Lower Bounds for Differentially Private RAMs

Giuseppe Persiano[1,2] and Kevin Yeo[1](✉)

[1] Google LLC, Mountain View, USA
giuper@gmail.com, kwlyeo@google.com
[2] Università di Salerno, Salerno, Italy

Abstract. In this work, we study privacy-preserving storage primitives that are suitable for use in data analysis on outsourced databases within the differential privacy framework. The goal in differentially private data analysis is to disclose global properties of a group without compromising any individual's privacy. Typically, differentially private adversaries only ever learn global properties. For the case of outsourced databases, the adversary also views the patterns of access to data. Oblivious RAM (ORAM) can be used to hide access patterns but ORAM might be excessive as in some settings it could be sufficient to be compatible with differential privacy and only protect the privacy of individual accesses.

We consider (ϵ, δ)-*Differentially Private RAM*, a weakening of ORAM that only protects individual operations and seems better suited for use in data analysis on outsourced databases. As differentially private RAM has weaker security than ORAM, there is hope that we can bypass the $\Omega(\log(nb/c))$ bandwidth lower bounds for ORAM by Larsen and Nielsen [CRYPTO '18] for storing an array of n b-bit entries and a client with c bits of memory. We answer in the negative and present an $\Omega(\log(nb/c))$ bandwidth lower bound for privacy budgets of $\epsilon = O(1)$ and $\delta \leq 1/3$.

The *information transfer* technique used for ORAM lower bounds does not seem adaptable for use with the weaker security guarantees of differential privacy. Instead, we prove our lower bounds by adapting the *chronogram* technique to our setting. To our knowledge, this is the first work that uses the chronogram technique for lower bounds on privacy-preserving storage primitives.

1 Introduction

In this work, we study *privacy-preserving storage* schemes involving a client and an untrusted server. The goal is to enable the client to outsource the storage of data to the server such that the client may still perform operations on the stored data (e.g. retrieving and updating data). For privacy, the client wishes to keep the stored data hidden from server. One way to ensure the contents of the data remain hidden is for the client to encrypt all data before uploading to the server. However, the server can still view how the encrypted data is accessed as the client performs operations. Previous works such as [4,20] have shown that the

Y. Ishai and V. Rijmen (Eds.): EUROCRYPT 2019, LNCS 11476, pp. 404–434, 2019.
https://doi.org/10.1007/978-3-030-17653-2_14

leakage of patterns of access to encrypted data can be used to compromise the privacy of the encrypted data. Therefore, a very important privacy requirement is also to protect the access patterns.

The traditional way to define the privacy of access pattern is *obliviousness.* An oblivious storage primitive ensures that any adversary that is given two sequences of operations of equal length and observes the patterns of data access performed by one of the two sequences cannot determine which of the two sequences induced the observed access pattern. The most famous oblivious storage primitive is Oblivious RAM (ORAM) that outsources the storage of an array and allows clients to retrieve and update array entries. ORAM was first introduced by Goldreich [16] who presented an ORAM with sublinear amortized bandwidth per operation for clients with constant size memory. Goldreich and Ostrovsky [17] give the first ORAM construction with polylogarithmic amortized bandwidth per operation. In the past decade, ORAM has been the subject of extensive research [18,19,21,27,28,31,32] as well as variants such as statistically secure ORAMs [7,8], parallel ORAMs [2,5,6] and garbled RAMs [14,15,25]. The above references are just a small subset of all the results for ORAM constructions.

Instead, we focus on a different definition of privacy using *differential privacy* [10–12]. The representative scenario for differential privacy is *privacy-preserving data analysis* which considers the problem of disclosing properties about an entire database while maintaining the privacy of individual database records. A mechanism or algorithm is considered differentially private if any fixed disclosure is almost as likely to be outputted for two different input databases that only differ in exactly one record. As a result, an adversary that views the disclosure is unable to determine whether an individual record was part of the input used to compute the disclosure. We consider the scenario of performing privacy-preserving data analysis on data outsourced to an untrusted server. By viewing the patterns of access to the outsourced data, the adversarial server might be able to determine which individual records were used to compute the disclosure compromising differential privacy.

One way to protect the patterns of data access is to outsource the data using an ORAM. However, in many cases, it turns out that the obliviousness guarantees of ORAM may be stronger than required. For example, let's suppose that we wish to disclose a differentially private regression model over a sample of the outsourced data. ORAM guarantees that the identity of all sampled database records will remain hidden from the adversary. On the other hand, the differentially private regression model only provides privacy about whether an individual record was sampled or not. Instead of obliviousness, we want a weaker notion of privacy for access patterns suitable for use with differentially private data analytics. With a weaker notion of privacy, there is hope for a construction with better efficiency than ORAM.

With this in mind, we turn to the notion of *differentially private access* which provides privacy for individual operations but might reveal information about a sequence of many operations. Differentially private access has been previously considered in [33,34]. In particular, this privacy notion ensures that the patterns of data access caused by a fixed sequence of operations is almost as likely to be induced by another sequence of operations of the same length with a single different operation. We define (ϵ, δ)-*differentially private RAM* as a storage primitive that outsources the storage of an array in a manner that allows a client to retrieve and update array entries while providing differentially private access. As this privacy notion is weaker than obliviousness, the $\Omega(\log(n/c))$ lower bounds for ORAMs that store n array entries and clients with c bits of storage by Larsen and Nielsen [23] do not apply. There is hope to achieve a differentially private RAM construction with smaller bandwidth. In this work, we answer in the negative and show that an $\Omega(\log(n/c))$ bandwidth lower bound also exists for differentially private RAM for typical privacy budgets of $\epsilon = O(1)$ and $\delta \leq 1/3$. As differential privacy with budgets of $\epsilon = O(1)$ and $\delta \leq 1/3$ provide weaker security than obliviousness, any ORAM is also a differentially private RAM. Therefore, our lower bounds show that the ORAM constructions by Patel *et al.* [27] and by Asharov *et al.* [1] are, respectively, asymptotically optimal up to an $O(\log \log n)$ factor and asymptotically optimal (ϵ, δ)-differentially private RAM for any constant ϵ and $\delta \leq 1/3$ and any block size b. Our results also prove that Path ORAM [32] is tight, for $b = \Omega(\log^2 n)$.

1.1 Our Results

In this section, we will present our contributions. We first describe the scenarios where our lower bounds apply. Our lower bounds apply to *differentially private* RAMs that process operations in an *online* fashion. The RAM must be both *read-and-write*, that is, the set of permitted operations include both reading and writing array entries. The server that stores the array is assumed to be *passive* in that the server may not perform any computation beyond retrieving and overwriting cells but no assumptions are made on the *storage encoding* of the array. Finally, we assume that the adversary is *computationally bounded.* We now go into detail about each of these requirements.

Differential Privacy. The goal of differential privacy is to ensure that the removal or replacement of an individual in a large population does not significantly affect the view of the adversary. Differential privacy is parameterized by two values $0 \leq \epsilon$ and $\delta \in [0, 1]$. The value ϵ is typically referred to as the *privacy budget.* When $\delta = 0$, the notion is known as *pure* differential privacy while, if $\delta > 0$, the notion is known as *approximate* differential privacy. In our context, an *individual* is a single operation in a sequence (the *population*) of read (also called queries) and write (also called updates) operations over an array of n entries stored on a, potentially adversarial, remote server. For any implementation $\mathbf{DS}$ and for any sequence Q, we define $\mathbb{V}_{\mathbf{DS}}(Q)$ to be the view of the server when sequence Q is executed by $\mathbf{DS}$. A differentially private RAM, $\mathbf{DS}$, is defined to ensure that the adversary's view on one sequence of operations should not be significantly different when $\mathbf{DS}$ executes another sequence of operations which differs for only one operation. We assume that our adversaries are computationally bounded.

Formally, if $\mathbf{DS}$ is (ϵ, δ)-differentially private, then for any two sequences Q_1 and Q_2 that differ in exactly one operation, it must be that $\Pr[\mathcal{A}(\mathbb{V}_{\mathbf{DS}}(Q_1)) = 1] \leq e^{\epsilon} \Pr[\mathcal{A}(\mathbb{V}_{\mathbf{DS}}(Q_2)) = 1] + \delta$ for any probabilistic polynomial time (PPT) algorithm $\mathcal{A}$. The notion of computational differential privacy was studied by Mironov *et al.* [26] where various classes of privacy were described. Our lower bounds consider the weakest privacy class and, thus, apply to all privacy classes in [26]. In the majority of scenarios, differential privacy is only considered useful for the cases when $\epsilon = O(1)$ and δ is negligible. This is exactly the scenario where our lower bounds will hold. In fact, our lower bounds hold for any $\delta \leq 1/3$. We note that differential privacy with $\epsilon = O(1)$ and $\delta \leq 1/3$ is a weaker security notion than obliviousness. Obliviousness is equivalent to differential privacy when $\epsilon = 0$ and δ is negligible. Therefore, our lower bounds also hold for ORAM and match the lower bounds of Larsen and Nielsen [23]. We refer the reader to Sect. 2 for a formal definition of differential privacy.

Online RAMs. It is important that we discuss the notion of *online* vs. *offline* processing of operations by RAMs. In the offline scenario, it is assumed that all operations are given before the RAM must start processing updates and answering queries. The first ORAM lower bound by Goldreich and Ostrovsky [17] considered offline ORAMs with "balls-and-bins" encoding and security against an all-powerful adversary. "Balls-and-bins" refers to the encoding where array entries are immutable balls and the only valid operation is to move array entries into various memory locations referred to as bins. Boyle and Naor [3] show that proving an offline ORAM lower bound for non-restricted encodings is equivalent to showing lower bounds in sorting circuits, which is a long-standing problem in complexity. Instead, we consider online RAMs where operations arrive one at a time and must be processed before receiving the next operation. The assumption of online operations is realistic as the majority of RAM constructions consider online operations and almost all applications of RAMs consider online operations. Our lower bounds only apply for online differentially private RAMs.

Read-and-Write RAMs. Traditionally, all ORAM results consider the scenario where the set of valid operations include both reading and writing array entries. A natural relaxation would be to consider *read-only* RAMs where the only valid operation is reading array entries. Any lower bound on read-only RAMs would also apply to read-and-write RAMs. However, in a recent work by Weiss and Wichs [36], it is shown that any lower bounds for read-only ORAMs would imply very strong lower bounds for either sorting circuits and/or locally decodable codes (LDCs). Proving lower bounds for LDCs has, like sorting circuits, been an open problem in the world of complexity theory for more than a decade. As differential privacy is weaker than obliviousness, any lower bounds on read-only, differentially private RAMs also imply lower bounds on read-only ORAMs. To get around these obstacles, our work focuses only on proving lower bounds for read-and-write differentially private RAMs.

Passive Server. In our work, we will assume that the server storing the array is *passive*, which means that the server will not any perform computation beyond retrieving and overwriting the contents of the local memory cell to satisfy the client's requests. This assumption is necessary as there are ORAM constructions that use server computation to achieve constant bandwidth operations [9]. Therefore, our lower bounds on bandwidth only apply to differentially private RAMs with a passive server. For active servers we can reinterpret our results as lower bounds on the amount of server computation required to guarantee differential privacy.

We now informally present our main contribution.

Theorem 1 (informal). *Let* **DS** *be any online, read-and-write RAM that stores n array entries each of size b bits on a passive server without any restrictions on storage encodings. Suppose that the client has c bits of storage. Assume that* **DS** *provides (ϵ, δ)-differential privacy against a computational adversary that views all cell probes performed by the server. If $\epsilon = O(1)$ and $0 \leq \delta \leq 1/3$, then the amortized bandwidth of both reading and writing array entries by* **DS** *is $\Omega(b \log(nb/c))$ bits or $\Omega(\log(nb/c))$ array entries. In the natural scenario where $c \leq b \cdot n^{\alpha}$ for some $0 \leq \alpha < 1$, then $\Omega(\log n)$ array entries of bandwidth are required.*

1.2 Previous Works

In this section, we present a small survey of previous works on data structure lower bounds. We also describe the first lower bound for data structures that provide privacy guarantees.

The majority of data structure lower bounds are proved using the cell probe model introduced by Yao [37], which only charges for accessing memory and allows unlimited computation. In the case for passive servers that only retrieve and overwrite memory, the costs of the cell probe model directly imply costs in bandwidth. The *chronogram* technique was introduced by Fredman and Saks [13] which can be used to prove $\Omega(\log n / \log\log n)$ lower bounds. Pătraşcu and Demaine [30] presented the *information transfer* technique which could be used to prove $\Omega(\log n)$ lower bounds. Larsen [22] presented an $\tilde{\Omega}(\log^2 n)$ lower bound for two-dimensional dynamic range counting, which remains the highest lower bound proven for any $\log n$ output data structures. Recently, Larsen *et al.* [24] presented an $\tilde{\Omega}(\log^{1.5} n)$ lower bound for data structures with single bit outputs which is the highest lower bound for decision query data structures.

For ORAM, Goldreich and Ostrovsky [17] presented an $\Omega(\log_c n)$ lower bound for clients with storage of c array entries. However, Boyle and Naor [3] showed that this lower bound came with the cavaets that the lower bound only for statistical adversaries and constructions in "balls-and-bins" model where array entries could only be moved between memory and not encoded in a more complex manner. Furthermore, Boyle and Naor [3] show that proving lower bounds for offline ORAMs and arbitrary storage encodings imply sorting circuit lower bounds.

In their seminal work, Larsen and Nielsen [23] presented an $\Omega(\log(n/c))$ bandwidth lower bound removing the cavaets such that lower bounds applies to any types of storage encodings and computational adversaries. Recently, Weiss and Wichs [36] show that lower bounds for online, read-only ORAMs would imply lower bounds for either sorting circuits and/or locally decodable codes.

We present a brief overview of the techniques used by Larsen and Nielsen [23], which uses the information transfer technique. We also describe why information transfer does not seem to be of use for differentially private RAM lower bounds. Information transfer first builds a binary tree over $\Theta(n)$ operations where the first operation is assigned to the leftmost leaf, the second operation is assigned to the second leftmost leaf and so forth. Each cell probe is assigned to at most one node of the tree as follows. For a cell probe, we identify the operation that is performing the probe as well as the most recent operation that overwrote the cell that is being probed. The cell probe is assigned to the lowest common ancestor of the leaves associated with the most recent operation to overwrite the cell and the operation performing the probe. Let us fix any node of the tree and consider the subtree rooted at the fixed node. It can be shown that the probes assigned to the root is the entirety of information that can be transferred from the updates of the left subtree to be used to answer queries in the right subtree. Consider the sequence of operations where all leaves in the left subtree write a randomly chosen b-bit string to unique array entries and all leaves in the right subtree read an unique, updated array entry. For any **DS** to return the correct b-bit strings asked by the queries in the right subtree, a large amount of information must be transferred from the left subtree to the right subtree. Thus, many probes should be assigned to the root of this subtree. Suppose that for another sequence of operations, **DS** assigns significantly less probes to the root of this subtree. Then, a computational adversary can count the probes and distinguish between the worst case sequence and any other sequence contradicting obliviousness. As a result, there must be many probes assigned to each node of the information transfer tree. Each cell probe is assigned to at most one node. So, summing up the tree provides a lower bound on the number of cell probes required.

Unfortunately, we are unable to use the information transfer technique to prove lower bounds for differentially private RAMs. The main issue comes from the fact that differentially private RAMs have significantly weaker privacy guarantees compared to ORAMs. When $\epsilon = \Theta(1)$, the probabilistic requirements of the adversary's view when **DS** processes two sequences Q_1 and Q_2 degrade exponentially in the number of operations that Q_1 and Q_2 differ in. On the other hand, the privacy requirements of obliviousness do not degrade when considering two sequences that differ in many operations. Larsen and Nielsen [23] use obliviousness to argue that the adversary's view for the worst case sequence of any subtree cannot differ significantly from any other sequence. However, for any fixed sequence of operations, the worst case sequence for the majority of subtrees differ in many operations (on the order of the number of leaves of the subtree). Applying differential privacy will not yield strong requirements for the number of cell probes assigned to the majority of the nodes in the information transfer binary tree. As a result, we could not adapt the information transfer technique for differentially private RAM lower bounds and resort to other techniques.

1.3 Overview of Our Proofs

In this section, we present an overview of the proof techniques used in Sects. 3 and 4. Our lower bounds use ideas from works by Pătraşcu and Demaine [30] and Pătraşcu [29]. However, we begin by reviewing the original chronogram technique of Fredman and Saks [13].

Consider an ORAM that stores n b-bit array entries in a cell probe model with w-bit cells. We make the reasonable assumption that $w = \Omega(\log n)$ so that a cell can hold the index of an entry. Let t_w and t_r denote the number of cell probes of an update (`write`) and of a query (`read`) operation, respectively, and consider a sequence of $\Theta(n)$ update operations followed by a single query. Starting from the query and going backwards in time, updates are partitioned into exponentially increasing epochs at some rate r, so that the i-th epoch will have $\ell_i = r^i$ update operations. Epochs are indexed in reverse time, so the smallest epoch closest to the query is epoch 1. The goal of the chronogram is to prove that there exists a query that requires information from many of the epochs simultaneously. To do this, we first observe that if each update writes a randomly and independently chosen b-bit entry, an update operation preceding epoch i cannot encode any information about epoch i. Therefore, all information about entries updated in epoch i can only be found in cells that have been written as part of the update operations of epoch i or any following epochs, that is epochs $i-1, \ldots, 1$. Since each update stores b random bits, epoch i encodes $\ell_i \cdot b$ bits in total. On the other hand, the write operations of epochs $i-1, i-2, \ldots, 1$ can probe at most $t_w(r^{i-1} + \ldots + r)$ and by setting $r = (t_w w)^2$, we obtain that $O(\ell_i/(t_w w^2))$ cells can be probed and $O(\ell_i/(t_w w))$ bits can be written. As a result, the majority of the bits encoded by updates in epoch i remain in cells last written in epoch i. Thus, if we construct a random query such that $\Omega(b)$ bits must be transferred from each epoch, then we obtain that $\max\{t_w, t_r\} = \Omega((b/w)\log_r n) = \Omega((b/w)\log n/\log\log n)$.

This lower bound can be improved to $\Omega((b/w)\log n)$ by using an improvement of the chronogram technique by Pătraşcu [29]. In the original chronogram technique, the epochs are fixed since the query's location and the number of updates are fixed. An algorithm may attempt to target an epoch i by having all future update operations encode information only about epoch i. To counteract this, we consider a harder update sequence where epoch locations cannot be predicted by the algorithm. Specifically, we consider a sequence that consists of a random number of update operations followed by a single query operation. For such a sequence, even if an algorithm attempts to target epoch i, it cannot pinpoint the location of epoch i (remember that epochs are indexed starting from the query operation and going back in time) and may only prepare over all possible query locations. We show that any update operation may now only encode $O(t_w w/\log_r n)$ about epoch i where $\log_r n$ is the number of epochs. As a result, future update operations can only encode a $O(1/\log_r n)$ fraction as much information about epoch i as the previous lower bound attempt. This allows us to fix $r = 2$ which increases the number of epochs $\log n$. If we can find a query that requires $\Omega(b)$ bits of information transfer from the majority of epochs, we can prove that $\max\{t_w, t_r\} = \Omega((b/w) \log n)$.

For differentially private RAMs, the update operations enable overwriting a b-bit array entry while the query operations allow retrieving an array entry. We choose our update operations to overwrite unique array entries and each entry is overwritten with a value that is independently and uniformly chosen at random from $\{0, 1\}^b$. Focus on an epoch i and consider picking a random query from the ℓ_i array indices updated in epoch i. The majority of these queries must read $\Omega(b)$ bits from cells last written in epoch i as future operations cannot encode all $\ell_i \cdot b$ bits encoded by epoch i. As a result, there exists some query such that $\Omega(b)$ bits must be transferred from epoch i for all sufficiently large epochs. We use differential privacy to show that $\Omega(b)$ bits must be transferred from all sufficiently large epochs. Consider two sequences of operations that only differ in the final query operation and suppose that the first query requires $\Omega(b)$ bits from epoch i. If the latter query transfers $o(b)$ bits from epoch i, the adversary can distinguish between the two sequences with high probability and this contradicts differential privacy as the two sequences only differ in one operations. Therefore, we can prove that $\Omega(b)$ bits have to be transferred from most epochs and thus $\max\{t_w, t_r\} = \Omega((b/w) \log n)$. The proof of this lower bound is found in Sect. 3.

A stronger lower bound is obtained in Sect. 4 using more complex epoch constructions, adapting ideas from [30] and [29]. The lower bound outlined above shows that $\max\{t_w, t_r\} = \Omega((b/w) \log n)$ but it does not preclude the case where $t_w = \Theta((b/w) \log n)$ and $t_r = O(1)$, for example. We show this cannot be the case. In particular, we show that if $\max\{t_w, t_r\} = O((b/w) \log n)$, then it must be the case that $t_w = \Theta((b/w) \log n)$ and $t_r = \Theta((b/w) \log n)$. The idea is to consider different epoch constructions for the cases when t_w and t_r are small, respectively. When $t_w = o((b/w) \log n)$, we know that operations in future epochs cannot encode too much information. We consider an epoch construction where epochs grow by a rate of $\omega(1)$ every r epochs thus increasing the number of epochs to $\omega(\log n)$. In exchange, there are many operations after an epoch i. Since t_w is small, the future operations may not encode too much information about epoch i ensuring most of the information about epoch i remain in cells last written during epoch i. As a result, it can be shown again that $\Omega(b)$ bits must be read from many epochs implying an $t_r = \omega((b/w) \log n)$ lower bound. On the other hand, consider the case when $t_r = o((b/w) \log n)$. We consider epoch constructions that increase exponentially with rate $r = \omega(1)$. As a result, the number of operations after epoch i is a factor of $O(1/r)$ smaller than the ℓ_i operations in epoch i and there are $\Theta(\log_r n)$ epochs. If $t_r = o((b/w) \log_r n)$, then a query operation may not read $\Omega(b)$ from each of the epochs. Instead, update operations must encode a large amount of information about previous epochs to compensate for t_r being so small. As a result, it can be shown that $t_w = \omega((b/w) \log n)$. Combining the above two statements implies that if $\max\{t_w, t_r\} = O((b/w) \log n)$, then $t_w = \Theta((b/w) \log n)$ and $t_r = \Theta((b/w) \log n)$.

2 Differentially Private Cell Probe Model

We start by formalizing the model for which we prove our lower bounds. We rely on the *cell probe model*, first described by Yao [37], and typically used to prove lower bounds for data structures without any requirements for privacy of the stored data and/or the operations performed. In a recent work by Larsen and Nielsen [23], the *oblivious cell probe model* was introduced and used to prove a lower bound for oblivious RAM. The oblivious cell probe model was defined for any data structures where the patterns of access to memory should not reveal any information about the operations performed. We generalize the oblivious cell probe model and present the (ϵ, δ)*-differentially private cell probe model.* In this new model, all data structures are assumed to provide differential privacy for the operations performed with respect to memory accesses viewed by the adversary. The differentially private cell probe model is a generalization of the oblivious cell probe model as obliviousness is equivalent to differential privacy with $\epsilon = 0$ and $\delta = \mathsf{negl}(n)$, that is, any function negligible in the number of items stored in the data structure.

The cell probe model is an abstraction of the interaction between CPUs and word-RAM memory architectures. Memory is defined as an array of *cells* such that each cell contains exactly w bits. Any *operation* of a data structure is allowed to *probe* cells where a probe can consist of either reading the contents of a cell or overwriting the contents of a cell. The running time or cost for any operation of a data structure is measured by the number of cell probes performed. An algorithm is free to do unlimited amounts of computation based on the contents of probed cells.

In this paper we are interested in data structures that provide privacy of the operations performed in a scenario involving two parties denoted the *client* and the *server*. The client outsources the storage of data to the server while maintaining the ability to perform some set of operations over the data efficiently. In addition, the client wishes to hide the operations performed from the adversarial server that views the contents of all cells in memory as well as the sequence of cells probed in memory. The crucial privacy requirement is that the server does not learn about the contents and the sequence of accesses performed by the client's storage. To properly capture the above setting, Larsen and Nielsen [23] defined the *oblivious cell probe model* and proved lower bounds for oblivious RAMs. We introduce the *differentially private cell probe model* that is identical to the oblivious cell probe model of Larsen and Nielsen, except for the simple replacement of obliviousness with differential privacy as the privacy requirement. For a full description of the oblivious cell probe model, we refer the reader to Sect. 2 of [23]. To formally define the differentially private cell probe model, we first describe a *data structure problem* as well as a *differentially private cell probe data structure* for any data structure problem.

Definition 1. *A* data structure problem *P is defined by a tuple (U, Q, O, f) where*

1. *U is the universe of all* update *operations;*
2. *Q is the universe of all* query *operations;*

3. O is domain of all possible outputs for all queries;
4. $f : U^* \times Q \to O$ is a function that describes the desired output of any query $q \in Q$ given the history of all updates, $(u_1, u_2, \ldots, u_m) \in U^*$.

A *differentially private cell probe data structure* **DS** for the data structure problem P consists of a randomized algorithm implementing update and query operations for P. **DS** is parameterized by the integers c and w denoting the client storage and cell size in bits respectively. Additionally, **DS** is given a random string $\mathcal{R}$ of finite length r containing all randomness that **DS** will use. Note that $\mathcal{R}$ can be arbitrarily large and, thus, contain all the randomness of a random oracle. Given the random string, our algorithms can be viewed as deterministic. Each algorithm is viewed as a finite decision tree executed by the *client* that probes (read or overwrite) memory cells owned by the *server*. For each $q \in Q$ and $u \in U$, there exists a (possibly) different decision tree. Each node in the decision tree is labelled by an index indicating the location of the server-held memory cell to be probed. For convenience, we will assume that a probe may both read and overwrite cell contents. This only reduces the number of cell probes by a factor of at most 2. Additionally, all leaf nodes are labelled with an element of O indicating the output of **DS** after execution.

Each edge in the tree is labelled by four bit strings. The first bit-string of length w represents the contents of the cell probed. The next w-bit string represents the new cell contents after overwriting. There are two c-bit strings representing the current client storage and the new client storage after performing the probe. Finally, there is a r-bit string representing the random string. The client executes **DS** by traversing the decision tree starting from the root. At each node, the client reads the indicated cell's contents. Using the random string, the current client storage and the cell contents, it finds the edge to the next node and updates the probed cell's contents and client storage accordingly. When reaching a leaf, **DS** outputs the element of O denoted at the leaf.

Note, **DS** is only permitted to use the contents of the previously probed cell, current client storage and the random string as input to generate the next cell probe or produce an output. The running time of **DS** is related to the depth of the decision tree as each edge corresponds to a cell probe. Furthermore, as the servers are passive, the server can only either update or retrieve a cell for the client. As a result, the running time (number of cell probes) multiplied by w (the cell size) gives us the bandwidth of the algorithm in bits. We now define the failure probability of **DS**.

Definition 2. *A* **DS** *for data structure problem $P = (U, Q, O, f)$ has* failure probability $0 \leq \alpha \leq 1$ *if for every sequence of updates $u_1, \ldots, u_m \in U^*$ and query $q \in Q$:*

$$\Pr[\mathbf{DS}(u_1, \ldots, u_m, q) \neq f(u_1, \ldots, u_m, q)] \leq \alpha$$

where randomness is over the choice of $\mathcal{R}$.

As $\mathcal{R}$ is finite, it might seem that we do not consider algorithms whose failure probability decreases in the running time but may never terminate. Instead, we can consider a variant of the algorithm that may run for an arbitrary long time but must provide an answer once its failure probability is small enough (for example, negligible in the number of item stored). Therefore, by sacrificing failure probability, we can convert such possibly infinitely running algorithms into finite algorithms with slightly larger failure probabilities. As we will prove our lower bounds for **DS** with failure probabilities at most 1/3, we may also consider these kind of algorithms with vanishing failure probabilities and no termination guarantees.

We now move to privacy requirements and define the random variable $\mathbb{V}_{\mathbf{DS}}(Q)$ as the *adversary's view* of **DS** processing a sequence of operations Q where randomness is over the choice of the random string $\mathcal{R}$. The adversary's view contains the sequence of probes performed by **DS** to server-held memory cells. We stress that the view does not include the accesses performed by **DS** to client storage. We now define *differentially private access.*

Definition 3. **DS** *provides* (ϵ, δ)-differentially private access *against computational adversaries if for any two sequences* $Q = (\mathsf{op}_1, \ldots, \mathsf{op}_m)$ *and* $Q' = (\mathsf{op}'_1, \ldots, \mathsf{op}'_m)$ *such that* $|\{i \in \{1, \ldots, m\} \mid \mathsf{op}_i \neq \mathsf{op}'_i\}| = 1$ *and any PPT algorithm* $\mathcal{A}$*, it holds that*

$$\Pr[\mathcal{A}(\mathbb{V}_{\mathbf{DS}}(Q)) = 1] \leq e^{\epsilon} \cdot \Pr[\mathcal{A}(\mathbb{V}_{\mathbf{DS}}(Q')) = 1] + \delta.$$

Our results focus on *online* data structures where each cell probe may be assigned to a unique operation.

Definition 4. *A* **DS** *is* online *if for any sequence* $Q = (\mathsf{op}_1, \ldots, \mathsf{op}_m)$*, the adversary's view can be split up as:*

$$\mathbb{V}_{\mathbf{DS}}(Q) = (\mathbb{V}_{\mathbf{DS}}(\mathsf{op}_1), \ldots, \mathbb{V}_{\mathbf{DS}}(\mathsf{op}_m))$$

where each cell probe in $\mathbb{V}_{\mathbf{DS}}(\mathsf{op}_i)$ *is performed after receiving* op_i *and before receiving* op_{i+1}.

Finally, we present the definition of an (ϵ, δ)*-differentially private cell probe data structure*. We present a diagram of the model in Fig. 1.

Definition 5. *A* **DS** *for problem* P *is an* (ϵ, δ)-differentially private cell probe data structure *if* **DS** *has failure probability* $1/3$*, provides* (ϵ, δ)*-differentially private access and is online.*

We comment that the failure probability of 1/3 does not seem to be reasonable for any scenario. However, proving a lower bound for **DS** with failure probability 1/3 results in stronger lower bounds as they also hold for more reasonable situations with zero or negligibly small failure probabilities.

We now present the *array maintenance* problem introduced by Wang *et al.* [35], which crisply defines the online RAM problem.

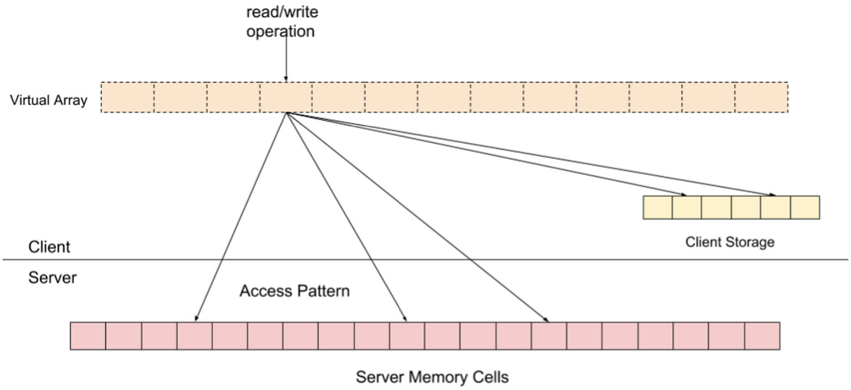

Fig. 1. Diagram of differentially private cell probe model.

Definition 6. *The* array maintenance *problem* AM *is parameterized by two integers* $n, b > 0$ *and defined by the tuple* $(U_{\mathsf{AM}}, Q_{\mathsf{AM}}, O_{\mathsf{AM}}, f_{\mathsf{AM}})$ *where*

- $U_{\mathsf{AM}} = \{\mathtt{write}(i, B) : i = 0, \ldots, n-1, B \in \{0,1\}^b\}$;
- $Q_{\mathsf{AM}} = \{\mathtt{read}(i) : i = 0, \ldots, n-1\}$;
- $O_{\mathsf{AM}} = \{0,1\}^b$;

and, for a sequence $Q = (u_1, \ldots, u_m)$ *where* $u_1, \ldots, u_m \in U^*$, f_{AM} *is:*

$$f_{\mathsf{AM}}(Q, \mathtt{read}(i)) = \begin{cases} B, & \textit{where } j \textit{ is largest index such that } q_j = \mathtt{write}(i, B); \\ 0^b, & \textit{if there exists no such } j. \end{cases}$$

In words, the array maintenance problem requires that a data structure to store an array of n entries each of b bits. Each array location is uniquely identified by a number in $[n]$. Typically, it is assumed that a cell is large enough to contain an index. In this case, $w = \Omega(\log n)$. However, in our lower bounds, we will only assume that $w = \Omega(\log\log n)$ to achieve a stronger lower bound. An update operation (also called a *write*) takes as input an integer $i \in [n]$ and a b-bit string B and overwrites the array entry associated with i with the string B. For convenience, we denote a write operation with inputs i and B as $\mathtt{write}(i, B)$. A query operation (also called a *read*) takes as input an integer $i \in [n]$ and returns the current b-bit string of the array entry associated with i. We denote a read operation with input i as $\mathtt{read}(i)$. We will prove lower bounds for (ϵ, δ)*-differentially private RAMs* which are differentially private cell probe data structures for the AM problem.

3 First Lower Bound

Let **DS** be a (ϵ, δ)-differentially private RAM storing n b-bit entries indexed by the integers from 0 to $n-1$. For any sequence of operations $Q = (\mathtt{op}_1, \ldots, \mathtt{op}_m)$, we denote $t_w(Q)$ as the worst case number of cell probes on a write operation and $t_r(Q)$ as the expected amortized number of cell probes on a read operation. Both expectations are over the choice of the random string $\mathcal{R}$ used by **DS**. We write t_w and t_r as the largest value of $t_w(Q)$ and $t_r(Q)$ respectively over all sequences Q. We assume that cells are of size w bits and the client has c bits of storage. In this section, we will prove the following preliminary result. The result will be strengthened in Sect. 4 where we present our main result.

Theorem 2. *Let $\epsilon > 0$ and $0 \le \delta \le 1/3$ be constants and let* **DS** *be an (ϵ, δ)-differentially private RAM for n b-bit entries implemented over w-bit cells that uses c bits of local storage. If* **DS** *has failure probability at most $1/3$ and $w = \Omega(\log\log n)$, then $t_w + t_r = \Omega\left(\frac{b}{w} \cdot \log\left(\frac{nb}{c}\right)\right)$.*

In terms of block bandwidth, this implies that at least one of read and write has an expected amortized $\Omega(\log(nb/c))$ block bandwidth overhead. The above theorem will be shown when **DS** has to process a sequence Q sampled according to distribution $\mathcal{Q}(0)$. For index $\mathsf{idx} \in \{0, \ldots, n-1\}$, distribution $\mathcal{Q}(\mathsf{idx})$ is defined by the following probabilistic process:

1. Pick m uniformly at random from $\{n/2, n/2 + 1, \ldots, n-1\}$.
2. Draw $B_1, \ldots, B_m$ independently and uniformly at random from $\{0, 1\}^b$.
3. Construct the sequence $U = \mathtt{write}(1, B_1), \ldots, \mathtt{write}(m, B_m)$.
4. Output $Q = (U, \mathtt{read}(\mathsf{idx}))$.

Thus $\mathcal{Q}(\mathsf{idx})$ assigns positive probability to sequences $Q = (U, \mathtt{read}(\mathsf{idx}))$ that consist of a sequence U of m write operation of m b-bit blocks one for each index $1, 2, \ldots, m$, followed by a single read to index idx. It will be useful to define $\mathcal{U}$ to be the distribution of U as determined by the first three steps of the process described above.

In particular, we prove the above theorem using $\mathcal{Q}(0)$, which for convenience, will be denoted by $\mathcal{Q}$ from now on. If privacy is not a concern, sequences in the support of $\mathcal{Q}$ do not seem to require many probes as index 0 is not overwritten. However, the lower bound will, critically, use the fact that the view of any computational adversary cannot differ significantly from the view of sequences whose last operation attempts to read a previously overwritten index $\mathsf{idx} \in \{1, \ldots, m\}$.

We prove the lower bound using the *chronogram* technique first introduced by Fredman and Saks [13] along with the modifications by Pǎtraşcu [29]. The strategy employed by the chronogram technique when applied to a sequence sampled according to $\mathcal{Q}$ goes as follows. For any choice of m, we consider the $n/2$ `write` operations that immediately precede the `read`(0) operation and we split them into consecutive and disjoint groups, which we denote as *epochs*. The epochs will grow exponentially in size and are indexed going backwards in time (order of operations performed). That is, the epoch consisting of the `write` operations immediately preceding the `read` operation will have the smallest index, while the epoch furthest in the past will have the largest index. Note that, when the sequence of operations is sampled according to $\mathcal{Q}$ or, more generally, according to $\mathcal{Q}(\mathsf{idx})$, the set of indices overwritten by the `write` operations that fall into epoch i is a random variable which we denote by $\mathcal{U}^i$ and that depends on the value of m.

To prove Theorem 2, we consider a simple epoch construction. For $i > 0$, epoch i consists of $\ell_i = 2^i$ `write` operations and thus there will be $k = \log_2(n/2+2)-1$ epochs. We also define s_i to be the total size of epochs $1, \ldots, i$. In the epoch construction of this section, we have $s_i = 2^{i+1}-2$. See Fig. 2 for a diagram of the layout of the epochs with regards to a sequence of operations. In Sect. 4, we will derive stronger lower bounds by considering more complex epoch constructions with different parameters.

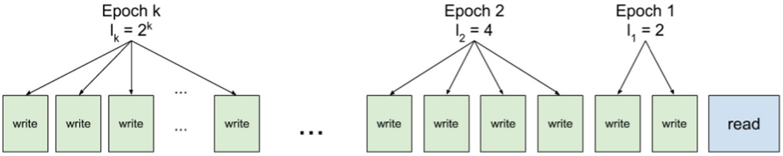

Fig. 2. Diagram of epoch construction of Sect. 3. Operations are performed from left to right.

Defining Random Variables. Since we are considering online data structures, each cell probe performed by **DS** while processing a sequence Q can be uniquely associated to a `read` or a `write` operation of Q. Random variable $\mathcal{T}_w(Q)$ is defined as the set of cell probes performed by **DS** while processing the `write` operations of the sequence Q. Similarly, we define $\mathcal{T}_r(Q)$ as the random variable of the set of cell probes performed by **DS** when processing the `read` operations of Q. The probability spaces of the two variables are over the choice of $\mathcal{R}$.

The following random variables are specifically defined for sequences $Q = (U, \texttt{read}(\mathsf{idx}))$ in the support of distribution $\mathcal{Q}(\mathsf{idx})$, for some idx. We remind the reader that these sequences perform a sequence U consisting of m `write` operations followed by a single `read`(idx) operation. The m `write` operations overwrite entries $1, \ldots, m$ with random b-bit strings. We denote by $\mathcal{T}_w^j(Q)$ the random variable of the cells that are probed during the execution of a `write` operation of epoch j in Q. We further partition the cell probes in $\mathcal{T}_w^j(Q)$ according to the epoch the cell was last overwritten before being probed in epoch j. Specifically, for $i \geq j$, we define $\mathcal{T}_w^{i,j}(Q)$ as the random variable of the subset of the probes of $\mathcal{T}_w^j(Q)$ performed to a cell that was last overwritten by an operation in epoch i. Note that the sets $\mathcal{T}_w^{i,j}(Q)$ for all pairs (i,j) with $i \geq j$ constitute a partition of $\mathcal{T}_w(Q)$. It will be convenient in the proof to define $\mathcal{T}_w^{<i}(Q) = \mathcal{T}_w^{i,1}(Q) \cup \ldots \cup \mathcal{T}_w^{i,i-1}(Q)$ as the set of probes that are performed by an operation in any of epochs $\{1, \ldots, i-1\}$ to a cell that was last overwritten by an operation in epoch i. Note that if two sequences $Q_1 = (U, \mathsf{idx}_1)$ and $Q_2 = (U, \mathsf{idx}_2)$ share the same initial sequence U of `write` operations, then, clearly, $\mathcal{T}_w^{<i}(Q_1) = \mathcal{T}_w^{<i}(Q_2)$ if they both use the same random string $\mathcal{R}$.

Finally, we denote the random variable $\mathcal{T}_r^i(Q)$ as the set of probes performed by the `read` operation of Q to cells that were last overwritten by an operation in epoch i. In Fig. 3, we show a diagram of $\mathcal{T}_w^{<i}(Q)$ and $\mathcal{T}_r^i(Q)$.

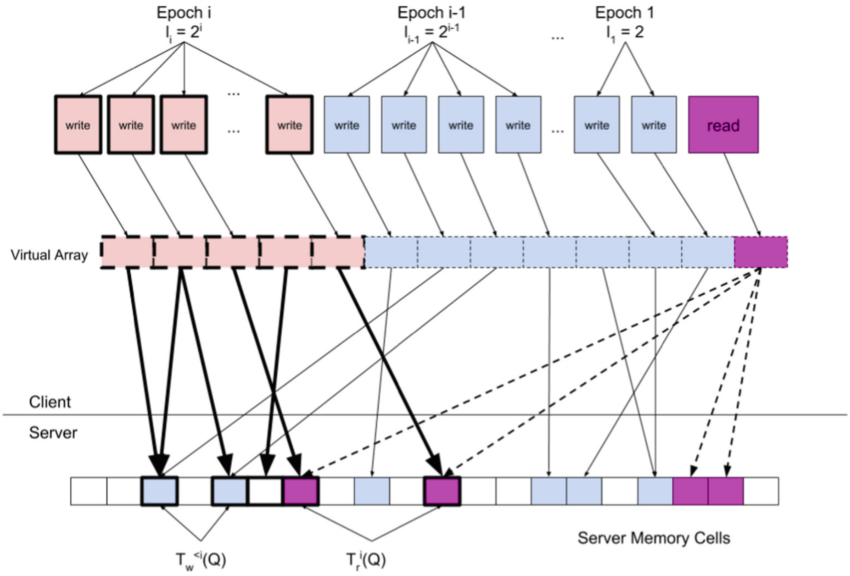

Fig. 3. Diagram of $\mathcal{T}_w^{<i}(Q)$ and $\mathcal{T}_r^i(Q)$.

We extend the definitions above to distributions of sequences in a natural way. For example, $\mathcal{T}_w(\mathcal{Q}(\mathsf{idx}))$ is defined by first picking sequence Q according to $\mathcal{Q}(\mathsf{idx})$ and then sampling a set according to $\mathcal{T}_w(Q)$. Note that, since $\mathcal{T}_w^{<i}(\mathcal{Q}(\mathsf{idx}))$ does not depend on the read operation, we have that for all $\mathsf{idx}_1, \mathsf{idx}_2$ $\mathcal{T}_w^{<i}(\mathcal{Q}(\mathsf{idx}_1)) = \mathcal{T}_w^{<i}(\mathcal{Q}(\mathsf{idx}_2)) = \mathcal{T}_w^{<i}(\mathcal{U})$.

3.1 A Tradeoff Between $\mathcal{T}_w^{<i}(\mathcal{Q})$ and $\mathcal{T}_r^i(\mathcal{Q})$

From a high level, the proof of Theorem 2 is based on the fact that $\mathcal{T}_w^{<i}(\mathcal{Q})$ and $\mathcal{T}_r^i(\mathcal{Q})$ cannot be both small for all epochs i. To see why this must be intuitively true consider distribution $\mathcal{Q}_i$ over query sequences where the last `read` operation to the 0-th index is replaced with an index chosen uniformly at random from $\mathcal{U}^i$ (remember $\mathcal{U}^i$ are the indices of the array entries that are overwritten by `write` operations in epoch i). Since each `write` operation overwrites a distinct entry with a uniformly chosen b-bit string, a sufficiently large number of bits that were encoded by `write` operations in epoch i must be retrieved by the `read` operation. There are only three ways that these bits can be retrieved by the `read` operation. The first way is to probe cells that were last overwritten by any `write` operation of epoch i which corresponds to $\mathcal{T}_r^i(\mathcal{Q}_i)$. Another way is to probe cells

that were last overwritten by operations that occurred after epoch i; that is, in any epoch $1 \leq j < i$. However, the total number of bits encoded by operations in epochs $1 \leq j < i$ is upper bounded by the number of probes performed in epochs $1 \leq j < i$ to cells that were last overwritten by an operation in epoch i, which corresponds to $\mathcal{T}_w^{<i}(\mathcal{Q}_i)$. The final way to retrieve information from the `write` operations of epoch i is to encode information in the client's storage of c bits. However, if we consider the case when the number of entries overwritten in epoch i, ℓ_i, is significantly larger than c, then the client's storage is too small to encode any significantly large amount of information compared to the total number of `write` operations of epoch i. As a result, the total combined size of $\mathcal{T}_w^{<i}(\mathcal{Q}_i)$ and $\mathcal{T}_r^{i}(\mathcal{Q}_i)$ or, better, a function of the two quantities, can be lower bounded. However, recall that we wish to lower bound the values when processing the random sequence $\mathcal{Q}$ and not $\mathcal{Q}_i$. The only difference between $\mathcal{Q}$ and $\mathcal{Q}_i$ is the index of the `read` operation performed at the end. By computational differential privacy, any random event that can be verified by a PPT adversary cannot occur with significantly different probabilities when **DS** processes $\mathcal{Q}$ as opposed to $\mathcal{Q}_i$. Since the sets of cell probes can easily be computed in polynomial time, a lower bound on the sum of $|\mathcal{T}_w^{<i}(\mathcal{Q}_i)| + |\mathcal{T}_r^{i}(\mathcal{Q}_i)|$ also implies a lower bound on the $|\mathcal{T}_w^{<i}(\mathcal{Q})| + |\mathcal{T}_r^{i}(\mathcal{Q})|$ for a differentially private **DS**.

As explained above, the technical crux of the lower bound on $|\mathcal{T}_w^{<i}(\mathcal{Q}_i)| + |\mathcal{T}_r^{i}(\mathcal{Q}_i)|$ is an encoding argument that is captured by the following lemma that shows that a certain random variable $\mathcal{Z}_i(\mathcal{Q}(j))$ is "large" with probability at least 1/2. We say that an epoch i is *large* if $l_i \geq \max\{\sqrt{n}, c^2/b\}$.

Lemma 1. *Assume that* **DS** *has failure probability at most* $1/3$. *Then, for any large epoch* i*, there exists an index* $\mathsf{idx} \in \{1, \ldots, n-1\}$ *such that*

$$\Pr[\mathcal{Z}_i(\mathcal{Q}(\mathsf{idx})) \geq b/8)] \geq 1/2$$

where $\mathcal{Z}_i(\mathcal{Q}(\mathsf{idx}))$ *is*

$$\frac{1}{\ell_i}\left(|\mathcal{T}_w^{<i}(\mathcal{Q}(\mathsf{idx}))|w + \log\binom{t_w s_{i-1}}{|\mathcal{T}_w^{<i}(\mathcal{Q}(\mathsf{idx}))|}\right) + \left(|\mathcal{T}_r^{i}(\mathcal{Q}(\mathsf{idx}))|w + \log\binom{t_r}{|\mathcal{T}_r^{i}(\mathcal{Q}(\mathsf{idx}))|}\right) + \frac{c}{\ell_i}.$$

The proof of Lemma 1 is found in Sect. 3.4. $\mathcal{Z}_i(\mathcal{Q}(\mathsf{idx}))$ can be viewed as the total average information that the `read`(idx) operation at the end of $\mathcal{Q}(\mathsf{idx})$ retrieves from the `write` operations of epoch i. Let us explain the meaning of each term of the value $\mathcal{Z}_i(\mathcal{Q}(\mathsf{idx}))$. The first term of $\mathcal{Z}_i(\mathcal{Q}(\mathsf{idx}))$ measures the average amount of information pertaining to each of the ℓ_i `write` operations of epoch i that are read by cell probes performed in epochs following epoch i. Each of the cell probes in $\mathcal{T}_w^{<i}(\mathcal{Q}(j))$ reads exactly w bits in a cell. In addition, the choice of which cell probes performed in epochs following epoch i actually belong to $\mathcal{T}_w^{<i}(\mathcal{Q}(j))$ also encodes some information. As there are s_{i-1} `write` operations epochs following epoch i, there are at most $t_w s_{i-1}$ cell probes and at most $\binom{t_w s_{i-1}}{|\mathcal{T}_w^{<i}(\mathcal{Q}(j))|}$ choices of the cells to probe leading to $\log\binom{t_w s_{i-1}}{|\mathcal{T}_w^{<i}(\mathcal{Q}(j))|}$ bits. Similarly, each probe in $\mathcal{T}_r^{i}(\mathcal{Q}(j))$ reads w bits in each cell and there are at most $\binom{t_r}{|\mathcal{T}_r^{i}(\mathcal{Q}(j))}$ choices of probes when performing `read`(j) that belong to $\mathcal{T}_r^{i}(\mathcal{Q}(j))$.

The last term of $\mathcal{Z}_i(\mathcal{Q}(j))$ considers the average amount of information for each of the ℓ_i operations in epoch i that are encoded in the client's storage of c bits. Now, observe that the expected total amount of information that need to be transferred if all ℓ_i possible `read` operations of $\mathcal{Q}_i$ are performed is $\ell_i \cdot b$ bits. As a result, by taking idx to be the index which requires the most bit transferred of the ℓ_i indices overwritten in epoch i leads us to the above lemma. A formal proof of these ideas is presented in Sect. 3.4.

3.2 Using Differential Privacy

We note that Lemma 1 does not suffice to prove that $t_w + t_r = \omega(1)$. Typically, chronogram lower bounds will find a single sequence that forces a large amount of information transfer from all epochs simultaneously. Instead, Lemma 1 states, that for each epoch, there exists some sequence that forces a large information transfer that the sequences are possibly different for each epoch. In fact, without assuming privacy about a data structure, there can be no single sequence that requires large information from many epochs as there are trivial $\Theta(1)$ data structures that solve the array maintenance problem without any privacy guarantees. As Lemma 1 does not assume privacy for **DS**, we will need to incorporate the fact that **DS** is differentially private to achieve a statement that there exists a single sequence that forces large information transfer from many epochs simultaneously.

Let us now assume that **DS** provides differential privacy against computational adversaries with parameters $\epsilon = O(1)$ and $0 \leq \delta \leq 1/3$. For any fixed sequence Q, we consider any probabilistic event $\mathcal{E}(Q)$ over the randomness of the choice of the random string $\mathcal{R}$ such that there exists a probabilistic polynomial time algorithm that can verify $\mathcal{E}(Q)$ being true or false. Then, computational differential privacy implies that, for any fixed sequence Q_1 and Q_2 that differ in exactly one operation, $\Pr[\mathcal{E}(Q_1) \text{ is true}] \leq e^\epsilon \Pr[\mathcal{E}(Q_2) \text{ is true}] + \delta$. In particular, we can consider the event $\mathcal{E}(Q) =$ "$\mathcal{Z}_i(Q) \geq b/8''$. Note that $\mathcal{Z}_i(Q)$ can be computed by any computational adversary by simply assigning each cell probe performed by **DS** over Q into one of $\{\mathcal{T}_w^{<i}(Q)\}_{i=1,\ldots,k}$ or $\{\mathcal{T}_r^i(Q)\}_{i=1,\ldots,k}$ where assigning a cell probe depends only on the last time the cell was overwritten and the current operation of Q. As a result, we know that for any two fixed sequences Q_1 and Q_2 that differ in exactly one operation, then

$$\Pr[\mathcal{Z}_i(Q_1) \geq b/8] \leq e^\epsilon \Pr[\mathcal{Z}_i(Q_2) \geq b/8] + \delta.$$

Note that $\mathcal{Q}$ and $\mathcal{Q}(\mathsf{idx})$ only differ in the input index to the `read` operation at the end of the sequence. We use this fact to prove the following lemma that $\mathcal{Z}_i(\mathcal{Q})$ cannot differ significantly from $\mathcal{Z}_i(\mathcal{Q}(\mathsf{idx}))$ for any idx.

Lemma 2. *Let* **DS** *be an* (ϵ, δ)*-differentially private RAM and let* i *be a large epoch. Then,*

$$\Pr[\mathcal{Z}_i(\mathcal{Q}) \geq b/8] \geq 1/(6e^\epsilon).$$

Proof. By Lemma 1, there exists an index idx such that $\Pr[\mathcal{Z}_i(\mathcal{Q}(\mathsf{idx})) \geq b/8] \geq 1/2$. We define $Q(\mathsf{idx}, B_1, \ldots, B_m) = (\texttt{write}(1, B_1), \ldots, \texttt{write}(m, B_m), \texttt{read}(\mathsf{idx}))$. Then,

$$\Pr[\mathcal{Z}_i(\mathcal{Q}) \geq b/8] = \sum_{m=n/2}^{n-1} \sum_{B_1,\ldots,B_m \in \{0,1\}^b} \frac{1}{m2^{bm}} \Pr[\mathcal{Z}_i(Q(0, B_1, \ldots, B_m)) \geq b/8]$$

$$\geq \sum_{m=n/2}^{n-1} \sum_{B_1,\ldots,B_m \in \{0,1\}^b} \frac{1}{m2^{bm}} \left(\frac{\Pr[\mathcal{Z}_i(Q(\mathsf{idx}, B_1, \ldots, B_m)) \geq b/8] - \delta}{e^{\epsilon}} \right)$$

$$= \frac{\Pr[\mathcal{Z}_i(\mathcal{Q}(\mathsf{idx})) \geq b/8] - \delta}{e^{\epsilon}} \geq \frac{1/2 - 1/3}{e^{\epsilon}} = \frac{1}{6e^{\epsilon}}.$$

3.3 Completing the Proof of Theorem 2

Lemma 2 resembles the typical desired statement for data structure lower bounds as it guarantees existence of a distribution of query sequences, $\mathcal{Q}$, that forces a large amount of information transfer from all epochs in expectation.

Recall that we consider epoch i consisting of $\ell_i = 2^i$ `write` operation for a total of $s_i = 2^{i+1} - 2$ `write` operation in epochs $1, \ldots, i$. We refer the reader to Fig. 2 for a visual reminder of our epoch construction. Using Lemma 2, we will show that $\Omega(b/w)$ bits must be transferred from the *majority* of *large* epochs. In particular, we focus on epochs i for which the number of blocks, ℓ_i, written by the `write` operations is much larger than the number of blocks that can be stored in client's memory, c/b. For otherwise, the blocks written by the `write` operations in epoch i may be entirely encoded into client's storage of c bits and thus no information from epoch i is required to be transferred by cell probes of future operations. Concretely, we say that an epoch is *large* if $\ell_i \geq \max\{\sqrt{n}, c^2/b\}$ and note that, by our definition of epochs, we have $\hat{k} := \Theta(\log(nb/c))$ large epochs. We will show that for many large epochs $\Omega(b/w)$ bits must be transferred by cell probes of either `write` operations of future epochs or the `read` operation.

To achieve our lower bound, we will analyze the expectation of $\mathcal{Z}_i(\mathcal{Q})$ based on our epoch construction. We will provide a high-level overview of the steps of our analysis in this paragraph before performing a formal analysis. Recall that t_w and t_r are an upper bound on the expected amortized number of cells probed per `write` and `read` operation for any sequence. For the majority of epochs i, we cannot expect the `read` operation of $\mathcal{Q}$ to probe more than $t_r/\hat{k}$ cells containing information about the `write` operations of epoch i. This provides an upper bound on $|\mathcal{T}_r^i(\mathcal{Q})|$ for the majority of epochs. We want a similar upper bound on the value of $|\mathcal{T}_w^{<i}(\mathcal{Q})|$. Recall that this number corresponds to the number of probes performed by `write` operations that read cells that encode information about the `write` operations of epoch i. Our argument will critically use the fact that the sequence $\mathcal{Q}$ is chosen at random. Recall that $\mathcal{Q}$ is chosen to have m `write` operations where m is chosen uniformly at random from $\{n/2, n/2+1, \ldots, n-1\}$. The data structure **DS** is unable to predict the point in time when the `read` operation will occur. Instead, the best that **DS** can achieve is to prepare for

all possible epoch configurations. Since there are $\hat{k}$ epochs with size at least $\max\{\sqrt{n}, c^2\}$, each update should be only able to encode $\frac{t_w \cdot w}{\hat{k}}$ about each of these epochs. As a result, we can prove the majority of epochs cannot have very large values of $|\mathcal{T}_w^{<i}(\mathcal{Q})|$ in expectation.

As the two bounds above hold for the majority of epochs, we can show there exists at least one large epoch i such that both the values of $|\mathcal{T}_w^{<i}(\mathcal{Q})|$ and $|\mathcal{T}_r^{i}(\mathcal{Q})|$ are small. In particular, we show the following:

Lemma 3. *There exists a large epoch i for which $\mathsf{E}\left[|\mathcal{T}_w^{<i}(\mathcal{Q})|/\ell_i\right] = O\left(t_w/\log(nb/c)\right)$ and $\mathsf{E}\left[|\mathcal{T}_r^{i}(\mathcal{Q})|\right] = O\left(t_r/\log(nb/c)\right)$.*

Proof. The lemma is derived from the following two statements:

1. There exists $\hat{k}/2+1$ large epochs i such that $\mathsf{E}[|\mathcal{T}_w^{<i}(\mathcal{Q})|/\ell_i] = O(t_w/\log(nb/c))$.
2. There exists $\hat{k}/2+1$ large epochs i such that $\mathsf{E}[|\mathcal{T}_r^{i}(\mathcal{Q})|] = O(t_r/\log(nb/c))$.

Since there are only $\hat{k}$ large epochs, there must exist at least one large epoch where both inequalities hold. We now show the two statements are true.

Let us pick epoch i uniformly at random amongst the $\hat{k}$ large epochs and fix the random string $\mathcal{R}$ as well as the $n-1$ block values $\mathcal{B}_1, \ldots, \mathcal{B}_{n-1}$. We now fix a cell probe `probe` of the execution of **DS** over the `write` operations `write`$(1, \mathcal{B}_1), \ldots,$ `write`$(n-1, \mathcal{B}_{n-1})$ and consider the probability that `probe` contributes to $\mathcal{T}_w^{<i}(\mathcal{Q})$ from which we derive a bound on $\mathsf{E}[|\mathcal{T}_w^{<i}(\mathcal{Q})|/\ell_i]$. Note that, having fixed $\mathcal{R}$ and the values $\mathcal{B}_j$'s, the probability space is over the choice of m from $\{n/2, n/2+1, \ldots, n-1\}$ and of i. We denote p_r as the index of the `write` operation in $\mathcal{U}$ when `probe` is performed. The value p_w is denoted as the index of the `write` operation in $\mathcal{U}$ when the cell of `probe` was last overwritten. Using p_r and p_w, we can attempt to upper bound the probability that the `probe` belongs to $\mathcal{T}_w^{<i}(\mathcal{Q})$. First, let e be the smallest integer such that $p_r - p_w \leq s_e$. Note that `probe` cannot contribute to $\mathcal{T}_w^{<j}(\mathcal{Q})$ for any epoch $j \leq e-1$, since there are only s_j operations between the beginning of epoch k and the `read` operation. Since $s_j \leq s_{e-1} < p_r - p_w$, either the read operation has to occur after the `read` operation or the last operation to overwrite the cell probe occurs before the j-th epoch. We remind the reader that the exact locations of epochs is determined by m. The boundary denoting the end of epoch j has to occur after p_w and before p_r meaning there are at most s_e choices from the position of the `read` operation such that this cell probe contributes to $\mathcal{T}_w^{<j}(\mathcal{Q})$. There are $n/2$ choices for m, so the probability is at most $2s_e/n$. We now compute

$$\mathsf{E}[|\mathcal{T}_w^{<i}(\mathcal{Q})|/\ell_i] = \frac{1}{\hat{k}} \sum_{j:\ell_j \geq \max\{\sqrt{n}, c^2\}} \mathsf{E}[|\mathcal{T}_w^{<j}(\mathcal{Q})|/\ell_j].$$

The `probe` only contributes to epochs $j \geq e$. Note, there are at most (in expectation) $t_w \cdot (n-1)$ cell probes performed when processing the `write` operations of $\mathcal{Q}$. By linearity of expectation,

$$\sum_{j:\ell_j \geq \max\{\sqrt{n}, c^2/b\}} \mathsf{E}\left[\frac{|\mathcal{T}_w^{<j}(\mathcal{Q})|}{\ell_j}\right] \leq t_w \cdot n \sum_{j \geq e} \frac{2 \cdot s_e}{n \cdot l_j} \leq 2t_w \cdot \left(\frac{s_e}{l_e} + \frac{s_e}{l_{e+1}} + \ldots\right) \leq 4t_w.$$

As a result, there exists $\hat{k}/2+1$ fixed epochs i such that their expectation over the m is at most $12t_w$.

We know that $\sum_i \mathsf{E}[|\mathcal{T}_r^i(\mathcal{Q})|] \leq t_r$. Therefore, there exists $\hat{k}/2+1$ large epochs i such that $\mathsf{E}[|\mathcal{T}_r^i(\mathcal{Q})|] \leq 3t_r$ completing the proof.

We can now achieve our goal of proving Theorem 2 that gives a lower bound on the sum $t_w + t_r$ by plugging the inequalities in Lemma 3 into the expectation of $\mathcal{Z}_i(\mathcal{Q})$ and then using the bound from Lemma 2.

Proof (Theorem 2). First, we analyze the expectation of $\mathcal{Z}_i(\mathcal{Q})$. Note that, for every x, y, $\log\binom{y}{x} = O(x\log(y/x))$. Moreover, for every y, $x\log(y/x)$ is a convex function over x, so we can write the $\mathsf{E}[x\log(y/x)] \leq \mathsf{E}[x]\log(y/\mathsf{E}[x])$ where the expectation is over the choice of x. We now apply this observation to $\mathsf{E}[\mathcal{Z}_i(\mathcal{Q})]$

$$O\left(\frac{\mathsf{E}[|\mathcal{T}_w^{<i}(\mathcal{Q})|]}{\ell_i}\left(w + \log\frac{t_w s_{i-1}}{\mathsf{E}[|\mathcal{T}_w^{<i}(\mathcal{Q})|]}\right) + \mathsf{E}[|\mathcal{T}_r^i(\mathcal{Q})|]\left(w + \log\frac{t_r}{\mathsf{E}[|\mathcal{T}_r^i(\mathcal{Q})|]}\right) + \frac{c}{\ell_i}\right).$$

By Lemma 2, we know that $\mathsf{E}[\mathcal{Z}_i(\mathcal{Q})] = \Omega(b)$. We now pick our epoch i as the one chosen by Lemma 3 and plug in the inequalities to get

$$\frac{t_w}{\log(nb/c)}\left(w + \log\frac{t_w s_{i-1}}{\ell_i t_w/\log(nb/c)}\right) + \frac{t_r}{\log(nb/c)}\left(w + \log\frac{t_r}{t_r/\log(nb/c)}\right) = \Omega(b).$$

Here we have used the fact that epoch i is large and thus $\frac{c}{\ell_i} = O(b)$, since $\ell_i \geq c^2/b$. Also, note that $s_{i-1} = \Theta(\ell_i)$. Therefore, we can simplify and get that $t_w + t_r = \Omega\left((b/(w + \log\log n))\log(nb/c)\right)$. If we assume that $w = \Omega(\log\log n)$, we can simplify and get the following result $t_w + t_r = \Omega\left((b/w)\log(nb/c)\right)$ which completes the proof.

Therefore, the lower bound of $t_w + t_r$ described in Theorem 2 can be entirely derived from Lemma 1. It remains to prove Lemma 1, which we do next.

3.4 An Encoding Argument Using $\mathcal{T}_w^{<i}(\mathcal{Q})$ and $\mathcal{T}_r^i(\mathcal{Q})$

In this section, we prove Lemma 1. We first give a high level description of the proof. The main idea involves converting any **DS** that solves the array maintenance problem into a one-way communication problem between two parties, for which we have a lower bound on the number of bits that must be sent.

Specifically, we consider the case in which for a fixed epoch $i \in \{1, \ldots, k\}$ and for a sequence drawn according to $\mathcal{Q}$, one party, Alice, receives the m values $\mathcal{B}_1, \ldots, \mathcal{B}_m$ and a random string $\mathcal{R}$ and the other party, Bob, receives the same random string $\mathcal{R}$ as well as $m - \ell_i$ values; that is, all of $\mathcal{B}_1, \ldots, \mathcal{B}_m$ except for the ℓ_i values updated in epoch i of sequence Q. The goal of the protocol is to let Bob obtain the missing ℓ_i values.

As the ℓ_i b-bit values are generated uniformly and independently at random, Alice's input has $\ell_i \cdot b$ bits of entropy conditioned on Bob's input and $\mathcal{R}$. By Shannon's Source Coding Theorem, any protocol for the above problem must have expected communication of at least $\ell_i \cdot b$ bits. We show that if Lemma 1 does not hold, then Shannon's Theorem is contradicted by giving an encoding constructed by simulating **DS** that beats Shannon's bound.

Recall that, for any idx, $\mathcal{Q}(\mathsf{idx})$ is constructed by picking m uniformly at random from $\{n/2, n/2+1, \ldots, n-1\}$ and constructing the sequence of m updates $\mathcal{U} = \mathtt{write}(1, \mathcal{B}_1), \ldots, \mathtt{write}(m, \mathcal{B}_m)$ where each $\mathcal{B}_1, \ldots, \mathcal{B}_m$ is drawn independently and uniformly at random from $\{0,1\}^b$. We also denote by $\mathcal{U}^i$ the set of `write` of epoch i, for $i = 1, \ldots, k$.

Consider the following protocol. Alice and Bob locally execute all `write` operations in epochs $k, k-1, \ldots, i+1$ using the random string $\mathcal{R}$. Bob keeps a snapshot snap_B of **DS** at this point. Now Bob can learn each of the ℓ_i values $\mathcal{B}_{\mathsf{idx}}$ for $\mathsf{idx} \in \mathcal{U}^i$ written during epoch i, by simulating epoch j, for $j = i, i-1, \ldots, 1$ followed by the $\mathtt{read}(\mathsf{idx})$ operation. To do this, Bob uses the snapshot snap_B, that gives the state of **DS** before any `write` operations of epoch i are executed, and the following information that can be transferred by Alice.

1. The c bits of client storage of **DS** after the `write` operations of epoch i have been processed.
2. The location and contents of the cells that are probed by the `write` operations of epochs $j = i-1, \ldots, 1$ and by the $\mathtt{read}(\mathsf{idx})$ operation.

Given this information as well as the random string $\mathcal{R}$, Bob can simulate **DS** by starting from snap_B and executing all the `write` operations of $\mathcal{U}$ occurring after epoch i as well as $\mathtt{read}(\mathsf{idx})$ and thus recover $\mathcal{B}_{\mathsf{idx}}$. To encode all ℓ_i block values updated in epoch i, Alice and Bob can repeat the simulation of the `read` operation ℓ_i times with idx ranging over the set of the ℓ_i indices that are updated in epoch i. The number of bits that need to be transferred to Bob by Alice depends on the following three values:

1. The number of bits of the client storage, c.
2. The number of probes performed in epochs $j = i-1, \ldots, 1$ to cells last written in epoch i.
3. The number of probes performed by the ℓ_i `read` operations to the ℓ_i indices updated in epoch i.

By Shannon's source coding theorem, we have a lower bound on the number of bits that can be transferred and, consequently, a lower bound on the number of probes performed by **DS**. The rough description above only works for **DS** that never fails but it only requires some small changes to work for failure probability 1/3. In particular, Alice can indicate the indices idx for which **DS** fails to return $\mathcal{B}_{\mathsf{idx}}$ and explicitly transfer the b bits of $\mathcal{B}_{\mathsf{idx}}$ to Bob in addition to the above protocol. We now present the formal proof of Lemma 1.

Proof (Lemma 1). In our proof, we consider **DS** that have failure probability at most 1/512. Note that any **DS** with failure probability 1/3 implies the existence of a **DS** with failure probability 1/512 as one can execute **DS** a constant number of times with independently chosen randomness and return the most popular result to answer any `read` operation. In fact, proving a lower bound for **DS** with failure probability $1/512$ implies any **DS** with failure probability that is a constant greater than $1/2$ using the above method.

Recall that $\mathcal{U}^i$ denotes the set of all ℓ_i indices that are updated by `write` operations in epoch i. It suffices to prove that $\Pr[\sum_{\mathsf{idx} \in \mathcal{U}^i} \mathcal{Z}_i(\mathcal{Q}(\mathsf{idx})) \geq \ell_i b/8] \geq 1/2$. Since $\mathcal{U}^i$ contains ℓ_i indices, the previous statement implies that there must exist some $\mathsf{idx} \in \mathcal{U}^i$ such that $\Pr[\mathcal{Z}_i(\mathcal{Q}(\mathsf{idx})) \geq b/8] \geq 1/2$ which would complete the proof. Therefore, towards a contradiction, assume that $\Pr[\sum_{\mathsf{idx} \in \mathcal{U}^i} \mathcal{Z}_i(\mathcal{Q}(\mathsf{idx})) \geq \ell_i b/8] < 1/2$ for some data structure **DS** that solves the array maintenance problem with failure probability at most $1/512$. We will present an encoding of $\ell_i \cdot b$ random bits from Alice and Bob using **DS** that uses strictly less than $\ell_i \cdot b$ bits in expectation contradicting Shannon's source coding theorem.

In computing the encoding, Alice receives the m b-bit random values used by the sequence of `write` operations, $\mathcal{U}$, and a random string $\mathcal{R}$.

Alice's Encoding

1. Alice executes **DS** on the sequence $\mathcal{U}$ using the random string $\mathcal{R}$ up to the final `write` operation of epoch i. The content of the c bits of client storage after epoch i is completed are added to the encoding.
2. Alice then executes the remaining s_{i-1} `write` operations of $\mathcal{U}$ of epochs $i-1, i-2, \ldots, 1$. While processing these `write` operations, Alice records the subset $\mathcal{T}_w^{<i}(\mathcal{U})$ of probes to cells that were last written in epoch i as well as their contents. This information is encoded as follows. First the size $|\mathcal{T}_w^{<i}(\mathcal{U})|$ (at most $\log(t_w \cdot s_{i-1})$ bits) is added to the encoding. Then Alice adds an encoding of which $|\mathcal{T}_w^{<i}(\mathcal{U})|$ probes of the at most $t_w(n-1)$ probes over the entire sequence belong to $\mathcal{T}_w^{<i}$ (this costs $\log \binom{t_w \cdot (n-1)}{|\mathcal{T}_w^{<i}(\mathcal{U})|}$ bits). Finally, for each such probe, w bits are added to the encoding to specify the content of the cell probed (for additional $|\mathcal{T}_w^{<i}(\mathcal{U})| \cdot w$ bits).
3. Alice stores a snapshot snap_A of the **DS** after processing all `write` operations of $\mathcal{U}$. Alice will use this snapshot to simulate the `read` operations for the ℓ_i entries written in epoch i.
4. For each of the ℓ_i indices $\mathsf{idx} \in \mathcal{U}^i$, Alice executes $\mathtt{read}(\mathsf{idx})$ on snap_A. Let F be the number of $\mathtt{read}(\mathsf{idx})$ operations that return a wrong value (that is, they return a value other than $\mathcal{B}_{\mathsf{idx}}$). Alice adds the value F to the encoding costing $\log n$ bits and an encoding of the subset of the F *failing* indices costing $\log \binom{\ell_i}{F}$ expected bits.
5. For each of the F failing indices $\mathsf{idx} \in \mathcal{U}^i$, Alice adds $\mathcal{B}_{\mathsf{idx}}$ to the encoding costing a total of $F \cdot b$ bits.
6. For each non-failing index $\mathsf{idx} \in \mathcal{U}^i$ (that is, for which $\mathtt{read}(\mathsf{idx})$ executed on snap_A with $\mathcal{R}$ successfully returns $\mathcal{B}_{\mathsf{idx}}$), Alice adds the subset of probes performed during $\mathtt{read}(\mathsf{idx})$ to the cells in $\mathcal{T}_r^i(\mathcal{Q}(\mathsf{idx}))$ (these are the cells last written in epoch i) as well as their content to the encoding. This costs w bits for each cell in $\mathcal{T}_r^i(\mathcal{Q}(\mathsf{idx}))$ as well as $\log \binom{t_r}{|\mathcal{T}_r^i(\mathcal{Q}(\mathsf{idx}))|}$ bits to encode the subset $\mathcal{T}_r^i(\mathcal{Q}(\mathsf{idx}))$ of the at most t_r probes in $\mathtt{read}(\mathsf{idx})$.

7. Alice checks whether either of $\sum_{\mathsf{idx} \in \mathcal{U}^i} \mathcal{Z}_i(\mathcal{Q}(\mathsf{idx})) > \ell_i b/8$ or $F > \ell_i/64$. If either are true, Alice stops and returns an encoding consisting of a 0 bit followed by $\ell_i \cdot b$ bits of the ℓ_i blocks updated by the `write` operations in $\mathcal{U}^i$.
8. Otherwise, when both $\sum_{\mathsf{idx} \in \mathcal{U}^i} \mathcal{Z}_i(\mathcal{Q}(\mathsf{idx})) \leq \ell_i b/8$ and $F \leq \ell_i/64$, Alice prepends a 1 bit to the encoding computed in Steps 1–6 and returns it.

In decoding the message sent by Alice, Bob receives the random string $\mathcal{R}$ but does not receive the entirety of $\mathcal{U}$. Instead, Bob receives $\mathcal{U}$ except all block values that are updated in epoch i.

Bob's Decoding

1. Bob checks the first bit of Alice's encoding. If the first bit is a 0, then Bob parses the next $\ell_i \cdot b$ bits as the contents of the ℓ_i block values updated in $\mathcal{U}$ completing the decoding.
2. If the encoding begins with a 1, Bob will execute the `write` operations in epochs $j = k, k-1, \ldots, i-1$ using random string $\mathcal{R}$. Note that this is straightforward as Bob received all the needed values as input and the indices of the `write` are fixed.
3. Note that Bob does not have access to the updated array entries of epoch i, and thus will skip it.
4. Next, Bob sets the client storage as specified in the encoding and starts simulating the `write` operations for epochs $j = i-1, \ldots, 1$. As long as the `write` operations do not require probing a cell that was last written in epoch i, Bob can simulate **DS** in the exact same way as done by Alice to compute the encoding (note Bob has access to the same $\mathcal{R}$). Whenever **DS** requires probing a cell last written in epoch i, Bob will use the encoding of the cell contents found in the encoding to continue simulation. As a result, Bob can simulate all `write` operations of $\mathcal{U}$ after epoch i identically to Alice. Bob will now take a (partial) snapshot of **DS** including all cell locations and contents that Bob is aware of.
5. Next, Bob obtains F, the number of failing `read`, from the encoding along with the indices $\mathsf{idx} \in \mathcal{U}^i$ where `read`(idx) fails to return $\mathcal{B}_{\mathsf{idx}}$. For each of these F indices, Bob obtains the corresponding value $\mathcal{B}_{\mathsf{idx}}$ from the encoding.
6. For the remaining $\ell_i - F$ indices $\mathsf{idx} \in \mathcal{U}^i$ such that `read`(idx) returns $\mathcal{B}_{\mathsf{idx}}$, Bob will execute `read`$(\mathsf{idx})$ on the snapshot of **DS**. From the encoding, Bob knows which of the (at most, in expectation) t_r probes performed by `read`(idx) are to cells last written in epoch i. Bob simulates `read`(idx) on his snapshot with $\mathcal{R}$ using the cell contents encoded by Alice to retrieve $\mathcal{B}_{\mathsf{idx}}$.

Analysis. It remains to analyze the expected length of Alice's encoding. Recall that we know from Shannon's source coding theorem that Alice's encoding has to be at least $\ell_i \cdot b$ bits long in expectation.

There are two cases to consider. In the first case, when the first bit is a 0, the encoding will be $1 + \ell_i \cdot b$ bits long. Let us now consider the case in which the first bit is 1 and thus $F \leq l_i/64$ and $\sum_{\mathsf{idx} \in \mathcal{U}^i} \mathcal{Z}_i(\mathcal{Q}(\mathsf{idx})) \leq \ell_i b/8$. The encoding of the failed indices has expected length

$$\log n + \mathsf{E}\left[\log\binom{\ell_i}{F}\right] + b\cdot\mathsf{E}[F] \le \log n + \mathsf{E}\left[\log\binom{\ell_i}{\frac{\ell_i}{64}}\right] + b\cdot\frac{\ell_i}{64}$$
$$\le \log n + \frac{\ell_i}{64}\cdot(b+\log(64\cdot e)) \le \log n + \frac{9}{64}(\ell_i\cdot b).$$

The second inequality uses Stirling's approximation which states that $\binom{x}{y} \le (ex/y)^y$. We know Alice's encoding of client storage will always be c bits. We know the expected bits of encoding $\mathcal{T}_w^{<i}(\mathcal{U})$ is

$$\log(t_w\cdot s_{i-1}) + \mathsf{E}\left[\log\binom{t_w\cdot s_{i-1}}{|\mathcal{T}_w^{<i}(\mathcal{U})|} + |\mathcal{T}_w^{<i}(\mathcal{U})|\cdot w\right].$$

Note, $\log(t_w\cdot s_{i-1}) \le 2\log n$. Similarly, for all $\mathsf{idx}\in\mathcal{U}^i$ that successfully return $\mathcal{B}_{\mathsf{idx}}$, we know that the encoding requires

$$\mathsf{E}\left[|\mathcal{T}_r^i(\mathcal{Q}(\mathsf{idx}))|w + \log\binom{t_r}{|\mathcal{T}_r^i(\mathcal{Q}(\mathsf{idx}))|}\right].$$

Note that

$$\mathsf{E}\left[\log\binom{t_w\cdot s_{i-1}}{|\mathcal{T}_w^{<i}(\mathcal{U})|} + |\mathcal{T}_w^{<i}(\mathcal{U})|w + \sum_{\mathsf{idx}\in\mathcal{U}^i}|\mathcal{T}_r^i(\mathcal{Q}(\mathsf{idx}))|w + \log\binom{t_r}{|\mathcal{T}_r^i(\mathcal{Q}(\mathsf{idx}))|}\right] + c$$
$$\le \sum_{\mathsf{idx}\in\mathcal{U}^i}\mathcal{Z}_i(\mathcal{Q}(\mathsf{idx})) \le \frac{1}{8}(\ell_i\cdot b).$$

Summing over all parts of the encoding, we get that

$$3\log n + \frac{9}{64}(\ell_i\cdot b) + \sum_{\mathsf{idx}\in\mathcal{U}^i}\mathcal{Z}_i(\mathcal{Q}(\mathsf{idx})) \le 3\log n + \frac{17}{64}(\ell_i\cdot b).$$

Finally, we compute the probabilities that Alice places a 0 or a 1 as the first bit of the encoding. By Markov's inequality, $\Pr[F \ge \ell_i/64] \le 1/8$ and we know that $\Pr[\sum_{\mathsf{idx}\in\mathcal{U}^i}\mathcal{Z}_i(\mathcal{Q}(\mathsf{idx})) \ge \ell_i b/8] < 1/2$ by our initial assumption towards a contradiction. As a result, we know that $\Pr[F \ge \ell_i/64$ or $\sum_{\mathsf{idx}\in\mathcal{U}^i}\mathcal{Z}_i(\mathcal{Q}(\mathsf{idx})) \ge b/8] < 5/8$. So, Alice's expected encoding size is at most

$$1 + 3\log n + \frac{5}{8}(\ell_i\cdot b) + \frac{17}{64}(\ell_i\cdot b) < \ell_i\cdot b$$

contradicting Shannon's source coding theorem when $\ell_i \ge \sqrt{n}$.

4 Main Result

In Sect. 3, we presented a lower bound on the sum of t_w, the worst case bandwidth for `write` operations, and t_r, the worst case expected amortized bandwidth for `read` operations that implies that $\max\{t_w, t_r\} = \Omega((b/w)\log(nb/c))$. However, this lower bound does not preclude the existence of a differentially private RAM with $t_w = \Theta((b/w)\log(nb/c))$ and $t_r = o((b/w)\log(nb/c))$ or $t_r = \Theta((b/w)\log(nb/c))$ and $t_w = o((b/w)\log(nb/c))$. In this section, we strenghten our lower bound and prove the following two statements, for (ϵ, δ)-differentially private RAM, for any constant ϵ and $\delta \leq 1/3$,

1. If $t_w = o((b/w)\log(nb/c))$, then $t_r = \omega((b/w)\log(nb/c))$;
2. If $t_r = o((b/w)\log(nb/c))$, then $t_w = \omega((b/w)\log(nb/c))$.

Therefore, since $\max\{t_w, t_r\} = O((b/w)\log(nb/c))$, then it must be the case that both $t_w = \Theta((b/w)\log(nb/c))$ and $t_r = \Theta((b/w)\log(nb/c))$ showing that imbalanced running times for `write` and `read` operations cannot improve the asymptotic efficiency of differentially private RAM constructions.

To achieve these tradeoffs, we revisit our epoch construction of Sect. 3. Let us, first, focus our attention on the first statement where we show that $t_r = \omega((b/w)\log(nb/c))$ when $t_w = o((b/w)\log(nb/c))$. Recall that we constructed epochs that grew exponentially by a factor of 2 for a total of $\Theta(\log n)$ epochs and the number of large epochs (that is with at least $\max\{\sqrt{n}, c^2/b\}$ `write` operations) is $\Theta(\log(nb/c))$. In the techniques used in Sect. 3, we are only able to show that $\Omega(b/w)$ cells must be probed from the majority of the epochs. As there are only $\Theta(\log(nb/c))$ large epochs, there is no hope for us to prove a stronger lower bound t_r with this epoch construction.

Instead, we will use a different epoch construction that is suitable for the scenario where we know that t_w is small. In Lemma 3, we show that, on average, for any large epoch i any `write` operations of future epochs $j \in \{1, \ldots, i-1\}$ can only encode $O(t_w w/\hat{k})$ bits about epoch i where $\hat{k}$ is the number of large epochs. It is important that future `write` operations cannot encode a lot of information about epoch i as it forces the final `read` operation to read sufficient information from epoch i directly. However, as we are assuming that t_w is already small, we may increase the number of future operations after epoch i while simultaneously ensuring that future epochs cannot encode too much information about epoch i. With this observation, we hope that we can increase the number of total epochs which allows us to prove $\omega((b/w)\log(nb/c))$ lower bounds on t_r as desired. We now materialize these ideas in the next section.

4.1 First Epoch Construction

In this section, we consider an epoch construction where epochs grow by the rate r every r epochs, with $r = \omega(1)$ and $r = O(\log n)$. That is, the first r epochs will each have r `write` operations; the next r epochs will each have r^2 `write` operations; the next r epochs will each have r^3 `write` operations and so forth. See Fig. 4 for a diagram of this epoch construction. Once again, we define ℓ_i to be the number of `write` operations of the i-th epoch and s_i to be the total number of `write` operations of epochs $1, \ldots, i$. We note that, by writing $\ell_i = r^f$ for some $f \geq 1$, we have

$$s_{i-1} \leq r \cdot (r^f + r^{f-1} + \ldots + r) \leq 2r \cdot \ell_i.$$

The new epoch construction will potentially give us, for each epoch, r times more future operations in comparison to the epoch construction of Sect. 3. On the other hand, we note that the number of large epochs (that is, with at least $\max\{\sqrt{n}, c^2/b\}$ `write` operations) is $\hat{k} = \Theta(r \log_r(nb/(rc^2))) = \Theta(r \log_r(nb/c))$, which is larger by a super-constant factor of $\Theta(r/\log r)$ than the number of epochs in the construction of Sect. 3. As a result, this epoch construction matches exactly the requirements that we wanted there to be more epochs which are required to be read by the `read` operation while only sacrificing that there are more future `write` operations for any epoch i. We now present a generalization of Lemma 3 which can be applied for the new epoch constructions that are introduced here and in Sect. 4.2.

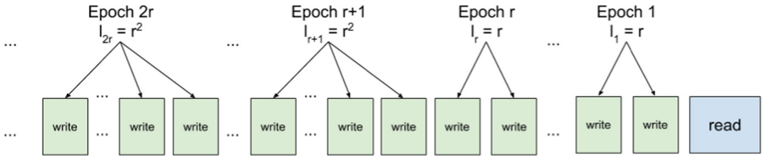

Fig. 4. Diagram of epoch construction of Sect. 4.1.

Lemma 4. *There exists a large epoch i such that* $\mathsf{E}[|\mathcal{T}_w^{<i}(\mathcal{Q})|] = O\left(\frac{t_w}{\hat{k}} \cdot \max_e \sum_{j \geq e} \frac{s_e}{l_j}\right)$ *and* $\mathsf{E}[|\mathcal{T}_r^{i}(\mathcal{Q})|] = O\left(\frac{t_r}{\hat{k}}\right)$.

Proof. Using the same ideas of Lemma 3, we will show that there exists $\hat{k}/2+1$ epochs that satisfy the first statement and $\hat{k}/2+1$ epochs that satisfy the second statement. As a result, there exists at least one epoch satisfying both statements.

Pick an epoch i uniformly at random from the $\hat{k}$ large epochs. Fix $\mathcal{B}_1, \ldots, \mathcal{B}_{n-1}$ and $\mathcal{R}$ arbitrarily. We will prove an upper bound on $\mathsf{E}[|\mathcal{T}_w^{<i}(\mathcal{Q})|]$ over the randomness of the location of the `read` operation and the randomly chosen i. As a result, the expectation's upper bound will hold over any distribution of $\mathcal{B}_1, \ldots, \mathcal{B}_{n-1}$ and $\mathcal{R}$. Fix any cell probe performed by **DS** when processing `write`$(1, \mathcal{B}_1), \ldots,$ `write`$(n-1, \mathcal{B}_{n-1})$ and suppose that probe occurs when processing the p_r-th `write` operation to a cell that was last written by the p_w-th `write` operation. Once again, we pick the smallest e such that $p_r - p_w \leq s_e$. Consider any epoch j where $j \leq e-1$. Note that there are only s_j operations between the `read` operation and the beginning of epoch j. But, since $j \leq e-1$, we know that $s_j \leq s_{e-1} < p_r - p_w$ meaning that either the probe occurs after the `read` operation or the cell was last written before epoch j. When we fix the location of the `read` operation, we fix the epoch construction. As the boundary of the j-th epoch must occur after the p_w-th operation and before the p_r-th operation, there are at most s_e good locations for the `read` out of $n/2$ total locations. For any $j \geq e$, this cell probe has probability $2s_e/n$ of contributing to $\mathcal{T}_w^{<j}(\mathcal{Q})$. Therefore, by linearity of expectation over the $(n-1)t_w$ expected cell probes:

$$\mathsf{E}\left[\frac{|\mathcal{T}_w^{<i}(\mathcal{Q})|}{\ell_i}\right] = \frac{1}{\hat{k}} \sum_{\text{epoch } j \text{ is large}} \mathsf{E}\left[\frac{|\mathcal{T}_w^{<j}(\mathcal{Q})|}{\ell_j}\right] \leq \frac{t_w}{\hat{k}}\left(\max_e \sum_{j \geq e} \frac{2s_e}{\ell_j}\right).$$

Therefore, there exists $\hat{k}/2+1$ fixed epochs i such that $\mathsf{E}[|\mathcal{T}_w^{<i}(\mathcal{Q})|/\ell_i]$ over the choice of the `read` location is at most 3 times the above bound.

As $\sum_i \mathsf{E}[|\mathcal{T}_r^i(\mathcal{Q})|] \leq t_r$, there exists $\hat{k}/2+1$ epochs i where $\mathsf{E}[|\mathcal{T}_r^i(\mathcal{Q})|] \leq 3t_r$ completing the proof.

Theorem 3. *Let* **DS** *be an* (ϵ, δ)*-differentially private RAM for* n b*-bit array entries implemented over* w*-bit cells. Assuming that* $\epsilon = O(1)$ *and* $0 \leq \delta \leq 1/3$*,* **DS** *has failure probability at most* $1/3$ *and* $w = \Omega(\log\log n)$*, then*

$$t_w = o((b/w)\log(nb/c)) \implies t_r = \omega((b/w)\log(nb/c)).$$

Proof. Recall we get the following inequality by applying convexity to the inequality of Lemma 2 and noting that $c/\ell_i = O(1)$ for our choices i:

$$\frac{\mathsf{E}[|\mathcal{T}_w^{<i}(\mathcal{Q})|]}{\ell_i}\left(w + \log\frac{t_w s_{i-1}}{\mathsf{E}[|\mathcal{T}_w^{<i}(\mathcal{Q})|]}\right) + \mathsf{E}[|\mathcal{T}_r^i(\mathcal{Q})|]\left(w + \log\frac{t_r}{\mathsf{E}[|\mathcal{T}_r^i(\mathcal{Q})|]}\right) = \Omega(b).$$

By applying Lemma 4, we get that $\mathsf{E}[|\mathcal{T}_r^i(\mathcal{Q})|] = O(t_r \log r/(r\log(nb/c)))$ and $\mathsf{E}[|\mathcal{T}_w^{<i}(\mathcal{Q})|/\ell_i] = O(t_w \log r/\log(nb/c))$ since

$$\left(\frac{s_j}{\ell_j} + \frac{s_j}{\ell_{j+1}} + \ldots\right) \leq 2r\sum_{j\geq 0}\frac{1}{r^j} = O(r).$$

Plugging into the inequality above and assuming that $w = \Omega(\log\log n)$,

$$t_w + (t_r/r) = \Omega((b/w)\log(nb/c)/\log r) \implies t_r = \Omega((b/w)\log(nb/c)r/\log r)$$

as $t_w = o(b/w)\log(nb/c)$. Since $r/\log r = \omega(1)$, we complete the proof.

4.2 Second Epoch Construction

In this section, we deal with the opposite scenario when we assume that $t_r = o((b/w)\log(nb/c))$ and want to show that $t_w = \omega((b/w)\log(nb/c))$. The same intuition from the previous section can be used for this situation: to show that t_w has to be very large, we will need to require that for any epoch i, the total number of future `write` operations in epochs $j \in \{1,\ldots,i-1\}$ is small. If for any epoch i, the number of future `write` operations after epoch i is small and the `read` operation also cannot perform many cell probes into epoch i, then each future `write` operation must encode a large amount of information about epoch i which will be used by the `read` operation. As a result, we can prove a large lower bound on t_w.

Specifically, we consider an epoch construction in which the number of `write` operations in an epoch is larger by a super-constant factor $r = O(\log n)$ compared with the number in the previous epoch. So, the first epoch will have r `write` operations, the second epoch will have r^2 `write` operations, etc. So,

$$s_{i-1} = \ell_{i-1} + \ell_{i-2} + \ldots \leq \frac{1}{r}(\ell_i + \ell_{i-1} + \ldots) \leq 2\ell_i/r.$$

As a result, the number of future operations is $\Theta(1/r)$ times smaller than the epoch construction of Sect. 3. The number of large epochs, that is with at least $\max\{\sqrt{n}, c^2/b\}$ `write`, is $\hat{k} = \Theta(\log_r nb/c)$.

Theorem 4. *Let* **DS** *be an* (ϵ, δ)*-differentially private RAM for* n b*-bit array entries implemented over* w*-bit cells. Assuming that* $\epsilon = O(1)$ *and* $0 \leq \delta \leq 1/3$*,* **DS** *has failure probability at most* $1/3$ *and* $w = \Omega(\log \log n)$*, then*

$$t_r = o((b/w)\log(nb/c)) \implies t_w = \omega((b/w)\log(nb/c)).$$

Proof. By applying Lemma 4, we get that $\mathsf{E}[|\mathcal{T}_r^i(\mathcal{Q})|] = O(t_r \log r/\log(nb/c))$ and $\mathsf{E}[|\mathcal{T}_w^{<i}(\mathcal{Q})|/\ell_i] = O(t_w \log r/r \log(nb/c))$ since

$$\left(\frac{s_j}{\ell_j} + \frac{s_j}{\ell_{j+1}} + \ldots\right) \leq 2\sum_{j \geq 0} \frac{1}{r^j} = O(1).$$

Plugging into the inequality of Lemma 2 after applying convexity and noting that $w = \Omega(\log \log n)$ and that, for large epochs, $c/\ell_i = O(1)$, we obtain

$$(t_w/r) + t_r = \Omega((b/w)\log(nb/c)/\log r) \implies t_w = \Omega((b/w)\log(nb/c)r/\log r)$$

since $t_r = o((b/w)\log(nb/c))$. Noting that $r/\log r = \omega(1)$ completes our proof.

5 Discussion

We now discuss three extensions that follow from our lower bound techniques.

Our techniques only enforce the requirements of differential privacy for a single `read` operation. Therefore, our lower bounds would also apply *differentially private-read RAMs* where differential privacy is guaranteed only for sequences of operations that differ in exactly one `read` operation. This might be important in scenarios where the indices of `write` operations are not sensitive (or may be public) but only the indices of `read` operations need to be protected. Once again, this weakening of security does not suffice to get around the $\Omega(\log(nb/c))$ bandwidth overhead lower bounds.

The lower bounds of Sects. 3 and 4 hold for $\delta \leq 1/3$. Most practical scenarios require that δ must be negligible in n, so the above results suffice. For theoretical exploration, we note that our results can be extended to any constant $\delta < 1$. In particular, for any $\rho < 1$, by picking a sufficiently large enough constant C, we can prove that $\Pr[\mathcal{Z}_i(\mathcal{Q}(\mathsf{idx})) \geq b/C] \geq \rho$ which is a variation of Lemma 1.

By using $\rho > \delta$ we can extend Lemma 2 and prove that for all epochs i, $\Pr[\mathcal{Z}_i(\mathcal{Q}) \geq b/C] \geq (\rho - \delta)/e^{\epsilon}$. This will suffice to extend Theorem 2 to any $\delta < 1$. The main result of Sect. 4 can be similarly extended.

Finally, our lower bound assumes **DS** has worst time case cost on update operations, but may be extended to worst case amortized update costs. In particular, we only apply Lemma 1 to epochs whose sum of probed cells by update operations is not too much larger than expected. By an averaging argument, it can be shown that a constant fraction of all epochs satisfy this property.

In this work, we show that the $\Omega(\log(nb/c))$ bandwidth overhead lower bound for the array maintenance problem with obliviousness extends to the weaker notion of differential privacy with reasonable privacy budgets of $\epsilon = O(1)$ and $\delta \leq 1/3$. The result is surprising as differentially private RAM provides significantly weaker privacy. This leads to the following natural open question: Does there exist a natural, weaker notion of privacy that enables $o(\log(nb/c))$ bandwidth overhead for the array maintenance problem?

References

1. Asharov, G., Komargodski, I., Lin, W.-K., Nayak, K., Peserico, E., Shi, E.: OptORAMa: Optimal oblivious RAM. ePrint Report 2018/892
2. Boyle, E., Chung, K.-M., Pass, R.: Oblivious parallel RAM and applications. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016. LNCS, vol. 9563, pp. 175–204. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49099-0_7
3. Boyle, E., Naor, M.: Is there an oblivious RAM lower bound? In: ITCS 2016, pp. 357–368 (2016)
4. Cash, D., Grubbs, P., Perry, J., Ristenpart, T.: Leakage-abuse attacks against searchable encryption. In: CCS 2015, pp. 668–679 (2015)
5. Chan, T.-H.H., Guo, Y., Lin, W.-K., Shi, E.: Oblivious hashing revisited, and applications to asymptotically efficient ORAM and OPRAM. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10624, pp. 660–690. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_23
6. Chen, B., Lin, H., Tessaro, S.: Oblivious parallel RAM: improved efficiency and generic constructions. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016. LNCS, vol. 9563, pp. 205–234. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49099-0_8
7. Chung, K.-M., Liu, Z., Pass, R.: Statistically-secure ORAM with $\tilde{O}(\log^2 n)$ overhead. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8874, pp. 62–81. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45608-8_4
8. 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
9. 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
10. Dwork, C.: A firm foundation for private data analysis. Commun. ACM **54**(1), 86–95 (2011)

11. Dwork, C., McSherry, F., Nissim, K., Smith, A.: Calibrating noise to sensitivity in private data analysis. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 265–284. Springer, Heidelberg (2006). https://doi.org/10.1007/11681878_14
12. Dwork, C., Roth, A., et al.: The algorithmic foundations of differential privacy. Found. Trends Theor. Comput. Sci. **9**, 211–407 (2014)
13. Fredman, M., Saks, M.: The cell probe complexity of dynamic data structures. In: STOC 1989, pp. 345–354 (1989)
14. Garg, S., Lu, S., Ostrovsky, R., Scafuro, A.: Garbled RAM from one-way functions. In: STOC 2015, pp. 449–458 (2015)
15. 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
16. Goldreich, O.: Towards a theory of software protection and simulation by oblivious RAMs. In: STOC 1987, pp. 182–194 (1987)
17. Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious RAMs. JACM **43**(3), 431–473 (1996)
18. 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. LNCS, vol. 6756, pp. 576–587. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22012-8_46
19. Goodrich, M.T., Mitzenmacher, M., Ohrimenko, O., Tamassia, R.: Privacy-preserving group data access via stateless oblivious RAM simulation. In: SODA 2012, pp. 157–167 (2012)
20. Islam, M.S., Kuzu, M., Kantarcioglu, M.: Access pattern disclosure on searchable encryption: ramification, attack and mitigation. In: NDSS 2012 (2012)
21. Kushilevitz, E., Lu, S., Ostrovsky, R.: On the (in) security of hash-based oblivious RAM and a new balancing scheme. In: SODA 2012, pp. 143–156 (2012)
22. Larsen, K.G.: The cell probe complexity of dynamic range counting. In: STOC 2012, pp. 85–94 (2012)
23. Larsen, K.G., Nielsen, J.B.: Yes, there is an oblivious RAM lower bound!. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10992, pp. 523–542. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96881-0_18
24. Larsen, K.G., Weinstein, O., Yu, H.: Crossing the logarithmic barrier for dynamic boolean data structure lower bounds. In: STOC 2018, pp. 978–989 (2018)
25. Lu, S., Ostrovsky, R.: Black-Box parallel garbled RAM. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10402, pp. 66–92. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63715-0_3
26. Mironov, I., Pandey, O., Reingold, O., Vadhan, S.: Computational differential privacy. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 126–142. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03356-8_8
27. Patel, S., Persiano, G., Raykova, M., Yeo, K.: PanORAMa: oblivious RAM with logarithmic overhead. In: FOCS 2018, pp. 871–882 (2018)
28. 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
29. Pătraşcu, M.: Lower bound techniques for data structures. Ph.D. thesis. MIT (2008)
30. Pătraşcu, M., Demaine, E.D.: Logarithmic lower bounds in the cell-probe model. SIAM J. Comput. **35**(4), 932–963 (2006)

31. Stefanov, E., Shi, E., Song, D.: Towards practical oblivious RAM. arXiv:1106.3652 (2011)
32. Stefanov, E., et al.: Path ORAM: an extremely simple oblivious RAM protocol. In: CCS 2013, pp. 299–310 (2013)
33. Toledo, R.R., Danezis, G., Goldberg, I.: Lower-cost ϵ-private information retrieval. Proc. Priv. Enhancing Technol. **2016**(4), 184–201 (2016)
34. Wagh, S., Cuff, P., Mittal, P.: Root ORAM: a tunable differentially private oblivious RAM. arXiv:1601.03378 (2016)
35. Wang, X.S., et al.: Oblivious data structures. In: CCS 2014, pp. 215–226 (2014)
36. Weiss, M., Wichs, D.: Is there an Oblivious RAM lower bound for online reads? ePrint report 2018/619
37. Yao, A.C.-C.: Should tables be sorted? JACM **28**(3), 615–628 (1981)