to Definition 4, then there exists a sequence y of $\Theta(n)$ operations such that the expected amortized running time of $\mathcal{D}$ on y is $\Omega(\lg(nr/m)r/w)$. In terms of bandwidth overhead, this means that the expected amortized bandwidth overhead is $\Omega(\lg(nr/m))$. For the most natural setting of $r \leq m \leq n^{1-\varepsilon}$, this simplifies to $\Omega(\lg n)$.*

Let $\mathcal{D}$ be as in Theorem 2 and let $[K] \subseteq [2^w]$ be the set of possible addresses of its memory cells. Throughout our lower bound proof, we assume that $\mathcal{D}$ has failure probability at most $1/32$ instead of $1/3$. Note that the lower bound extends to failure probability $1/3$ (or any failure probability bounded away from $1/2$) simply because one can always run a constant number of independent copies in parallel and use a majority vote when answering a read operation.

We prove our lower bound for processing the following fixed sequence of $M = 2n$ operations:

– We perform a sequence y of intermixed read and write operations. The sequence has n of each type and looks as follows:

$$y := \text{write}(0, \bar{0}), \text{read}(0), \text{write}(0, \bar{0}), \text{read}(0), \ldots, \text{write}(0, \bar{0}), \text{read}(0)$$

where $\bar{0}$ denotes the all-zeroes bit string of length r.

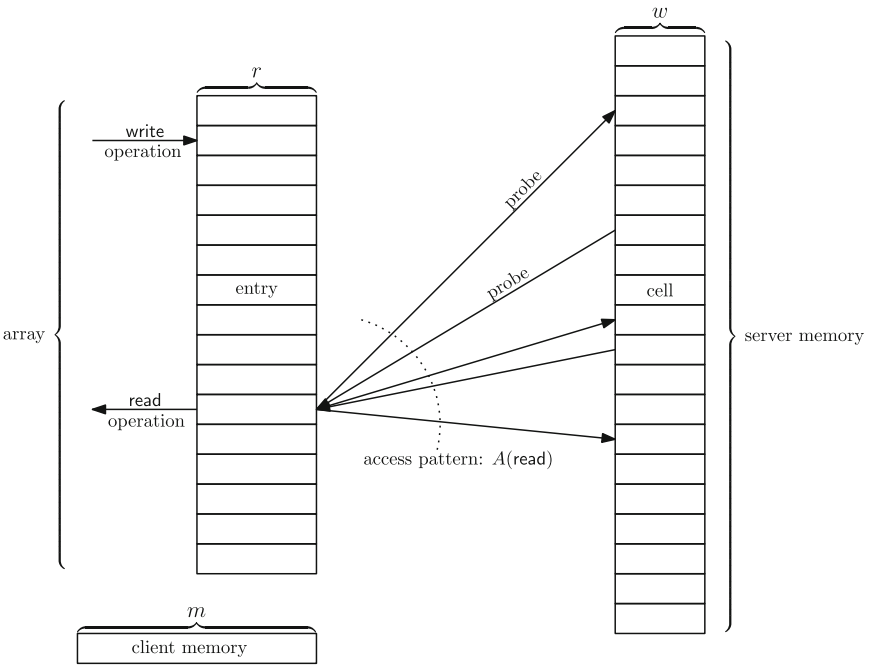

Fig. 1. An ORAM implements an array of r-bit entries. Each operation can be a read or a write. Each operation op makes read and write probes to the server memory. The sequence of probes made during an operation op is called the access pattern of op and is written as $A(\mathsf{op})$. Words in the server memory are called cells. Each cell is w bits. The ORAM is restricted to m bits of storage between two operations. During an operation it can use unlimited storage.

The sequence y is just the sequence of alternating write and read operations that all access the first array entry, and the write operations just store $\bar{0}$ in it. What makes this sequence costly is of course that the probe sequence of $\mathcal{D}$ on y must be computationally indistinguishable from all other sequences of $M = 2n$ operations.

To exploit this, define $A(y)$ as the random variable giving the probe sequence (as defined in Sect. 2) of $\mathcal{D}$ when processing y. Since our sequence has $M = 2n$ operations on an array with r-bit entries, we have a minimum bandwidth usage of $2nr$ bits. The data structure $\mathcal{D}$ has a cell size of w bits, and thus the minimum number of probes is $2nr/w$. Thus by definition, we have that the expected amortized bandwidth overhead is $\mathbb{E}[|A(y)|]w/(2nr)$. Our goal is thus to lower bound $\mathbb{E}[|A(y)|]$. Our proof is an adaptation of the *information transfer* technique by Pătraşcu and Demaine [PD06] for proving data structure lower bounds in the cell probe model. The basic proof strategy is as follows:

For any sequence of $M = 2n$ operations z of the form:

$$\mathrm{write}(i_1, d_1), \mathrm{read}(i_2), \mathrm{write}(i_3, d_3), \mathrm{read}(i_4), \ldots, \mathrm{write}(i_{2n-1}, d_{2n-1}), \mathrm{read}(i_{2n}),$$

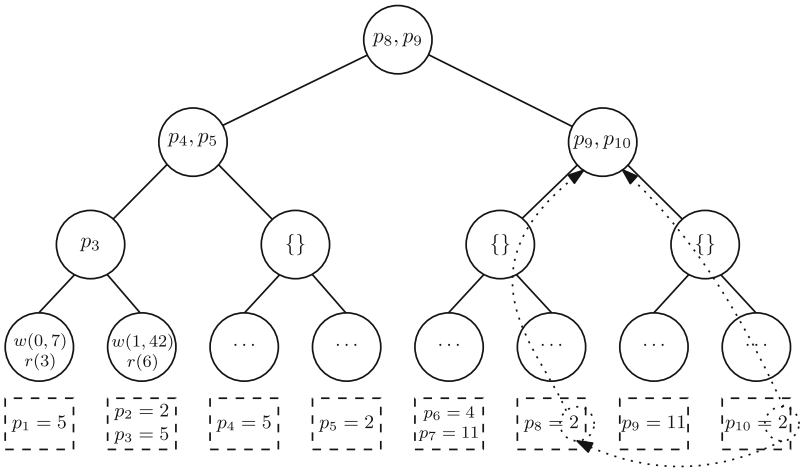

Fig. 2. Illustration of how probes are associated to nodes. The second to bottom layer is a sequence of 16 intermixed read and write operations $\text{write}(0,7), \text{read}(3), \text{write}(1,42), \text{read}(6), \ldots$. We only show the two first pairs. Under each pair of commands we show for illustration the probes that they made. In the nodes we illustrate where the probes would be associated. As an example, take leaf number 8 (the right most leaf). It did the 10'th probe. That probe probed cell 2. That happened last time in leaf number 6 by probe p_8. The lowest common ancestor of leafs 6 and 8 therefore contains p_{10}.

we consider a binary tree $\mathcal{T}(z)$ with n leaves, on top of the $2n$ operation. There is one leaf in $\mathcal{T}(z)$ for each consecutive pair $\text{write}(i_j, d_j), \text{read}(i_{j+1})$. Let op_i denote the i'th operation in the sequence z, i.e., op_1 is the first $\text{write}(i_1, d_1)$ operation, op_2 is the first $\text{read}(i_2)$ operation and so on. Consider the probe sequence $A(z) = (A(\text{op}_1), A(\text{op}_2), \ldots, A(\text{op}_{2n}))$ where each $A(\text{op}_i)$ is the sequence of memory addresses probed when $\mathcal{D}$ processes $A(\text{op}_i)$ during the sequence of operations z. Let $p_1, \ldots, p_T$ denote the concatenation of all the sequences of probed memory addresses, i.e., $p_1, \ldots, p_T = A(\text{op}_1) \circ A(\text{op}_2) \circ \cdots \circ A(\text{op}_{2n})$ where $\circ$ is concatenation.

We assign each probed address p_i to a node of $\mathcal{T}(z)$. If $p_i = s$ for some address $s \in [K] \subseteq [2^w]$, then let p_j with $j < i$ denote the last time cell s was probed prior to p_i. Let ℓ_i and ℓ_j denote the leaves of $\mathcal{T}(z)$ containing the two operations whose processing caused the probes p_i and p_j, i.e., if p_i is a probe resulting from op_a, then ℓ_i is the leaf containing op_a. We assign p_i to the lowest common ancestor of ℓ_i and ℓ_j. If p_j does not exist, i.e., the memory cell with address s was not probed before, then we do not assign p_i to any node of $\mathcal{T}(z)$. See Fig. 2 for an illustration.

Our goal is to show that for the sequence y, it must be the case that most nodes of $\mathcal{T}(y)$ have a large number of probes assigned to them (in expectation). Since a probe is assigned to only one node of the tree, we can sum up the number of probes assigned to all nodes of $\mathcal{T}(y)$ to get a lower bound on the total number of probes in $A(y)$. In more detail, consider a node $v \in \mathcal{T}(y)$. Our proof will show

that the read instructions in the right subtree of v have to probe many cells that were last probed during the operations in the left subtree. This corresponds precisely to the set of probes assigned to v being large.

To gain intuition for why the read operations in the right subtree must make many probes to cells written in the left subtree, observe that from the indistinguishability requirement, the probe sequence must look identical regardless of whether $\mathcal{D}$ is processing y, or if we instead process a random sequence in which the write operations in the left subtree are $\text{write}(1, d_1), \text{write}(2, d_2), \ldots$ and the reads in the right subtree are $\text{read}(1), \text{read}(2), \ldots$, where each d_i is a uniformly random r-bit string. In the latter case, the read operations have to recover all the (random) bits written in the left subtree and thus must probe many cells written in the left subtree by an entropy argument. Since counting the probes assigned to a node v can be done in poly-time in $M + r$ without knowing the arguments to the instructions, it follows from the indistinguishability property that for $A(y)$, there also has to be a large number of probes assigned to the node v. The argument is fleshed out in a bit more detail in Fig. 3.

We proceed to give a formal proof.

Nodes with Large Information Transfer. In the following, we argue that for many nodes in $\mathcal{T}(y)$, there must be a large number of probes assigned to v in expectation. We can then sum this up over all nodes in $\mathcal{T}(y)$. We do this as follows: For a sequence of operations z of the form

$$\text{write}(i_1, d_1), \text{read}(i_2), \text{write}(i_3, d_3), \text{read}(i_4), \ldots, \text{write}(i_{2n-1}, d_{2n-1}), \text{read}(i_{2n}),$$

and for every internal node v in $\mathcal{T}(z)$, let $P_v(z)$ denote the set of probes assigned to v. Using the terminology by Pătraşcu and Demaine [PD06], we refer to the set $P_v(z)$ as the *information transfer* in node v. We thus want to lower bound the size of the information transfer in the nodes of $\mathcal{T}(y)$. To do so, define $\text{depth}(v)$ to be the distance from the root of $\mathcal{T}(y)$ to v. We prove the following:

Lemma 1. *If $\mathcal{D}$ has failure probability at most $1/32$, then for every internal node $v \in \mathcal{T}(y)$ with depth $d \in \{5, \ldots, (1/2)\lg(nr/m)\}$, it holds that*

$$\mathbb{E}\left[|P_v(y)|\right] = \Omega(nr/(w2^d)).$$

Let us first briefly give an explanation why the lemma only is proven for $d \in \{5, \ldots, (1/2)\lg(nr/m)\}$. Consider a node at depth d. It has about $nr/2^d$ nodes below it. So a node at depth $d = \lg(nr/m)$ has about m nodes below it. Recall that m is the size of the client memory between two operations. By going only to depth $(1/2)\lg(nr/m)$ we ensure that the node has to transfer much more than m bits such that the client memory plays essentially no role in the information transfer. Starting only from $d = 5$ makes some steps in the proof simpler.

Before proving Lemma 1, we show that it implies our main result. Since every probe in $A(y)$ is assigned to at most one node in $\mathcal{T}(y)$, and using linearity of expectation together with the fact that there are 2^d nodes of depth d in $\mathcal{T}(y)$, we have:

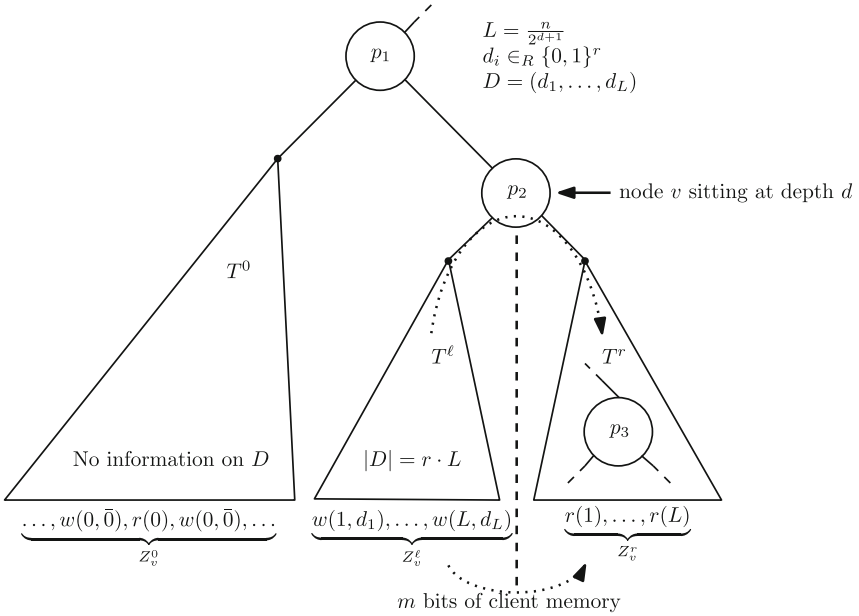

Fig. 3. We sketch the intuition of the proof. Assume for simplicity that the ORAM has perfect correctness, is deterministic and has no client memory. Consider a node v sitting at depth d. It has two sub-tree T^ℓ and T^r. Each of them has $L = \frac{n}{2^{d+1}}$ write and read operations in their leafs. Consider the sequence of operations which in each write in T^ℓ writes a fresh uniform value $d_i \in \{0,1\}^r$ and which in T^r reads these values back. In the sub-tree T^0 preceding T^ℓ the sequence will just write and read zero-values. There is an information transfer of $r \cdot L$ bits from the write operations in T^ℓ to the read operations in T^r, as the string $D = (d_1, \ldots, d_L)$ is being retrieved by the ORAM while in T^r. We argue that this information goes through v in the sense that there must be Lr/w probes assigned to v (illustrated using the dotted arrow in the figure). Namely, there must in T^r clearly be Lr/w probes that give information on D as each probe gives at most w bits of information. A probe in T^r gets assigned either to a node in T^r, to the node v or to an ancestor of v. Consider first a probe p_1 which gets assigned to an ancestor of v. It reads a cell which was last written before D was stored. Hence it cannot give information on D. Consider a probe p_3 assigned to a node in T^r. This node was last written by an operation in T^r. Hence it cannot give new information which was not previously learned by a probe in T^r. Therefore all probes giving information on D are assigned to v, so there are Lr/w probes assigned to v. If the scheme is only correct on a fraction $1-\delta$ of the reads, proportionally less information is transferred, so only $(1-\delta)Lr/w$ probes are needed. Also, m bits of information could be transferred via the client memory when moving from T^ℓ to T^r, reducing the needed number of probes to $\delta Lr/w - m/w$. If d is small enough, this will still be $\Omega(Lr/w)$.

$$\begin{aligned}
\mathbb{E}[|A(y)|] &\geq \sum_{v \in \mathcal{T}(y)} \mathbb{E}[|P_v(y)|] \\
&\geq \sum_{d=5}^{(1/2)\lg(nr/m)} \sum_{v \in \mathcal{T}(y):\mathrm{depth}(v)=d} \mathbb{E}[|P_v|] \\
&= \Omega\left(\sum_{d=5}^{(1/2)\lg(nr/m)} \sum_{v \in \mathcal{T}(y):\mathrm{depth}(v)=d} nr/(w2^d) \right) \\
&= \Omega\left(\sum_{d=5}^{(1/2)\lg(nr/m)} 2^d \cdot nr/(w2^d) \right) \\
&= \Omega\left(nr \lg(nr/m)/w \right).
\end{aligned}$$

Thus the expected amortized bandwidth overhead is

$$\mathbb{E}[|A(y)|]w/(2nr) = \Omega(\lg(nr/m))$$

as claimed. What remains is to prove Lemma 1.

Lower Bounding the Probes. Consider a node v in $\mathcal{T}(y)$ whose depth is $d = \mathrm{depth}(v) \in \{5, \ldots, (1/2)\lg(nr/m)\}$. To prove Lemma 1, we consider a distribution over sequences of $M = 2n$ operations of the form

$$\mathrm{write}(i_1, d_1), \mathrm{read}(i_2), \mathrm{write}(i_3, d_3), \mathrm{read}(i_4), \ldots, \mathrm{write}(i_{2n-1}, d_{2n-1}), \mathrm{read}(i_{2n}).$$

Our distribution will be chosen such that if we draw a sequence Z_v from the distribution and run $\mathcal{D}$ on Z_v, then the read operations in v's right subtree must probe many cells last written during the operations in v's left subtree. This means that $P_v(Z_v)$ must be large. We will then use the computational indistinguishability to argue that $P_v(y)$ must be just as large.

In greater detail, let Z_v be the random variable giving a sequence of $2n$ operations chosen as follows: For every $\mathrm{read}(i_j)$ not in the subtree rooted at v's right child, we simply have $i_j = 0$. For every $\mathrm{write}(i_j, d_j)$ not in the subtree rooted at v's left child, we have $i_j = 0$ and $d_j = \bar{0}$. For the $n/2^{d+1}$ read operations in v's right subtree $\mathrm{read}(i_j), \ldots, \mathrm{read}(i_{j+n/2^{d+1}-1})$, we have $i_j = 1, i_{j+1} = 2, \ldots, i_{j+n/2^{d+1}-1} = n/2^{d+1}$, i.e., the read operations in v's right subtree simply read the array entries $1, 2, 3, \ldots, n/2^{d+1}$ in that order. Finally for the $n/2^{d+1}$ write operations in v's left subtree $\mathrm{write}(i_j, d_j), \ldots, \mathrm{write}(i_{j+n/2^{d+1}}, d_{j+n/2^{d+1}-1})$ we have $i_j = 1, j_{j+1} = 2, \ldots, i_{j+n/2^{d+1}-1} = n/2^{d+1}$ and all d_j are independent and uniformly random w-bit strings. Thus for the random sequence Z_v, the read operations in v's right subtree precisely read the $n/2^{d+1}$ array entries that were filled with random bits during the $n/2^{d+1}$ write operations in v's left subtree.

All other operations just read and write array entry 0 as in the fixed sequence y. We prove the following via an entropy argument:

Lemma 2. *If $\mathcal{D}$ has failure probability at most $1/32$, then there exists a universal constant $C > 0$ such that*

$$\Pr[|P_v(Z_v)| \geq Cnr/(w2^d)] \geq 1/2.$$

Before proving Lemma 2, we show that it implies Lemma 1. For this, observe that by averaging, Lemma 2 implies that there must exist a sequence z in the support of Z_v such that

$$\Pr[|P_v(z)| \geq Cnr/(w2^d)] \geq 1/2.$$

From our security definition, $A(y)$ and $A(z)$ must be computationally indistinguishable. We argue that this implies that $\mathbb{E}[|P_v(y)|] \geq (C/4)nr/(w2^d) = \Omega(nr/(w2^d))$. To see this, assume for the sake of contradiction that $\mathbb{E}[|P_v(y)|] < (C/4)nr/(w2^d)$. By Markov's inequality, we get $\Pr[|P_v(y)| \geq Cnr/(w2^d)] \leq 1/4$. An adversary can now distinguish z and y as follows: Given a sequence $a \in \{y, z\}$, run $\mathcal{D}$ on the sequence a to obtain $A(a)$. Construct from $A(a)$ the tree $\mathcal{T}(a)$ and the set $P_v(a)$ (an adversary knows precisely which probes belong to which operations and can thus construct all the sets $P_v(a)$ for all nodes v in $\mathcal{T}(a)$ in polynomial time in M and w). Output 1 if $|P_v(a)| \geq Cnr/(w2^d)$ and 0 otherwise. This distinguishes y and z with probability at least $1/4$. Thus all that remains is to prove Lemma 2.

Encoding Argument. To prove Lemma 2, we assume for the sake of contradiction that the lemma is false, i.e., $\mathcal{D}$ has failure probability at most $1/32$ but:

$$\Pr[|P_v(Z_v)| \geq (1/100)nr/(w2^d)] < 1/2.$$

We will use this $\mathcal{D}$ to give an impossible encoding of the (random) data

$$d_j, d_{j+1}, \ldots, d_{j+n/2^{d+1}-1}$$

written in the left subtree of v in the sequence Z_v. Let $H(\cdot)$ denote binary Shannon entropy and observe that:

$$H(d_j, d_{j+1}, \ldots, d_{j+n/2^{d+1}-1} \mid R) = nr/2^{d+1} \ ,$$

where R denotes the random bits of the random oracle (these are independent of the input distribution). This is because the variables

$$d_j, d_{j+1}, \ldots, d_{j+n/2^{d+1}-1}$$

are uniformly random and independent r-bit strings. From Shannon's source coding theorem, any (possibly randomized) encoding of

$$d_j, d_{j+1}, \ldots, d_{j+n/2^{d+1}-1},$$

conditioned on R, must use $nr/2^{d+1}$ bits in expectation. This also holds for encoding and decoding algorithms which are not computationally efficient. Our encoding and decoding procedures are as follows:

Encoding. The encoder Alice is given

$$d_j, d_{j+1}, \ldots, d_{j+n/2^{d+1}-1}, R \ .$$

Alice does as follows:

1. From $d_j, d_{j+1}, \ldots, d_{j+n/2^{d+1}-1}$ she constructs the sequence Z_v. She then runs $\mathcal{D}$ on Z_v using the randomness R. While running the sequence Z_v on $\mathcal{D}$, she collects the set F of read operations read(i_j) in v's right subtree which fail to report the correct value d_j written by the write operation write(i_j, d_j) in v's left subtree. If either $|P_v(Z_v)| \geq (1/100)nr/(w2^d)$ or $|F| \geq (1/8)n/2^{d+1}$, then she writes down a 0-bit, followed by $nr/2^{d+1}$ bits giving a straight-forward encoding of $d_j, d_{j+1}, \ldots, d_{j+n/2^{d+1}-1}$. Otherwise, she writes down a 1-bit and proceeds to the next step.
2. She now writes down the contents and addresses of all memory cells whose address is in $P_v(Z_v)$. Let Z_v^ℓ denote operations in v's left subtree and let Z_v^0 denote the prefix of Z_v containing all operations up to just before Z_v^ℓ. The contents she writes down is the contents as they were just after processing the prefix $Z_v^0 \circ Z_v^\ell$. She also writes down the m client memory bits as they were immediately after processing $Z_v^0 \circ Z_v^\ell$. Finally, she also writes down $|F|$ using $\lg n$ bits as well as $\lg \binom{n/2^{d+1}}{|F|}$ bits specifying which read operations in v's right subtree that fail together with $|F|r$ bits specifying the correct answers to the failing read operations. The first part costs $|P_v(Z_v)|(\lg K + w) \leq |P_v(Z_v)|2w \leq (1/25)nr/2^{d+1}$ where $[K]$ is the address space of memory cells. Writing down the client memory costs m bits and writing down the failing read's and their correct answers costs at most

$$\begin{aligned}
\lg n + |F|r + \lg \binom{n/2^{d+1}}{|F|} &\leq \lg n + (1/8)nr/2^{d+1} + \lg \binom{n/2^{d+1}}{(1/8)n/2^{d+1}} \\
&\leq \lg n + (1/8)nr/2^{d+1} \cdot (1 + \lg(8e)/r) \\
&\leq \lg n + (3/4)nr/2^{d+1} \\
&\leq (4/5)nr/2^{d+1}.
\end{aligned}$$

   Thus Alice's message has length at most $m + (21/25)nr/2^{d+1}$ if we she reaches step 2.

Decoding. The decoder Bob is given the message from Alice as well as R. His task is to recover

$$d_j, d_{j+1}, \ldots, d_{j+n/2^{d+1}-1} \ .$$

He proceeds as follows:

1. He starts by checking the first bit of the encoding. If this is a 0-bit, the remaining part is an encoding of $d_j, d_{j+1}, \ldots, d_{j+n/2^{d+1}-1}$ and he is done. Otherwise, he proceeds to the next step.

2. Bob runs the operations in Z_v^0 on $\mathcal{D}$, using the randomness R. Note that these operations are fixed (they all access array entry 0) and are thus known to Bob. He now skips all the instructions Z_v^ℓ in v's left subtree (which are unknown to him). He sets the client memory to what Alice told him it was after processing $Z_v^0 \circ Z_v^\ell$. He then overwrites all memory cells that appear in $P_v(Z_v)$ (Alice sent the addresses and contents of these as they were right after processing $Z_v^0 \circ Z_v^\ell$). Let Z_v^r denote the operations in v's right subtree. He then starts processing the operations Z_v^r using the randomness R, starting with the client memory contents that Alice sent him. We claim that this will give exactly the same execution as when Alice executed Z_v^r. To see this, consider any memory cell with address s and look at the first time it is probed during Bob's simulation of Z_v^r. There are two cases: either the contents were overwritten due to Alice's message. In this case, the contents are consistent with Alice's execution. Otherwise, the probe must be assigned to some node $w \in \mathcal{T}(Z_v)$ other than v. If w is an ancestor of v, then the cell cannot have been updated during Z_v^ℓ (by definition of how we assign probes to nodes in $\mathcal{T}$) and Bob has the correct contents from his own simulation of Z_v^0. If w is a descendant of v, it means that the cell was already probed during Z_v^r, contradicting that this was the first probe to the cell. Since Bob can finish the simulation (using the randomness R), he gets the same set of answers to all read operations in v's right subtree as Alice did. Finally, he uses the last part of Alice's message to correct the answers to all read operations in Z_v^r which fail. He is now done since the answers to the read operations in Z_v^r reveal $d_j, d_{j+1}, \ldots, d_{j+n/2^{d+1}-1}$.

Analysis. What remains is to show that the above encoding length is less than $nr/2^{d+1}$ in expectation, yielding the sought contradiction. If G denotes the event that Alice writes a 0-bit, we have that the expected length of the encoding is no more than:

$$1 + \Pr[G] \cdot nr/2^{d+1} + (1 - \Pr[G])(m + (21/25)nr/2^{d+1}).$$

Since $5 \leq d \leq (1/2)\lg(nr/m)$, we have

$$m \leq nr/2^{2d} = (nr/2^{d+1})/2^{d-1} \leq (1/16)(nr/2^{d+1}) \ .$$

Therefore the above is no more than:

$$\begin{aligned} 1 + \Pr[G] \cdot nr/2^{d+1} + (1 - \Pr[G])(1/16 + 21/25)nr/2^{d+1} \leq \\ 1 + \Pr[G] \cdot nr/2^{d+1} + (1 - \Pr[G])(91/100)nr/2^{d+1}. \end{aligned}$$

Since the failure probability of $\mathcal{D}$ is no more than $1/32$, it follows from Markov's inequality and linearity of expectation that $\Pr[|F| \geq (1/8)n/2^{d+1}] \leq 1/4$. By a union bound, we have

$$\begin{aligned} \Pr[G] &\leq \Pr[|P_v(Z_v)| \geq (1/100)nr/(w2^d)] + \Pr[|F| \geq (1/8)n/2^{d+1}] \\ &\leq 1/2 + 1/4 \leq 3/4. \end{aligned}$$

This means that the expected length of our encoding is no more than

$$1 + (3/4) \cdot nr/2^{d+1} + (1/4)(91/100)nr/2^{d+1} < nr/2^{d+1}.$$

This gives our sought contradiction and completes the proof of Theorem 2.

4 Conclusion and Future Work

It is 22 years since Goldreich and Ostrovsky proved the ORAM lower bound [GO96] assuming statistical security and "balls in bins". No progress was done on strengthening the bound for two decades. Two years ago, Boyle and Naor asked the question, *Is There an Oblivious RAM Lower Bound?* [BN16]. We have answered this question in the affirmative by eliminating both restrictions of the Goldreich-Ostrovsky lower bound.

The oblivious cell probe model and our lower bound for the array maintenance problem and online ORAMs open up a number of exciting questions. A number of papers (cf. [WNL+14]) have designed oblivious data structures. There is no reason why our proof technique cannot also be applied to prove lower bounds for such oblivious data structures.

Acknowledgment. Kasper Green Larsen wishes to thank Vinod Vaikuntanathan for introducing him to oblivious RAMs and oblivious data structures during a visit at MIT, eventually leading to the results in this paper.

References

[AFN+16] Abraham, I., Fletcher, C.W., Nayak, K., Pinkas, B., Ren, L.: Asymptotically tight bounds for composing ORAM with PIR. Cryptology ePrint Archive, Report 2016/849 (2016). https://eprint.iacr.org/2016/849

[BCP16] Boyle, E., Chung, K.-M., Pass, R.: Oblivious parallel RAM and applications. In: Kushilevitz, E., Malkin, T. (eds.) [KM16], pp. 175–204 (2016)

[BN16] Boyle, E., Naor, M.: Is there an oblivious RAM lower bound? In: Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, pp. 357–368 (2016)

[CLP14] Chung, K.-M., Liu, Z., Pass, R.: Statistically-secure ORAM with $\tilde{O}(\lg^2 n)$ Overhead. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part II. LNCS, vol. 8874, pp. 62–81. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45608-8_4

[DMN11] Damgård, I., Meldgaard, S., Nielsen, J.B.: Perfectly secure oblivious RAM without random oracles. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 144–163. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19571-6_10

[DvDF+16] Devadas, S., van Dijk, M., Fletcher, C.W., Ren, L., Shi, E., Wichs, D.: Onion ORAM: a constant bandwidth blowup oblivious RAM. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016. LNCS, vol. 9563, pp. 145–174. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49099-0_6

[GHL+14] Gentry, C., Halevi, S., Lu, S., Ostrovsky, R., Raykova, M., Wichs, D.: Garbled RAM revisited. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 405–422. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_23

[GLO15] Garg, S., Lu, S., Ostrovsky, R.: Black-box garbled RAM. In: Guruswami, V. (ed.) IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17–20 October 2015, pp. 210–229. IEEE Computer Society (2015)

[GM11] Goodrich, M.T., Mitzenmacher, M.: Privacy-preserving access of outsourced data via oblivious RAM simulation. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part II. LNCS, vol. 6756, pp. 576–587. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22012-8_46

[GMOT12] Goodrich, M.T., Mitzenmacher, M., Ohrimenko, O., Tamassia, R.: Privacy-preserving group data access via stateless oblivious RAM simulation. In: Rabani, Y. (ed.) [Rab12], pp. 157–167 (2012)

[GO96] Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious RAMs. J. ACM **43**(3), 431–473 (1996)

[Goo17] Goodrich, M.T.: BIOS ORAM: improved privacy-preserving data access for parameterized outsourced storage. In: Thuraisingham, B.M., Lee, A.J. (eds.) Proceedings of the 2017 on Workshop on Privacy in the Electronic Society, Dallas, TX, USA, 30 October–3 November 2017, pp. 41–50. ACM (2017)

[Goo18] Goodrich, M.T.: Isogrammatic-fusion ORAM: improved statistically secure privacy-preserving cloud data access for thin clients. In: Proceedings of the 13th ACM ASIA Conference on Information, Computer and Communication Security (2018, to appear)

[KLO12] Kushilevitz, E., Lu, S., Ostrovsky, R.: On the (in)security of hash-based oblivious RAM and a new balancing scheme. In: Rabani, E. (ed.) [Rab12], pp. 143–156 (2012)

[KM16] Kushilevitz, E., Malkin, T. (eds.): TCC 2016, Part II. LNCS, vol. 9563. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49099-0

[Lar12] Larsen, K.G.: The cell probe complexity of dynamic range counting. In: Proceedings of the 44th ACM Symposium on Theory of Computation, pp. 85–94 (2012)

[LO17] Lu, S., Ostrovsky, R.: Black-box parallel garbled RAM. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part II. LNCS, vol. 10402, pp. 66–92. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63715-0_3

[LWY18] Larsen, K.G., Weinstein, O., Yu, H.: Crossing the logarithmic barrier for dynamic boolean data structure lower bounds. In: Symposium on Theory of Computing, STOC 2018 (2018, to appear)

[PD06] Pătraşcu, M., Demaine, E.D.: Logarithmic lower bounds in the cell-probe model. SIAM J. Comput. **35**(4), 932–963 (2006)

[PR10] Pinkas, B., Reinman, T.: Oblivious RAM revisited. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 502–519. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_27

[Rab12] Rabani, Y. (ed.) Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, 17–19 January 2012. SIAM (2012)

[SS13] Stefanov, E., Shi, E.: Oblivistore: high performance oblivious distributed cloud data store. In 20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, 24–27 February 2013. The Internet Society (2013)

[SvDS+13] 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: Sadeghi, A.-R., Gligor, V.D., Yung, M. (eds.) 2013 ACM SIGSAC Conference on Computer and Communications Security, CCS 2013, Berlin, Germany, 4–8 November 2013, pp. 299–310. ACM (2013)

[WNL+14] Wang, X.S., Nayak, K., Liu, C., Hubert Chan, T.-H., Shi, E., Stefanov, E., Huang, Y.: Oblivious data structures. In: Ahn, G.-J., Yung, M., Li, N. (eds.) Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, 3–7 November 2014, pp. 215–226. ACM (2014)

[WST12] Williams, P., Sion, R., Tomescu, A.: Privatefs: a parallel oblivious file system. In: Yu, T., Danezis, G., Gligor, V.D. (eds.) The ACM Conference on Computer and Communications Security, CCS 2012, Raleigh, NC, USA, 16–18 October 2012, pp. 977–988. ACM (2012)

[Yao81] Yao, A.C.-C.: Should tables be sorted? J. ACM **28**(3), 615–628 (1981)