Lower Bound Framework for Differentially Private and Oblivious Data Structures

Giuseppe Persiano[*] Kevin Yeo[†]

Abstract

In recent years, there has been significant work in studying data structures that provide privacy for the operations that are executed. These primitives aim to guarantee that observable access patterns to physical memory do not reveal substantial information about the queries and updates executed on the data structure. Multiple recent works, including Larsen and Nielsen [Crypto'18], Persiano and Yeo [Eurocrypt'19], Hubáček *et al.* [TCC'19] and Komargodski and Lin [Crypto'21], have shown that logarithmic overhead is required to support even basic RAM (array) operations for various privacy notions including obliviousness and differential privacy as well as different choices of sizes for RAM blocks b and memory cells ω.

We continue along this line of work and present the first logarithmic lower bounds for differentially private RAMs (DPRAMs) that apply regardless of the sizes of blocks b and cells ω. This is the first logarithmic lower bounds for DPRAMs when blocks are significantly smaller than cells, that is $b \ll \omega$. Furthermore, we present new logarithmic lower bounds for differentially private variants of classical data structure problems including sets, predecessor (successor) and disjoint sets (union-find) for which sub-logarithmic plaintext constructions are known. All our lower bounds extend to the multiple non-colluding servers setting.

We also address an unfortunate issue with this rich line of work where the lower bound techniques are difficult to use and require customization for each new result. To make the techniques more accessible, we generalize our proofs into a framework that reduces proving logarithmic lower bounds to showing that a specific problem satisfies two simple, minimal conditions. We show our framework is easy-to-use as all the lower bounds in our paper utilize the framework and hope our framework will spur more usage of these lower bound techniques.

1 Introduction

In this work, we will study *privacy-preserving data structures* in the setting where a client outsources the storage of data to one or more potentially untrusted servers (such as a cloud provider). Even though the client delegates the storage to the server, the client may need to perform operations on the outsourced data in an efficient manner. In terms of privacy, the client wishes to maintain the confidentiality of the outsourced data. A straightforward first attempt is for the client to encrypt all data locally before transferring to the server. While guaranteeing that the server cannot see the data in plaintext, this technique does not address the leakage of access patterns that the server observes when the client performs operations on the outsourced data. For example, the server may observe the exact memory locations that are retrieved or modified. Therefore, it is integral

[*] Università di Salerno and Google, giuper@gmail.com.

[†] Google and Columbia University, kwlyeo@google.com.

to protect the patterns of data access to also maintain privacy for the actions performed over the outsourced data.

Oblivious RAMs. Oblivious RAMs (ORAMs), introduced by Goldreich and Ostrovsky [GO96], are one cryptographic primitive that may be leveraged to hide access patterns. At a high level, ORAMs can be viewed as a data structure that enables maintenance of a dynamic array where the client either query or update any entry. The obliviousness privacy guarantee of ORAMs ensures that any adversary given two candidate equal-length operational sequences and observes the access pattern incurred by the execution of one of the sequences still cannot determine the identity of the executed operational sequence. In recent years, ORAMs have been studied extensively to try and determine the optimal overhead (see [GO96, GMOT12, KLO12, SvS+13, LO13, CLP14, RFK+15, BCP16, CLT16, DvF+16, PPRY18, GKW18] and references therein). For b-bit entries on machines with memory cell (word) size of ω bits, the best known constructions obtain logarithmic overhead $O((1+b/\omega)\cdot\log n)$ [AKL+20]. This ends up being optimal as it matches the lower bounds of $\Omega((b/\omega)\cdot\log n)$ by Larsen and Nielsen [LN18] and $\Omega(\log n/(1+\log(\omega/b)))$ by Komargodski and Lin [KL21] up to logarithmic factors in b and ω for all choices of b and ω. Due to their strong privacy guarantees, ORAMs have seen usage in many applications such as multi-party computation [WHC+14, BCP15, Ds17] and secure cloud storage systems [SS13, BNP+15].

Differentially Private RAMs. In various practical applications, the guarantees provided by obliviousness end up being unnecessarily strong. For example, we can consider the problem of privacy-preserving data analysis where the goal is to reveal statistics about a data set, but still maintain the privacy of each individual. An algorithm is considered *differentially private* if the probability distribution of the output of the algorithm for two data sets that differ in only one record will not differ significantly. Therefore, if the adversary observes the disclosure of the algorithm, it may not learn information about whether an individual was a member of the input data set. Consider the problem of privacy-preserving data analysis over a data set outsourced to an untrusted server. For any accesses to the data set, we could use an ORAM to completely hide any subset of records accessed from the data set. However, this may be stronger privacy than needed as the differentially private disclosure only provides privacy for individuals.

Instead, we turn to differentially private RAMs (DPRAMs) whose privacy guarantees align closer to the ones used in privacy-preserving data analysis. DPRAMs aim to provide privacy for individual operations, but may reveal information about a sequence consisting of many operations. In more detail, if an adversary receives two candidate equal-length operational sequences that differ in one operation and the access pattern incurred by the execution of one of the two sequences, the adversary should not be able to guess the identity of the executed sequence with too high probability. Due to the weaker guarantees, there is hope to obtain sub-logarithmic overhead smaller than ORAMs. For example, sub-logarithmic constructions have been shown for differentially private Turing machines, stacks and queues [KS21] whereas logarithmic overhead is required for their oblivious counterparts [JLN19, KS21]. Unfortunately, the $\Omega(b/\omega\cdot\log n)$ lower bound for DPRAMs by Persiano and Yeo [PY19] showed that this is impossible when $b=\Omega(\omega)$. However, no such lower bound is known when blocks are significantly smaller than cells, $b\ll\omega$, leading to the following question that was also posed as an open problem in [KL21]:

What is the optimal overhead for differentially private RAMs
for the setting when blocks are much smaller than cells, $b\ll\omega$?

We resolve this by proving a logarithmic lower bound for all choices of b and ω.

Framework for Cell Probe Lower Bounds. Starting from the seminal work of Larsen and Nielsen [LN18] that introduced the usage of cell probe techniques for oblivious RAMs, there has been a significant amount of work for proving cell probe lower bounds for various data structure problems and privacy guarantees. Previous works have considered lower bounds for different privacy notions beyond obliviousness and differential privacy including obliviousness without adversarial knowledge of operational boundaries [HKKS19], obliviousness in the multiple non-colluding server setting [LSY20] and searchable encryption leakage functions [PPY20]. Lower bounds have also been proven for other oblivious data structure problems beyond RAMs including stacks, queues, deques, heaps and search trees [JLN19] as well as near-neighbor search [LMWY20].

Unfortunately, the lower bounds end up being very technical and customized to each specific setting. To date, if one wished to prove lower bounds for a specific data structure with certain privacy guarantees, one would have to understand all the various techniques and modify them accordingly to obtain the desired lower bound. Ideally, we would like to encapsulate the re-usable portions of the proofs into a blackbox framework that enables future users to prove lower bounds by only modifying parts that need to be customized for the specific data structure problem and/or privacy notion. This leads us to the following natural question:

Is it possible to generalize the techniques into a framework
that enables easier lower bound proofs for future works?

To address this, we present a framework that reduces proving logarithmic lower bounds for privacy-preserving data structures to showing that the data structure problem and privacy notion satisfy two simple (and seemingly minimal) conditions. Furthermore, we show that our framework is widely applicable by proving logarithmic lower bounds for a whole set of new data structure problems for which sub-logarithmic upper bounds are known with no privacy guarantees.

1.1 Our Contributions

We summarize our results below. All our lower bounds are proven in the cell probe model where overhead refers to the required number of probes into server memory cells. If one restricts the server to be passive (i.e., may not perform any computation), then our results become communication lower bounds.

Differentially Private RAMs (DPRAMs). For our first result, we present new lower bounds for DPRAMs. Informally speaking, a RAM is (ϵ, δ)-*differentially private* if, for any two sequences of operations O_1 and O_2 that differ in one operation and for any PPT adversary $\mathcal{A}$, we have that $p_1^{\mathcal{A}} \leq e^{\epsilon} \cdot p_2^{\mathcal{A}} + \delta$, where $p_\eta^{\mathcal{A}}$ is the probability that $\mathcal{A}$ outputs 1 on input the transcript of the RAM executing sequence O_η. We look at in the setting where blocks are significantly smaller than the word size, $b \ll \omega$, and show that DPRAMs must still have logarithmic overhead regardless of the parameter settings for b and ω. In our work, we will prove the following theorem. Throughout this section, we ignore $O(\log\log\log n)$ factors to avoid being overburdensome. See Theorem 5 for a more precise statement.

Theorem 1 (Informal). *Any (ϵ, δ)-DP RAM for n b-bit entries with constant $\epsilon > 0$, sufficiently small, constant $\delta > 0$ and client storage of c bits has overhead:*

$$\Omega\left(\frac{\log(nb/c)}{1 + \log(\omega/b)}\right).$$

To interpret the lower bound, we note that our lower bound is the same as the one proved in [PY19] for DPRAMs in the case $b = \Theta(\omega)$. However, for the case when $b \ll \omega$, our lower bound ends up peaking a lot higher. For example, consider the case where $b = \Theta(1)$ and $\omega = \Theta(\log n)$. Then, our lower bound ends being $\tilde{\Omega}(\log n)$ while the lower bound in [PY19] becomes trivial at $\Omega(1)$. In other words, our result ends up proving logarithmic lower bounds for all reasonable choices of block and cell sizes $b = \log^{O(1)}(n)$ and $\omega = \log^{O(1)}(n)$. In such regimes, our lower bound is tight up to $O(\log\log n)$ factors with the best known ORAM constructions [AKL+20].

Additionally, we show that we can extend our lower bound to the multiple server setting improving previous multi-server ORAM lower bounds by Larsen *et al.* [LSY20]. These are the first logarithmic lower bounds for DPRAMs in the multi-server setting (regardless of the choice of b and ω).

General Framework. To make these techniques more accessible, we develop a framework that abstracts out the necessary properties of a cryptographic data structure for which logarithmic lower bounds may be obtained. We modularize the proof such that the lower bound techniques leverage properties of either the data structure problem or privacy in exactly two points. Then, we identify the two properties needed to prove logarithmic lower bounds:

1. *Large Information Retrieval*: For any data structure problem P, one must find a random sequence of n updates $\mathbf{U} = (\mathbf{u}_1, \ldots, \mathbf{u}_n)$ such that for any consecutive sequence of ℓ updates $\mathbf{u}_a, \ldots, \mathbf{u}_{a+\ell-1}$, there exists a set Q of $O(\ell)$ queries whose answers have high entropy with respect to updates $\mathbf{u}_a, \ldots, \mathbf{u}_{a+\ell-1}$. If we let $\mathbf{A}(\mathbf{U}, Q)$ be the answers of all queries $q \in Q$ immediately executed after $\mathbf{U}$, then we must have that the average contribution to the entropy for each of the $O(\ell)$ queries is at least $\Omega(v)$ bits:
$$H(\mathbf{A}(\mathbf{U}, Q) \mid \mathbf{u}_1, \ldots, \mathbf{u}_{a-1}, \mathbf{u}_{a+\ell}, \ldots, \mathbf{u}_n)/\ell = \Omega(v).$$

2. *Event Probability Transfer*: Consider the setting with $k \geq 1$ server(s) where at most one server is compromised by the adversary. Let $E_i(U, q)$ be any event that is observable by a PPT adversary that compromises the i-th server when executing the update sequence $\mathbf{U}$ from above and a query q. Furthermore, suppose that the probability of the event satisfies $\Pr[E_i(\mathbf{U}, q)] \geq \zeta/k$ for some constant $\zeta > 0$. Then, the same event must occur with similar probability for any other query q':
$$\Pr[E_i(\mathbf{U}, q')] = \Omega(\Pr[E_i(\mathbf{U}, q)]).$$

The first property requires that the data structure problem is "complex" enough to enable retrieving updates with queries. For example, this rules out contrived data structures whose queries may not return any information about updates. The second property acts as a proxy for leveraging the privacy guarantees. For any data structure problem and associated privacy guarantees that can satisfy the above two properties, we immediately get the following theorem (see Theorem 3 for a formal statement).

Theorem 2 (Informal)**.** *Let P be a data structure problem satisfying the above two properties with query outputs of b bits. Any data structure* DS *solving P using at most client storage of c bits must have overhead:*
$$\Omega\left(\frac{b}{v} \cdot \frac{\log(nb/c)}{1 + \log(\omega/b)}\right).$$

As a result, we believe that we have made the lower bound techniques more accessible as one can reduce the problem of proving logarithmic lower bounds to simply showing that the data structure problem satisfies the two properties above. Furthermore, we identify that a key metric is the ratio between the size of the query output b and the amount of information gained per query v.

New Data Structure Lower Bounds. We show that our framework is widely applicable by proving logarithmic lower bounds for many data structure problems where lower bounds are not known with respect to any privacy guarantees. In our applications, we target data structure problems where $o(\log n)$ upper bounds are known when no privacy guarantees are required. By plugging these data structure problems into our framework, we obtain $\tilde{\Omega}(\log n)$ lower bounds showing that the differentially private versions of these data structures inherently require more overhead compared to the non-private versions. In particular, we prove logarithmic lower bounds for the following data structure problems:

- *Set Membership*: In this problem, the data structure maintains a subset $S \subseteq [n]$. A query for $i \in [n]$ returns a bit indicating whether $i \in S$. This is a natural problem where the output is a single bit and the cell size ω is much larger. Without privacy, one can solve this problem using a bit vector of length n and answer queries in constant time. Using our framework, we show that DP versions would, instead, require $\Omega(\log(n/c)/\log \omega)$ overhead.

- *Predecessor and Successor*: Predecessor (successor) aim to maintain a subset $S \subseteq U$ of size at most n. A query for some $i \in U$ returns the largest (smallest) item in S that is no larger (smaller) than the query input i. Without privacy requirements, one can solve predecessor in $O(\log \log n)$ overhead using van Emde Boas trees [vEB75] when $|U| = n^{O(1)}$. When DP guarantees are required, we show that the overhead must be $\Omega(\log(n/c)/\log(\omega/\log n))$.

- *Disjoint Sets (Union-Find)*: Finally, we consider the disjoint sets data structure that maintains a set of sets over n elements. The union operation takes two elements and joins their corresponding sets. The find operation takes an element and returns a set representation of the input element. For any two elements in the same set, the find operation will return the same set representation. The classical algorithm achieves overhead $O(\alpha(n))$ where $\alpha(n)$ is the inverse Ackermann function that is essentially constant in all practical settings. We show that the DP version requires overhead $\Omega(\log(n/c)/\log(\omega/\log n))$.

One result of our new framework is that we can prove lower bounds for natural data structures that do not enable writing of random blocks of data. Most prior works [LN18, JLN19, PY19, HKKS19, LMWY20, LSY20, KL21] considered "key-value" data structures where the values could be b-bit random blocks to derive enough entropy for lower bounds. The above data structure problems do not enable storing random b-bit blocks, but our framework is still able to prove logarithmic lower bounds. Finally, our framework may handle other privacy guarantees besides differential privacy and obliviousness. For example, our framework may prove lower bounds for leakage functions common in searchable encryption extending [PPY20].

Separation Result for Oblivious Stacks (and Queues). Finally, we consider the generality of our framework. For example, one may question whether there exist data structures that do not satisfy our framework's two required properties, but could still have a logarithmic lower bound. We provide evidence that our framework is quite general and tight by studying stacks and queues, two data structures that do not satisfy the first condition of large information retrieval. For oblivious stacks and queues, Jacob, Larsen and Nielsen [JLN19] proved an $\Omega(b/\omega \cdot \log(nb/c))$ lower bound. For differentially private stacks and queues, Komargodski and Shi [KS21] showed an upper bound of $O((1 + b/\omega) \cdot \log\log n)$. The correct overhead is unknown for oblivious stacks and queues when $b \ll \omega$.

We present constructions of oblivious stacks and queues with $O(b/\omega \cdot \log(nb/c))$ amortized overhead. So, one may obtain sub-logarithmic overhead when $b \ll \omega$. If $b = O(1)$ and $\omega = \Theta(\log n)$, then our construction uses $O(1)$ amortized overhead. Furthermore, our result can obtain even sub-constant amortized times. When $b = O(1)$ and $\omega = \Theta(\log^2 n)$, our construction requires $O(1/\log n)$ overhead meaning that, on average, only one operation amongst $\Theta(\log n)$ operations require interacting with the server. To our knowledge, this is the first separation between an online oblivious data structure and ORAMs when $b \ll \omega$.

Re-framing this result with respect to our framework, it becomes clear that oblivious stacks and queues should not satisfy the properties of our framework. Therefore, we believe that if one can prove logarithmic lower bounds for a differentially private version of a data structure problem P for all choices of b and ω, then one should be able to do so using our framework by showing that P satisfies the two necessary properties.

1.2 Related Works

Balls-and-Bins Lower Bounds. The first logarithmic lower bounds were proven by Goldreich and Ostrovsky [GO96] of the form $\Omega((b/\omega) \cdot (\log n/\log c))$ where the client has storage of c bits. Boyle and Naor [BN16] pointed out that these lower bounds only existed in the balls-and-bins model with a non-encoding assumption on the underlying blocks. Lower bounds of the form $\Omega(b/\omega \cdot (\log n/\log c))$ for DPRAMs were proven in [PPY19]. Cash, Drucker and Hoover [CDH20] proved lower bounds showing that one-round ORAMs must have $\Omega(\sqrt{n})$ overhead or client storage in the balls-and-bins model. Lower bounds for PIR were proven in similar models including for public preprocessing [BIM04, PY22] and private preprocessing [CK20, CHK22, Yeo23].

Cell Probe Lower Bounds. The cell probe model is a computational model where only probes into memory are charged cost. Everything else such as computation or randomness generation can be done for free. Therefore, proving cell probe lower bounds is the holy grail as these lower bounds will apply to any realistic computational model. Although, proving cell probe lower bounds ends up being difficult for this reason as the highest static lower bounds are $\tilde{\Omega}(\log n)$ [PTW10] and the highest dynamic lower bounds are $\tilde{\Omega}(\log^2 n)$ [Lar12]. For privacy-preserving data structures, the first cell probe lower bounds were proven by Larsen and Nielsen [LN18] for ORAMs. Further works have proven lower bounds for other oblivious data structures [JLN19] and near-neighbor search [LMWY20]. Other works have also considered various privacy notions including differentially private RAMs [PY19], ORAMs where adversaries do not know the boundaries of operations [HKKS19], ORAMs with multiple servers [LSY20] and searchable encryption [PPY20].

Lower Bound Barriers. Boyle and Naor [BN16] showed that proving unconditional lower bounds for offline ORAMs (that is, all operations are provided at one time) would imply currently unknown circuit lower bounds. Extending this result, Weiss and Wichs [WW18] showed that lower bounds for read-only online ORAMs would result in new lower bounds for either locally decodable codes or circuits.

Constructions. As mentioned early, there has been a long line of work attempting to construct ORAMs efficiently such as [GO96, GMOT12, KLO12, SvS$^+$13, RFK$^+$15, DvF$^+$16, PPRY18, AKL$^+$20] as well as in various settings including statistical security [CLP14], multi-party computation [WHC$^+$14, Ds17], multiple non-colluding servers [LO13, GKW18] and parallel access [BCP16, CLT16] to list some examples. Beyond ORAMs, other works have considered construction oblivious variants of other data structures [WNL$^+$14, Shi20, JLS21]. Previous works also presented constructions for differentially private RAMs [WCM18, PPY19], search trees [CCMS19], Turing machines, stacks and queues [KS21].

2 Technical Overview

Reviewing the Information Transfer Tree. We start with the information transfer tree technique of Pătraşcu and Demaine [PD06] used first by Larsen and Nielsen [LN18] to prove ORAM cell probe lower bounds. Komargodski and Lin [KL21] extended the technique to enable proving logarithmic lower bounds for ORAMs even when blocks are smaller than cells ($b < \omega$). At a high level, the information transfer tree technique arranges n operations into a tree with arity $\chi \geq 2$ where each of the n operations are uniquely assigned to a leaf node based on the execution order. Each cell read is uniquely assigned to an internal node as the lowest common ancestor of the leaf nodes associated to the operation performing the cell read and the last operation to overwrite the read cell. For internal node v, the totality of information that is read by queries in the right subtree rooted at v from updates in the left subtree of v exists in the contents of cells in the set of probes assigned to v. For any subtree rooted at v with ℓ leaf nodes, the number of assigned probes is maximized at $\Omega(\ell)$ when right subtree reads blocks overwritten in the left subtree. As the adversary may also compute the information transfer tree, it must be that each internal node is assigned its maximum. Otherwise, the adversary can determine that the worst case sequence was not executed. Summing the worst case across all internal nodes obtains the lower bound.

Unfortunately, the information transfer technique seems to inherently require a strong privacy condition, like obliviousness, for sequences differing in $\tilde{\Omega}(n)$ operations as the worst-case sequences for each internal node differ drastically. This is incompatible with differential privacy as the privacy guarantees degrade exponentially in the number of different operations. We note that Patel *et al.* [PPY20] investigated weaker leakage guarantees for encrypted search using information transfer, but still leveraged privacy for sequences differing in $\tilde{\Omega}(n)$ operations.

Previous Chronogram Approach. To prove lower bounds for differentially private RAMs, Persiano and Yeo [PY19] adapted the chronogram (introduced by Fredman and Saks [FS89]). The chronogram considers hard sequences of $\Theta(n)$ updates followed by a single query. The n updates are divided into $K = \tilde{O}(\log n)$ epochs that decay exponentially by a factor of $r \geq 2$. Epochs are numbered in reverse order, so that the i-th epoch has r^i updates. The main idea is as follows. For any epoch i, the information stored about updates occuring in the i-th epoch must appear in updates following the i-th epoch. Since we chose epochs to decay exponentially, the total size of epochs $\{1, \ldots, i-1\}$ is strictly smaller than the i-th epoch. As a result, future update operations cannot encode all the information written in the i-th epoch as long as r is chosen sufficiently large. Consider the final query to randomly retrieve information from written in the i-th epoch. If the data structure answers queries correctly, then, intuitively, the query must directly probe cells last overwritten in the i-th epoch with high probability. Finally, differential privacy guarantees require that the query probes a similar number of cells last overwritten from all K epochs to hide the identity of the epoch from which information is retrieved. We highlight privacy is only needed for sequences differing in the final query.

The crux of the above technique is an efficient communication protocol built using a too-good-to-be-true data structure. In this communication game, Alice and Bob both receives updates in all but the i-th epoch. Alice also receives the answers to queries in the i-th epoch that it wishes to encode to Bob. To do this, Alice and Bob will jointly execute the data structure with Alice helping Bob to fill in the i-th epoch. For all updates before epoch i, Alice and Bob can individually execute the updates. Alice will execute the i-th epoch of updates and keep track of all cell writes. For updates in following epochs, Alice will record any reads to cells (both locations and contents) last overwritten in epoch i. Finally, Alice will also execute all queries relevant to the i-th epoch and record all reads to cell last overwritten in epoch i. The set of all cell locations and contents that are read during operations following epoch i are encoded to Bob. So, Bob executes the data structure identically to Alice and retrieves query outputs to the i-th epoch.

There are two key observations to complete the proof. First, the encoding of cells and locations of all updates following epoch i is too small to encode everything about epoch i. Therefore, the information needed to retrieve a query must be encoded in cells last overwritten in the i-th epoch. Since queries output b bits, one can use an averaging argument to show that $\Omega(b/\omega)$ cells must be probed by random queries to retrieve b bits of information from the i-th epoch.

A New Chronogram Approach for Small Blocks. Unfortunately, the above approach suffers from an b/ω factor that seems inherent in the specific communication protocol. When $b < \omega$, there is nothing ruling out the data structure from storing the answers for $\Theta(\omega/b)$ queries in a single cell.

Our paper introduces a more efficient communication protocol than the one in [PY19] to handle these settings. If our goal is to prove logarithmic lower bounds, then we must show that random queries must probe $\Omega(1)$ cell last overwritten in epoch i regardless of the choices of b and ω. This is impossible if we rely on trying to encode the contents of cells probed by a query since cell sizes ω are larger than the output of the query b. Instead, we make the observation that the outputs of queries are actually smaller than contents of cells. In other words, it is more efficient for Alice to simply encode the answers to queries instead of encoding the contents of even a single cell probed by a query. However, Alice cannot simply encode the answers of all queries to Bob as this would not enable deriving any meaningful lower bound on query time. So, we also need another method to further compress Alice's encoding. The second idea is for Alice to identify queries that do not need to be encoded at all. For example, consider a query that does not probe any cell last overwritten by the i-th epoch. These queries may be executed correctly by Bob for free without any additional information from Alice. However, the frequency of these free queries is unclear. If no free queries exist, we would obtain a trivial encoding of simply sending all the query's outputs. In fact, a contrived data structure could simply force every query retrieving updates from the i-th epoch to simply probe a cell last overwritten in the i-th epoch to guarantee that there are no free queries at all. By increasing the update overhead by a single probe, each update in the i-th epoch can write to one additional cell that will be read by the corresponding subsequent query ensuring no free queries.

Taking a closer look, the above approach by a contrived data structure only succeeds because the epoch structure is fixed ahead of time. Instead, we consider a randomized epoch structure that cannot be leveraged by the data structure. To do this, we pick a random number of updates from $\{n/2, \ldots, n\}$ followed by a single query. The structure of $K = \tilde{O}(\log n)$ epochs is built over the final $n/2$ updates. Consider any data structure that probes at most $K/100$ cells during queries. As the epochs are randomly placed, we can show that if the data structure can answer a query regarding a block written in the i-th epoch then it will have to probe a cell last overwritten in the

i-th epoch with some constant probability, of approximately $1/100$. In other words, we just showed that around $(99/100)$-fraction of queries that retrieve information from the i-th epoch end up being free queries in Alice's encoding. As a result, we obtain a very efficient communication protocol that allows us to prove that queries must probe $\Omega(1)$ cells from all K epochs.

We note that the lower bound in [PY19] also used randomized epoch structures, but did so to remove $\log \log n$ factors from the lower bound. Their work would still obtain an $\Omega(\log n / \log \log n)$ lower bound without random epochs. In our work, we leverage random query locations in a vastly different way to prove the existence of many free queries. Without this, we cannot prove anything more than a trivial $\Omega(1)$ lower bound.

Comparison with [PT07]. We note that Pătraşcu and Tarniţă [PT07] also studied the setting of proving chronogram lower bounds when blocks were smaller than memory cells. However, they only considered data structures without any privacy requirements. Instead, they had to leverage properties of the underlying data structure problems. One can view our work as an extension of [PT07] where we prove chronogram lower bounds for easier data structure problems, but heavily rely on the privacy requirements of the data structure. In fact, all problems studied in our paper are known to have sub-logarithmic upper bounds without privacy requirements. As a result, we had to develop new techniques (especially in the efficient communication protocol) to be able to utilize differential privacy to prove lower bounds for these easier data structure problems.

3 Lower Bound Model

We prove our lower bounds in a variant of the cell probe model that were introduced by Larsen and Nielsen [LN18]. At a high level, the cell probe model only charges data structures for probes into memory cells. All other operations are free of charge (such as computation and randomness generation). To enable lower bounds for cryptographic data structures, we use the cell probe model that adapts the setting to multiple parties representing the client and $k \geq 1$ server(s). We assume that the client has at most c bits of storage. Each server's storage consists of memory cells (words) of ω bits. In this variant of the cell probe model, the only operation that is charged cost is to probe a cell in any of the server's memory. In our model, all accesses into client memory are free of charge. Additionally, we assume there exists an arbitrarily long, but finite, random string $\mathcal{R}$ that is available to both parties without any cost to access. One can view $\mathcal{R}$ as a random oracle, so our lower bounds apply even if one assumes random oracles exists.

In our work, we will consider dynamic data structure problems. By dynamic, we refer to the fact that the data structure enables operations that allow its users to update the information stored by the data structure. Furthermore, dynamic data structures are allowed to update its memory representation during each operation. We present a formal definition of dynamic data structures below:

Definition 1 (Dynamic Data Structure Problem). *A dynamic data structure consists of the tuple (U_u, U_q) where U_u is the universe of update operations and U_q is the universe of query operations. The error probability is at most α if for every sequence of updates $u_1, \ldots, u_n \in (U_u)^n$ and every query $q \in U_q$, the probability that the query $q(u_1, \ldots, u_n)$ produces the wrong answer is at most α.*

Next, we consider the view of the adversarial server(s) in this model. In particular, each of the $k \geq 1$ servers will receive a transcript consisting of everything that each server observes while processing operations that are executed by the client. For any sequence of operations $O \in$

$(U_u \cup U_q)^{|O|}$, we denote by $\mathbb{V}_i(O)$ the view of the i-th server when processing the operational sequence O. The transcript $\mathbb{V}_i(O)$ will consist of the contents of all memory cells stored on the i-th server as well as sequences of probes to cells that occur for each of the operations in the sequence O. We note that $\mathbb{V}_i(O)$ also contains information denoting the boundaries of when each operation starts and ends[1]. If the adversary compromises the i-th server, the adversary will receive $\mathbb{V}_i(O)$. We use $\mathcal{T}_{\mathsf{DS}}(O)$ to denote the entire transcript seen by the adversary for all compromised servers. Note that our definition assumes that the adversarial server(s) are honest-but-curious. As we are proving lower bounds, assuming that the adversary is weaker makes our result stronger as our lower bounds immediately also apply to more stronger adversaries such as those that are malicious.

Using the above definition, one can now formulate privacy notions for data structures. For example, obliviousness guarantees that any efficient adversary $\mathcal{A}$ should not be able to distinguish between $\mathbb{V}_i(O_1)$ and $\mathbb{V}_i(O_2)$ for any two equal-length sequences O_1 and O_2. In our work, we will consider two weaker notions: differential privacy and privacy with respect to leakage functions. As our framework does not make assumptions on any specific privacy notion, we delay formal defintions of these notions until Section 5.

4 Framework for Lower Bounds

In this section, we present a formal framework for proving lower bounds. In particular, we will only assume certain properties of the data structure problem (that we will describe later) and then we show that for any problems that satisfy these properties, one can immediately utilize our framework to prove lower bounds. Later, we will show that one can utilize our framework for many settings with different privacy guarantees and/or data structure functionalities.

Consider a data structure problem $P = (U_u, U_q)$. For any sequence of U of `update` operations and for any `query` operation $q \in U_q$, we denote by $A(U, q)$ the *correct answer* to q when it is executed following the `update` operations in U. We abuse notation and, for a sequence $Q = (q_1, \ldots, q_\ell)$ of queries, we denote by $A(U, Q)$ the sequence of the *correct* answers for queries $q_i \in Q$ obtained by executing each query directly after the update sequence U. We re-iterate that this set consists of all the correct answers and not the answers returned by a potentially randomized data structure with non-zero error probabilities. We will abuse notation and use $\mathbf{A}(\mathbf{U}, Q)$ for distribution $\mathbf{U}$ over update sequences to denote the distribution over the sequences of correct answers with respect to a update sequence distributed according $\mathbf{U}$. When $\mathbf{U}$ and Q are clear from context, we will drop the arguments and simply use $\mathbf{A}$.

We are now ready to formally define the required properties.

Definition 2 (Large Information Retrieval)**.** *We say that a data structure problem P has the* Large Information Retrieval *property with parameter v if there exists a distribution $\mathbf{U} = (\mathbf{u}_1, \ldots, \mathbf{u}_n)$ over sequences of n update operations such that for any subsequence $(\mathbf{u}_a, \ldots, \mathbf{u}_{a+\ell-1})$ of $\ell \geq \sqrt{n}$ update operations, there exists a sequence Q of length $\ell \leq |Q| \leq c \cdot \ell$, for some constant $c \geq 1$, such that*

$$H(\mathbf{A}(\mathbf{U}, Q) \mid \mathbf{u}_1, \ldots, \mathbf{u}_{a-1}, \mathbf{u}_{a+\ell}, \ldots, \mathbf{u}_n) \geq \ell \cdot v.$$

Definition 3 (Event Probability Transfer)**.** *Consider a data structure* DS *for the problem P. For any update sequence U and query q, let $E(U, q)$ be some event that can be checked whether to have*

[1] We note that Hubáček *et al.* [HKKS19] proved a logarithmic lower bound for ORAMs even when the adversary does not learn operational boundaries. We leave it as future work to adapt their techniques to work with our proof.

occurred by a PPT adversary such that $\Pr[E(U,q)] \geq \zeta$ *for some constant* $\zeta > 0$. *We say that* P *has the* Event Probability Transfer *if for every* DS *and for any two queries* q *and* q', *it must be that*

$$\Pr[E(U,q')] = \Omega(\Pr[E(U,q)])$$

where the probability is over the internal randomness of DS.

Next, we present the main theorem of our framework.

Theorem 3. *Consider a data structure problem* P *that allows* `update` *and* `query` *operations such that query outputs are* b *bits and* $b = n^{O(1)}$.[2] *Let* DS *be any data structure for* P *with expected update and query overhead* t_u *and* t_q *respectively, client storage* c *and error probability* $\alpha \leq v/(b \log^2 n)$ *in the cell probe model with* $\omega \geq 1$ *cell size. If* P *enjoys the Large Information Retrieval property with parameter* v *and Event Probability Transfer property then*

$$t_u + t_q = \Omega\left(\frac{v}{b} \cdot \frac{\log(nv/c)}{1 + \log((\omega + \log\log n)/b)}\right).$$

We dedicate the remainder of this section to proving this theorem. Later, we will show how to apply our framework to prove lower bounds in various settings.

Discussion about b and v. In the above theorem, b is the number of bits to describe the output of each query. On the other hand, v is the amount of information that is retrieved about the random updates with each query. In general, we know that $v \leq b$ as we cannot learn more information that the query output's size. Prior works have made the assumption that $b = v$ such as for array maintenance. By generalizing this, we illuminate the importance of this ratio for lower bounds in cryptographic data structures. In later sections when we prove lower bounds for specific problems, we will convert natural problems to artificial variants with the goal of maximizing the ratio v/b to prove higher lower bounds.

We point out that this b/v factor is distinctly different from the b/ω factor that appears in prior lower bounds. The b/v factor characterizes the average information retrieved per bit in the query output. In contrast, the b/ω factor characterizes the number (or fraction) of cells needed to represent the answer of a single query. For the case when cell size is larger than the query output $\omega > b$, our lower bound is better than the previous one of $\Omega(b/\omega \log(nb/c))$ [PY19] as it only loses $1/(1 + \log((\omega + \log\log n)/b)$ factor.

Comparison with [KL21]. We note that our lower bound is slightly lower than the one proved by Komargodski and Lin [KL21]. They proved a lower bound of the form $\Omega(\log(nb/c)/\log(\omega/b))$ but, to our knowledge, may be only applied to strong oblivious guarantees. On the other hand, we prove a lower bound of the form $\Omega(\log(nb/c)/\log((\omega + \log\log n)/b))$, but it is applicable to a wider range of possibly weaker privacy guarantees and data structure functionalities. We note the gap is very small and only exists in very restricted settings. When $\omega = \Omega(\log\log n)$, both lower bounds are asymptotically identical. Furthermore, if $b = \Omega(\omega)$, we can use the original $\Omega(b/\omega \cdot \log(nb/c))$ lower bounds such as [LN18]. Therefore, a gap between the lower bounds exists only when $\omega = o(\log\log n)$ and $b = o(\omega)$. It is not hard to see that the gap is at most $O(\log\log\log n)$.

Discussion about Error Probability. We note that one can obtain a slightly stronger theorem for constant error probability α if one is willing to make additional assumptions about the data

[2] For most natural problems, the output size is $b = O(\log n)$. For generality, we picked the largest upper bound as possible for b without affecting our lower bound.

structure DS. In particular, if one assumes that $v = \Theta(b)$, then one can prove lower bounds that hold also for data structures that err with constant probability. For convenience and the ability to handle general data structures, we consider weaker error probabilities of $O(1/\log^2 n)$. This is still much larger than the negligible error required for cryptographic primitives.

Weaker Large Information Retrieval. In our definitions, we assumed that every subsequence of at least $\sqrt{n}$ updates enabled a query sequence whose outputs had high entropy. A weaker definition would have sufficed for our lower bound where only a certain fraction of the subsequences admit this property (such as if a random subsequence satisfied this property with constant probability). However, we chose the stronger definition as it was sufficient for all applications of our lower bound framework and enabled simpler proofs.

4.1 An Efficient Communication Protocol

In this section, we show that a data structure for any problem P with error probability at most α emits a public coin one-way communication protocol for the problem where Alice wishes to efficiently encode the correct answers for all queries in a query set to Bob. In particular, this protocol efficiently encodes the output of queries even if the query output of b bits is significantly smaller than the cell size ω. We describe the problem below:

Communication Problem. Let $U = (u_1, \ldots, u_m)$ be a sequence of update operations. In the communication problem, Alice and Bob will receive the same sub-sequence of update operations $U' = (u_1, \ldots, u_{a-1}, u_{a+\ell}, \ldots, u_m)$ where the consecutive ℓ operations $u_a, \ldots, u_{a+\ell-1}$ are omitted, along with a set of queries Q and a random string $\mathcal{R}$. Additionally, Alice will receive sequence $A(U, Q)$; that is, the set of correct answers for all $q \in Q$ where each query is executed immediately after U. The goal of Alice will be to encode $A(U, Q)$ to Bob. In particular, Alice's encoding will have to account for the fact that Bob is missing ℓ update operations while ensuring Bob receives the correct answers.

Random Variables. Next, we denote some additional random variables that will be used to bound the total communication of our protocol. In particular, these variables will measure the number of probes by future updates and queries into the group of updates that are missing in Bob's input.

- $\mathbf{X}_{u}^{\geq a+\ell}$ denotes the number of probes perform by the update operations $(u_{a+\ell}, \ldots, u_m)$ into a cell last overwritten by an update in the missing group $(u_a, \ldots, u_{a+\ell-1})$.
- $\mathbf{X}_Q$ denotes the number of probes performed by all queries $q \in Q$ into a cell last overwritten by an update in the missing group $(u_a, \ldots, u_{a+\ell-1})$.
- $\mathbf{T}_{u}^{\geq a+\ell}$ denotes the total number of probes performed by all update operations starting from and including $u_{a+\ell}$.

Lemma 1. *If there exists a data structure* DS *for problem* P *that has error probability* $0 < \alpha < 1$, *then there is a public coin one-way communication protocol solving the above problem using expected communication at most*

$$\mathbb{E}\left[\mathbf{X}_{u}^{\geq a+\ell}\right]\left(\omega + \log \frac{t_u(m-a-\ell+1)}{\mathbb{E}[\mathbf{X}_{u}^{\geq a+\ell}]}\right) + \mathbb{E}\left[\mathbf{X}_Q\right] \cdot \left(b + \log \frac{1}{\mathbb{E}[\mathbf{X}_Q]}\right) + \alpha \cdot |Q| \cdot \left(b + \log \frac{1}{\alpha}\right).$$

Proof. We start by presenting our communication protocol below followed an analysis of correctness and the encoding length.

Alice's Encoding. We describe the procedure used by Alice to encode the correct answers $A(U, Q)$. Recall that Alice and Bob share the update sequence U' as well as public randomness $\mathcal{R}$. The encoding consists of five phases.

1. Alice reconstructs the missing ℓ update operations $u_a, \ldots, u_{a+\ell-1}$ by trying all possible update operations until finding the sequence that matches the answers in the set $A(U, Q)$.

2. Alice runs the data structure, using shared random string $\mathcal{R}$, and executes all the update operations $(u_1, \ldots, u_{a-1})$. That is, Alice executes all updates until the ones missing from Bob's input.

3. Alice executes the missing update operations $(u_a, \ldots, u_{a+\ell-1})$ that are not part of Bob's input. At the end of this phase, Alice appends the c bits found in client memory after the last update operation $u_{a+\ell-1}$.

4. Alice executes all updates in the remaining update operations known to both parties ($u_{a+\ell}$, $\ldots$, u_m) using the shared random string $\mathcal{R}$.

   In this phase, Alice keeps a list of all the *special* probes of this phase. A probe is *special* if it is a probe to a cell last overwritten by an update in the missing group. For this purpose, the probes of this phase are indexed with the integers $0, 1, 2$ and so forth. At the end of this phase, Alice appends an encoding of the set of indices of the special probes along with the ordered sequence of all the cell contents read by these probes.

5. Alice executes each query q from the query set Q. All queries are executed starting from the state of the data structure at the end of update u_m (that is, after the last update operation). In this phase, Alice keeps two lists: a list of the *non-free* queries that include a probe to a cell last overwritten in epoch i and a list of the *wrong* queries for which the data structure returns the wrong answer. At the end of this phase, Alice appends an encoding of the subset of queries that are either non-free or wrong along with the ordered sequence of the correct answers of the non-free and wrong queries.

Bob's Decoding. We describe Bob's decoding algorithm to recover the correct answers in $A(U, Q)$. Recall that Bob receives the subsequence of update operations $U' = (u_1, \ldots, u_{a-1}, u_{a+\ell-1}, \ldots, a_m)$ and the random string $\mathcal{R}$.

1. Bob executes the updates $u_1, \ldots, u_{a-1}$ using the shared random string $\mathcal{R}$.

2. Bob skips the missing updates $u_a, \ldots, u_{a+\ell-1}$ and reads the content of the client memory at the end of update $u_{a+\ell-1}$ found in the encoding. Bob keeps the server memory in the state at the end of update operation u_{a-1}.

3. Bob executes the remaining updates $u_{a+\ell}, \ldots, a_m$ using the shared random string $\mathcal{R}$. Before performing a probe as requested by the data structure, Bob checks if the probe is in the list of the special probes as found in the encoding. If the probe is special, Bob uses the cell contents found in the encoding. Otherwise, Bob performs the probe using the current snapshot of the server memory.

4. Bob takes a snapshot of the server and client memory at the end of update u_m and uses it as a starting state for all the queries $q \in Q$. For each query q, Bob first checks whether the query is non-free or wrong. If so, the answer of the query is read from the encoding. Otherwise, the answer of the query is obtained by executing the data structure's query algorithm.

Correctness. As Alice and Bob share the same random string $\mathcal{R}$ and updates outside of the missing group $u_a, \ldots, u_{a+\ell-1}$, their executions of the data structure are identical up to update operation u_{a-1}. For all updates $u_{a+\ell}$ and afterwards, every probe to a cell last overwritten by an update in the missing group $u_a, \ldots, u_{a+\ell-1}$ (thus, the cell contents are unknown to Bob) are encoded by Alice. Therefore, all cells overwritten in update operation $u_{a+\ell}$ and after are correct and identical to Alice's execution. Finally, for the $|Q|$ queries, we note that Bob can get the correct answer whenever the query is correct and does not probe any cell last overwritten by the missing group. As Alice encodes the answer for all queries not satisfying the above two conditions, Bob will always retrieve the correct answers for every query $q \in Q$.

Expected length of the encoding. We now bound the expected length of the encoding produced by Alice in each phase.

1. Alice does not produce any encoding in phase 1 and phase 2.

2. In phase 3, Alice encodes client memory for a total of c bits.

3. Phase 4 contributes the encoding of the subset of special probes along with contents of the cells probed by a special probe. In other words, a subset of $X_u^{\geq a+\ell}$ probes of the set of total probes performed during the updates $u_{a+\ell}, \ldots, u_m$ after the missing group along with ω bits for each of the $X_u^{\geq a+\ell}$ probes. Therefore, phase 3 contributes at most:

$$
\begin{aligned}
&\mathbb{E}\left[\log \binom{\mathbf{T}_u^{\geq a+\ell}}{\mathbf{X}_u^{\geq a+\ell}} + \omega \cdot \mathbf{X}_u^{\geq a+\ell}\right] \\
&\quad \leq \mathbb{E}\left[\mathbf{X}_u^{\geq a+\ell} \log \frac{\mathbf{T}_u^{\geq a+\ell}}{\mathbf{X}_u^{\geq a+\ell}}\right] + \omega \cdot \mathbb{E}\left[\mathbf{X}_u^{\geq a+\ell}\right] \qquad \left(\text{by } \log \binom{n}{k} \leq k \log(n/k)\right) \\
&\quad \leq \mathbb{E}\left[\mathbf{X}_u^{\geq a+\ell}\right] \log \frac{\mathbb{E}[\mathbf{T}_u^{\geq a+\ell}]}{\mathbb{E}[\mathbf{X}_u^{\geq a+\ell}]} + \omega \cdot \mathbb{E}\left[\mathbf{X}_u^{\geq a+\ell}\right] \qquad (\text{by concavity}) \\
&\quad \leq \mathbb{E}\left[\mathbf{X}_u^{\geq a+\ell}\right] \left(\omega + \log \frac{t_u(m-a-\ell+1)}{\mathbb{E}[\mathbf{X}_u^{\geq a+\ell}]}\right). \qquad \left(\text{by } \mathbb{E}[\mathbf{T}_u^{\geq a+\ell}] \leq t_u(m-a-\ell)\right)
\end{aligned}
$$

4. Phase 5 contributes the encoding of the set of non-free and wrong queries. If the data structure has probability of error $0 \leq \alpha < 1$, then the expected number of wrong queries is $\alpha \cdot |Q|$ queries and b bits are added to the encoding for each wrong query. Similarly, for the non-free queries, the encoding of a subset of size P_Q^i of a set of size ℓ_i followed by $b \cdot P_Q^i$ bits. Therefore, we

know that phase 4 contributes at most the following expected number of bits to the encoding.

$$\begin{aligned}\mathbb{E}\left[\log\binom{|Q|}{\alpha\cdot|Q|}\right]+\alpha b|Q|+\mathbb{E}\left[\log\binom{|Q|}{\mathbf{X}_Q}\right]+b\cdot\mathbb{E}\left[\mathbf{X}_Q\right]&\\ \leq \alpha|Q|(b+\log 1/\alpha)+\mathbb{E}\left[\mathbf{X}_Q\cdot\log\frac{|Q|}{\mathbf{X}_Q}\right]+b\cdot\mathbb{E}\left[\mathbf{X}_Q\right]&\\ \leq \alpha|Q|(b+\log 1/\alpha)+\mathbb{E}\left[\mathbf{X}_Q\right]\cdot\left(b+\log\frac{|Q|}{\mathbb{E}[\mathbf{X}_Q]}\right).&\end{aligned}$$

This completes the proof of our public coin one-way communication protocol. □

4.2 The Hard Distribution

We now describe the distribution of updates and queries that we will use to prove our lower bound. We then show how to organize the updates into epochs so to utilize the protocol of the previous section. At a high level, by looking at the components of the communication cost of Lemma 1, we notice that the first component is due to the reconstruction of the necessary information for future updates occurring after the missing updates. The second component takes into account the amount of information for queries that were embedded during the missing update operations. Note that the first component depends only on the update time while the second component depends only on the query time. We organize our updates into epochs so that the two components are balanced and we can obtain a lower bound on the sum of query and update times.

Our hard distribution will make use of the random update sequence $\mathbf{U}=(\mathbf{u}_1,\ldots,\mathbf{u}_n)$ that we will assume exists for the data structure problem P. We define our hard distribution denoted by $\mathcal{U}$ as follows:

1. Pick m uniformly at random from $\{n/2+1, n/2+1,\ldots,n\}$.
2. Output sequence of operations $\mathcal{U}=(\mathbf{u}_1,\ldots,\mathbf{u}_m)$.

In other words, we are picking a random prefix of length between $n/2$ and n from the random update sequence $\mathbf{U}$. For any query q, we denote by $\mathcal{Q}(q)$ the distribution over the sequences $Q(q)=(U,q)$ obtained by selecting U according to $\mathcal{U}$. Additionally, we will consider $\mathcal{Q}(Q)$ with respect to a set of queries Q. In this case, $\mathcal{Q}(Q)$ consists of a random update sequence drawn from $\mathcal{U}$ as well as a uniformly random query drawn from Q. Our final hard distribution will be $\mathcal{Q}(q)$ for some query q.

Definition of epochs. Let $U=(u_1,\ldots,u_m)$ be a sequence of operations in the support of $\mathcal{U}$. Define r to be the multiplicative decay between each future epoch. We will choose a correct value of r later, but we will ensure that $r\geq 2$. We partition the operations of U into epochs of exponentially increasing sizes $\ell_1:=r,\ell_2:=r^2,\ldots$ with epochs starting from u_m and growing backward to u_1. That is, epoch 1 consists of operations $u_m,u_{m-1},\ldots,u_{m-r+1}$, epoch 2 of operations $u_{m-r},\ldots,u_{m-r-r^2+1}$ and so on. We define $s_i:=\sum_{j=1}^{i}\ell_j$ to be the number of the operations in epochs $1,\ldots,i$. Another way to view s_i is that it is the total number of operations that occur after epoch $i+1$. We will denote U_i to be the set of all updates in epoch i. We will also denote U_{-i} to be the set of all updates except for those that are updated in the i-th epoch. The index of the starting update operation of the i-th epoch will be denoted by p_i.

We say that an epoch is *large* if $\ell_i \geq \max\{8c/v, \sqrt{n}\}$ and we denote by K the number of large epochs. Note that the number of large epochs is $K = \Theta(\log_r(m/\ell_i)) = \Theta(\log_r(nv/c))$.

The organization of updates into epochs formalizes the intuition provided in the previous section. For any large epoch i with ℓ_i updates, we note that the number of update operations following it is at most $2\ell_i/r$. In other words, the future updates are a little bit smaller than the total number of updates within the i-th epoch. So, we can balance the two components of the communication cost in the one-way protocol from the prior section to prove our lower bound.

Important notions of information. Finally, we will introduce two more variables that will aim to capture the notion of information that will be utilized throughout our work. In particular, these will look very similar to the communication cost of the one-way protocol from the prior section. However, they will be used specifically with respect to our epoch organization. We denote $\mathbf{X}_u^{\geq p_{i-1}}$ to be the number of probes performed by updates occurring after the i-th epoch (that is, in epochs $i-1, \ldots, 0$) that access a cell last overwritten in the i-th epoch. We denote $\mathbf{X}_Q^i$ to be the number of probes performed by all queries $q \in Q$ to cells last overwritten in the i-th epoch. We denote $\mathbf{X}_q^i$ to the number of probes performed by a single query $q \in Q$ to cells last overwritten in the i-th epoch.

We denote by $\mathcal{Z}(i, Q)$ the quantity defined as follows:

$$\mathcal{Z}(i, Q) = \min\left\{|Q| \cdot v, \mathbb{E}\left[\mathbf{X}_u^{\geq p_{i-1}}\right](\omega + \log\log n) + b \cdot \mathbb{E}\left[\mathbf{X}_Q^i\right]\right\}.$$

This captures the total amount of information needed to answer all queries $q \in Q$. Note, this matches the communication cost of our one-way communication protocol in Lemma 1 by plugging in the i-th epoch as the missing group of updates. We use the minimum as we know the information transferred is at most $|Q| \cdot v$ bits as v bits are learned on average from each $|Q|$ queries. For a single query $q \in Q$, we can similarly define $\mathcal{Z}(i, q)$:

$$\mathcal{Z}(i, q) = \min\left\{v, \frac{\mathbb{E}[\mathbf{X}_u^{\geq p_{i-1}}](\omega + \log\log n)}{|Q|} + b \cdot \mathbb{E}\left[\mathbf{X}_q^i\right]\right\}.$$

We use minimum as the average information in a single query is v bits. We will utilize $\mathcal{Z}(i, Q)$ and $\mathcal{Z}(i, q)$ later as the events that can be viewed by an adversary.

4.3 Bounding Query and Update Times

Finally, we will finish the proof of Theorem 3 in this section. In particular, we will leverage the epoch organization as well as our one-way communication protocol to prove lower bounds on the query and update times. To do this, we start by showing that the cost of the one-way communication protocol can be directly related to the entropy of the correct answers of the query set.

Lemma 2. *Consider a data structure* DS *with error probability* $\alpha \leq v/(b\log^2 n)$ *for a data structure problem* P *that satisfies the Large Information Retrieval property. Then for every large epoch* i *such that* $\mathbb{E}[\mathbf{X}_u^{p_{i-1}}] = O(t_u\ell_i/(rK))$*, there exists a sequence* Q_i *of at least* ℓ_i *queries such that*

$$\mathcal{Z}(i, Q_i) = \Omega(\ell_i \cdot v).$$

Proof. We remind the reader that a large epoch i consists of $\ell_i \geq \sqrt{n}$ update operations. Therefore, if we consider a sequence $\mathbf{U}$ of n updates and the sub-sequence of the ℓ_i updates of the i-th epoch then, by the Large Information Retrieval property, there exists a sequence Q_i of queries such that

$$H(\mathbf{A}(\mathbf{U}, Q_i) \mid \mathbf{U}_{-i}) \geq \ell_i v,$$

where $\mathbf{U}_{-i}$ is the sequence of updates obtained from $\mathbf{U}$ by removing the updates of epoch i. Since Q_i has at least ℓ_i queries, it suffices to focus on the second argument of the minimum of the definition of $\mathcal{Z}(i)$.

Next, we utilize query set Q_i in the context of the one-way communication protocol of Lemma 1, where Alice and Bob receive the updates $\mathbf{U}_{-i}$. By Shannon's source coding theorem, the expected length of Alice's encoding of $\mathbf{A}(\mathbf{U}, Q)$ must be at least the entropy of $\mathbf{A}(\mathbf{U}, Q)$ conditioned on the shared information $\mathbf{U}_{-i}$ and $\mathcal{R}$. Moreover, observe that $\mathcal{R}$ is chosen independently from $\mathbf{U}$ and Q and thus the expected length of the encoding must be at least

$$H(\mathbf{A}(\mathbf{U}, Q), \mid \mathbf{U}_{-i}, \mathcal{R}) = H(\mathbf{A}(\mathbf{U}, Q) \mid \mathbf{U}_{-i}) \geq \ell_i v.$$

In other words, the expected communication cost must be $\Omega(\ell_i \cdot v)$.

Recall that we use p_i to denote the position of the first operation of the i-th epoch and s_{i-1} is the number of update operations in epochs $1, 2, \ldots, i-1$. Furthermore, we use $\mathbf{X}^i_{Q_i}$ to denote the number of probes by queries $q \in Q_i$ into cells last overwritten by updates $\mathbf{U}_i$ in the i-th epoch. Therefore, by Lemma 1,

$$c + \mathbb{E}\left[\mathbf{X}_u^{\geq p_{i-1}}\right]\left(\omega + \log \frac{t_u s_{i-1}}{\mathbb{E}[\mathbf{X}_u^{\geq p_{i-1}}]}\right) + \mathbb{E}\left[\mathbf{X}^i_{Q_i}\right]\left(b + \log \frac{|Q_i|}{\mathbb{E}[\mathbf{X}^i_{Q_i}]}\right) + \alpha \cdot |Q_i| \cdot \left(b + \log \frac{1}{\alpha}\right) \geq \ell_i v.$$

Note, that $\alpha \leq v/(b \log^2 n)$, so we get that the last addend is at most $|Q_i| v / \log^2 n \cdot (1 + \log(bn/v))$. As $b = n^{O(1)}$ and $|Q_i| = O(\ell_i)$, we get that this is at most $\ell_i v \cdot O(1/\log n) \leq \ell_i v/8$ for sufficiently large n. For a large epoch, we also have $c \leq \ell_i v/8$. Therefore,

$$\mathbb{E}\left[\mathbf{X}_u^{\geq p_{i-1}}\right]\left(\omega + \log \frac{t_u s_{i-1}}{\mathbb{E}[\mathbf{X}_u^{\geq p_{i-1}}]}\right) + \mathbb{E}\left[\mathbf{X}^i_{Q_i}\right]\left(b + \log \frac{|Q_i|}{\mathbb{E}[\mathbf{X}^i_{Q}]}\right) \geq \frac{3}{4} \cdot \ell_i v.$$

Consider two cases. If $\mathbb{E}\left[\mathbf{X}^i_{Q}\right] \leq |Q_i|/16$, then $\log \frac{|Q_i|}{\mathbb{E}[\mathbf{X}^i_{Q_i}]} \leq 4$. Therefore,

$$\mathbb{E}\left[\mathbf{X}_u^{\geq p_{i-1}}\right]\left(\omega + \log \frac{t_u s_{i-1}}{\mathbb{E}[\mathbf{X}_u^{\geq p_{i-1}}]}\right) + b \cdot \mathbb{E}\left[\mathbf{X}^i_{Q_i}\right] \geq \frac{1}{2} \cdot \ell_i v$$

as $|Q_i| \geq \ell_i$. Finally, we use the fact that $\mathbb{E}[\mathbf{X}_u^{p_{i-1}}] = O(t_u \ell_i/(rK))$ and plug it into the above to obtain the following inequality:

$$\mathbb{E}\left[\mathbf{X}_u^{\geq p_{i-1}}\right](\omega + \log\log n) + b \cdot \mathbb{E}\left[\mathbf{X}^i_{Q_i}\right] = \Omega(\ell_i v)$$

where we used the fact that $s_{i-1}/\ell_i \leq 2/r$ and $K = O(\log n)$ by our epoch construction. This completes the proof for the case of $\mathbb{E}\left[\mathbf{X}^i_{Q_i}\right] \leq |Q_i|/16$. For the other case when $\mathbb{E}\left[\mathbf{X}^i_{Q_i}\right] \geq |Q_i|/16$, we can see that the result is trivially obtained by plugging the value into $\mathcal{Z}(i)$ since $|Q_i| \geq \ell_i$. □

The lemma above tells us that, for every large epoch, there exists one set of queries with a large value of $\mathcal{Z}$. The set of "expensive" queries depends on the epoch and different epochs might have different bad queries. Conversely, a query q might be "expensive" for one epoch but not for the others. Next, we show that the Adversarially Observable Event implies that there exists one query that is "expensive" for all epochs.

Using the Adversarially Observable Event. As we shall see later, this property is guaranteed by the security notions (differential privacy, obliviousness) that we will consider for the specific data structure problems for which we will derive lower bounds. We note that quantity $\mathcal{Z}(i, q)$ only depends on the data structure probes and thus it can be efficiently computed by an adversary even without knowing the executed query q. Therefore, by the Adversarially Observable Event property, its value should not "vary too much" with q.

To formalize this, we define the event $\mathbf{E}_q^i$ to be a binary random variable that checks whether $\mathcal{Z}(i, q)$ is above a certain threshold. In particular, we denote

$$\mathbf{E}_q^i = 1 \iff \frac{\mathbf{X}_u^{\geq p_{i-1}}}{|Q|} (\omega + \log\log n) + b \cdot \mathbf{X}_q^i \geq \beta v$$

for some constant $\beta > 0$ that we will choose later. Note, the above formula is the second argument of $\mathcal{Z}(i, q)$. We will also use $\mathbf{E}_Q^i$ as a binary random variable with respect to the second argument of $\mathcal{Z}(i, Q)$ as follows:

$$\mathbf{E}_Q^i = 1 \iff \mathbf{X}_u^{\geq p_{i-1}} (\omega + \log\log n) + b \cdot \mathbf{X}_Q^i \geq |Q|\beta v$$

We show that there exists a single query q such that $\Pr[\mathbf{E}_q^i = 1] \geq p$ for some constant probability $p > 0$ for all large epochs i. We prove this next:

Lemma 3. *Consider a data structure* DS *for a data structure problem P satisfying the Large Information Retrieval and the Event Probability Transfer properties. Then, there exists a query q and a constant $0 < p \leq 1$ such that for all large epochs i where $\mathbb{E}[\mathbf{X}_u^{p_{i-1}}] = O(t_u \ell_i/(rK))$, $\Pr[\mathbf{E}_q^i = 1] \geq p$.*

Proof. By Lemma 2, we know that for each large epoch i, it must be that the following holds for some constant $0 < \gamma < 1$:

$$\mathcal{Z}(i, Q) = \min\left\{\ell_i \cdot v, \mathbb{E}\left[\mathbf{X}_u^{\geq p_{i-1}}\right] (\omega + \log\log n) + b \cdot \mathbb{E}\left[\mathbf{X}_Q^i\right]\right\} \geq \gamma \cdot |Q| \cdot v$$

as $|Q| = O(\ell_i)$. We set the value β from the definition of the events $\mathbf{E}_Q^i$ and $\mathbf{E}_q^i$ equal to $\beta := \gamma/2$. For each large epoch i where $\mathbb{E}[\mathbf{X}_u^{p_{i-1}}] = O(t_u \ell_i/(rK))$, we will show there exists some query $q_i \in Q$ such that $\Pr[\mathbf{E}_{q_i}^i = 1] > p'$ for some constant positive probability p'. Suppose this is false and $\Pr[\exists q \in Q, \mathbf{E}_q^i = 1] = o(1)$. Then, we get that with probability at least $1 - o(1)$, the following is true:

$$\sum_{q \in Q} \mathcal{Z}(i, q) = \sum_{q \in Q} \frac{\mathbf{X}_u^{\geq p_{i-1}}}{|Q|} (\omega + \log\log n) + b \cdot \mathbf{X}_q^i \leq \gamma/2 \cdot |Q| \cdot v$$

where we can always use the second argument of $\mathcal{Z}(i, q)$ by our assumption that $\mathbf{E}_q^i \neq 1$. Let

$p_q^i = \Pr[\exists q \in Q, \mathbf{E}_q^i = 1]$. Next, we bound the expectation of $\mathcal{Z}(i, Q)$:

$$\begin{aligned}\mathbb{E}[\mathcal{Z}(i,Q)] &\le (1-p_q^i)\sum_{q\in Q}\left(\frac{\mathbf{X}_u^{\ge p_{i-1}}}{|Q|}(\omega + \log\log n) + b\cdot \mathbf{X}_q^i\right) + p_q^i\cdot(|Q|\cdot v)\\ &\le \gamma/2\cdot|Q|\cdot v + o(|Q|v) < \gamma\cdot|Q|\cdot v.\end{aligned}$$

To understand this inequality, we consider the two cases. We can always bound the value of $\mathcal{Z}(i,Q)$ by $|Q|\cdot v$ as we do when $\exists q\in Q$ such that $\mathbf{E}_q^i = 1$. For the other case, note that $\mathcal{Z}(i,q) < v$ so we can replace it with the second argument of $\mathcal{Z}(i,q)$ and apply linearity of expectation. Note that the last derived inequality contradicts with the first inequality from Lemma 2. Therefore, for some probabilty $p' > 0$, for each large epoch i such that $\mathbb{E}[\mathbf{X}_u^{p_{i-1}}] = O(t_u\ell_i/(rK))$, there exists $q_i \in Q$ and $\Pr[\mathbf{E}_{q_i}^i = 1] > p'$.

Since the value of $\mathbf{E}_q^i$ can be efficiently computed, the Event Probability Transfer property gives that there exists a query q such that $\Pr[\mathbf{E}_q^i = 1] = p$ for some constant $p > 0$ and for all large epochs i. □

The above lemma shows that there must exist one "expensive" query q for which $\mathcal{Z}(i,q)$ is large for all large epochs i for which the expected value of $\mathbf{X}_u^{p_{i-1}}$ is not too large. We next show that these extra conditions holds for all large epochs. In particular, we show that the average number of probes to cells last overwritten in each of the ℓ_i update operations is at most $O(t_u/r)$.

Lemma 4. *$\sum_i \mathbb{E}[\mathbf{X}_u^{\ge p_{i-1}}]/\ell_i = O(t_u/r)$ over all large epochs i.*

Proof. We start by identifying which probes contribute to $\mathbf{X}_u^{\ge p_{i-1}}$ for all large epochs i. Let us consider a probe occurring as part of the β-th update operation and denote by γ the index of the operation that last overwrote the same cell. We index operations according to the time they were performed so that updates occurring in epoch 1 have the largest index. In other words, update operations are numbered from left to right and we remind the reader that epochs are numbered from right to left. Therefore we have $\beta > \gamma$ and we let x denote the epoch satisfying the inequality $s_{x-1} \le \beta - \gamma < s_x$. Note that this x is unique as s_i grows as i grows. We break down the analysis into two different cases.

Case I: $i < x$. The probe does not contribute to $\mathbf{X}_u^{\ge p_{i-1}}$ for $i < x$ regardless of the location of the query operation. First, suppose that the query operation occurs immediately after the β-th operation; that is, the β-th update operation is part of epoch 1. Since $\beta - \gamma \ge s_{x-1}$, the γ-th operation takes place after epoch $x-1$ has finished and, since $i < x$, this implies that epoch i begins after the γ-th update has been performed. If instead the query operation does not occur immediately following the β-th operation then i-th epoch will begin even later and thus it is still after the γ-th update operation has been performed.

Case II: $i \ge x$. First of all observe that epoch $i-1$ must start between γ and β and thus there at most $\beta - \gamma < s_x$ good positions. Moreover, observe that epoch $i-1$ cannot start before $\beta - s_{i-1} + 1$, for otherwise operation β would take place before operation 1 which is clearly a contradiction. We thus have at most s_{i-1} good positions. Therefore the probability that a probe performed as part of the β-th update contributes to $\mathbf{X}_u^{\ge p_{i-1}}$ is at most $2/n\cdot\min\{s_{i-1}, s_x\}$.

Now a probe associated with x contributes to $\mathbf{X}_u^{\ge p_{i-1}}$, for randomly chosen epoch i, for at most $2/(nK)\cdot\min\{s_{i-1}, s_x\}$, for $i \ge x$. By summing over all $i \ge x$, we can bound the contribution of one probe to a random epoch by $\sum_{i \geq x} \frac{2\min\{s_{i-1}, s_x\}}{n} \frac{1}{K} \leq \frac{4\ell_i}{rnK}$. As we have at most $n \cdot t_u$ probes, we conclude that $\sum_i \mathbb{E}[\mathbf{X}_u^{\geq p_{i-1}}]/\ell_i \leq 4t_u/r$. □

Finally, we are ready to prove our main theorem.

Proof of Theorem 3. We start from Lemma 2. For every large epoch i such that $\mathbb{E}[\mathbf{X}_u^{\geq p_{i-1}}] = O(t_u \ell_i/(rK))$, we have $\Pr[\mathcal{Z}(i,q) = \Omega(v)] \geq p$ for some constant $p > 0$ for every query q. In other words, we know that by linearity of expectation:

$$\frac{\mathbb{E}[\mathbf{X}_u^{\geq p_{i-1}}]}{|Q|}(\omega + \log\log n) + b \cdot \mathbb{E}\left[\mathbf{X}_q^i\right] = \Omega(v).$$

First, we do the easier task of bounding $\mathbb{E}[\mathbf{X}_q^i]$. Note that the expected query time is $\sum_i \mathbb{E}[\mathbf{X}_q^i] \leq t_q$ where we only iterate over all large epochs i. Consider the experiment of picking a random epoch $\mathbf{i}$. Then, know that $\mathbb{E}[\mathbf{X}_q^{\mathbf{i}}] \leq t_q/K$ where K is the number of large epochs. By Markov's inequality, we know that $\Pr_{\mathbf{i}}[\mathbb{E}[\mathbf{X}_q^{\mathbf{i}}] \leq 100t_q/K] \geq 99/100$.

By Lemma 4, we know that $\sum_i \mathbb{E}[\mathbf{X}_u^{\geq p_{i-1}}]/\ell_i \leq \gamma t_u/r$ for some constant $\gamma > 0$ over all large epochs i. Again, we can show that for a random index $\mathbf{i}$, that $\Pr_{\mathbf{i}}[\mathbb{E}[\mathbf{X}_u^{\geq p_{i-1}}] \leq 100\gamma\ell_i t_u/(rK)] \geq 99/100$ as there are K large epochs. Then,

$$\Pr_{\mathbf{i}}\left[\mathbb{E}[\mathbf{X}_u^{\geq p_{i-1}}] \leq \frac{100\gamma\ell_i t_u}{rK} \wedge \mathbb{E}[\mathbf{X}_q^{\mathbf{i}}] \leq \frac{100t_q}{K}\right] \geq 98/100.$$

We pick any such i satisfying the above two inequalities for the rest of the proof. By plugging the above bounds into the inequality and using $|Q| = \Theta(\ell_i)$,

$$\frac{t_u}{rK}\left(\omega + \log\frac{s_{i-1}}{\ell_i} rK\right) + \frac{t_q}{K} b = \Omega(v).$$

By using the fact that $s_{i-1}/\ell_i \leq 2/r$ and $K = O(\log n)$ as $r \geq 2$, we obtain

$$\frac{t_u}{rK}(\omega + \log\log n) + \frac{t_q}{K} b = \Omega(v).$$

Finally by substituting $r = 2 + (\omega + \log\log n)/b$ and $K = \Theta(\log_r(nv/c))$,

$$t_u + t_q = \Omega\left(\frac{v}{b} \cdot K\right) = \Omega\left(\frac{v}{b} \cdot \frac{\log(nv/c)}{1 + \log((\omega + \log\log n)/b)}\right)$$

that completes our proof. □

4.4 Extension to Multiple Non-Colluding Servers

In this section, we show that our framework may also be extended to the multiple non-colluding server setting. We assume there are k servers and the PPT adversary has compromised exactly one server. Our lower bound immediately applies to settings where the adversary compromises multiple (or even all) servers. First, we define the equivalent of the Event Probability Transfer property for k servers.

Definition 4 (k-Event Probability Transfer.)**.** *For any update sequence U and query q, let $E_i(U, q)$ be some event that can be checked whether to have occurred by a PPT adversary that compromised the i-th server. Suppose that $\Pr[E_i(U, q)] \geq \zeta/k$ for some constant $\zeta > 0$. Then, we say that a data structure enjoys the* k-Event Probability Transfer *property if for any query q', it holds that*

$$\Pr[E_i(U, q')] = \Omega(\Pr[E_i(U, q)])$$

where the probability is over the internal randomness of the data structure.

We present our theorem below and defer the proof to Appendix A that adapts some ideas from [LSY20] for our proof technique.

Theorem 4. *Consider a data structure problem P that allows* `update` *and* `query` *operations such that query outputs are b bits and $b = n^{O(1)}$. Consider a data structure* DS *that implements problem P over k servers with expected update and query overhead t_u and t_q respectively, client storage c and error probability $\alpha \leq v/(b \log^2 n)$ in the cell probe model with $\omega \geq 1$ cell size. If P enjoys the Large Information Retrieval property and the Event Probability Transfer property then*

$$t_u + t_q = \Omega\left(\frac{v}{b} \cdot \frac{\log(nv/c)}{1 + \log((\omega + \log\log n)/b)}\right).$$

The above lower bound holds even for $k = n^{O(1)}$ servers. In particular, the above can be used to show lower bound that even if a PPT adversary compromises only one of $k = n^{O(1)}$ servers under certain privacy properties. See Section 5.1 for some further discussion.

5 Lower Bounds

In this section, we show that our framework may be used to derive a whole new set of logarithmic lower bounds for differentially private (and, thus, oblivious) versions of data structure problems.

We start by applying our framework to prove our main result of logarithmic lower bounds for DP RAMs in the setting of $b \ll \omega$. To show that our framework may handle various privacy guarantees, we show that we can extend the searchable encryption lower bounds in [PPY20] for the setting of $b \ll \omega$. We also consider a suite of classical data structures where $o(\log n)$ overhead is known without any privacy guarantees. Through our framework, we show that these data structures require logarithmic overhead as soon as privacy requirements are enforced.

5.1 Differentially Private RAMs

As the first application of our framework, we will prove logarithmic lower bounds for differentially private (DP) RAMs. As a reminder, a prior lower bound of $\Omega(b/\omega \cdot \log(nb/c))$ was proved in [PY19]. However, this does not preclude sub-logarithmic overhead when $b \ll \omega$. For example, if $b = O(1)$ and $\omega = \Theta(\log n)$, the above lower bound becomes trivial at $\Omega(1)$. In this section, we show that this lower bound remains logarithmic even in the case when $b \ll \omega$.

We start by defining $(\epsilon, \delta, 1, k)$-DP for k-server data structures for which the view of an adversary that corrupts 1 of the k servers is (ϵ, δ)-DP, following the definition in [PY19] where neighboring sequences of operations are those that differ in exactly one operation. As a note, this definition uses computational differential privacy with respect to efficient adversaries.

Definition 5. *A data structure* DS *is* $(\epsilon, \delta, 1, k)$*-DP (differentially private) if for any pair of operational sequences* O_1 *and* O_2 *that differ in at most one operation and any PPT adversary* $\mathcal{A}$ *that compromises one of the* k *servers,*

$$\Pr[\mathcal{A}(\mathcal{T}_{\mathsf{DS}}(O_1)) = 1] \leq e^{\epsilon} \Pr[\mathcal{A}(\mathcal{T}_{\mathsf{DS}}(O_2) = 1] + \delta$$

where $\mathcal{T}_{\mathsf{DS}}(O)$ *is the transcript seen by the adversary across all compromised servers when the operational sequence* O *is executed by* DS*.*

We prove a generic lemma about $(\epsilon, \delta, 1, k)$-DP data structures. In particular, we show that any $(\epsilon, \delta, 1, k)$-DP data structure satisfies the second conditions of our frameworks. We choose to be generic and prove the result for arbitrary problems and $k \geq 1$ server(s).

Lemma 5. *Suppose a data structure is* $(\epsilon, \delta, 1, k)$*-DP for any constant* $\epsilon \geq 0$ *and* $\delta < \beta/k$*, for a sufficiently small constant* $\beta > 0$*. Let* $E(U, q)$ *be some event that can be checked whether to have occurred by a PPT adversary for update sequence* U *and query* q *such that* $\Pr[E(U, q)] \geq \zeta/k$ *for some constant* $\zeta > 0$*. Then, for any update sequence* U *and any queries* q *and* q'*,* $\Pr[E(U, q')] = \Omega(\Pr[E(U, q)])$ *where the probability is over the internal randomness of the data structure.*

Proof. Consider the operational sequences $O_1 = (U, q)$ and $O_2 = (U, q')$. By definition, O_1 and O_2 are neighboring inputs. Therefore, for any adversarially observable event E, we know that $\Pr[E(O_1)] \leq e^{\epsilon} \Pr[E(O_2)] + \delta$. By re-arranging, we get that $\Pr[E(O_2)] = \Omega(\Pr[E(O_1)] - \delta)$. As we assumed that $\Pr[E(O_1)] \geq \zeta/k$ for some constant $\zeta > 0$, there exists some constant $\beta > 0$ depending only on ζ such that if $\delta < \beta/k$, then the lemma is true. □

As our last step, we prove a lower bound for $(\epsilon, \delta, 1, k)$-DP RAMs. As the above lemma already shows that this privacy guarantees satisfies the second condition, it remains to show that the first condition of Theorem 4 can be satisfied.

Theorem 5. *Any* $(\epsilon, \delta, 1, k)$*-DP data structure* DS *that solves the dynamic array maintenance problem for* n b*-bit entries with constant* $\epsilon > 0$ *and* $\delta < \beta/k$*, for a sufficiently small constant* $\beta > 0$*, expected update and query time* t_u *and* t_q*, client storage* c *and error probability* $\alpha \leq 1/\log^2 n$ *in the cell probe model with* $\omega \geq 1$ *cell size must satisfy the following:*

$$t_u + t_q = \Omega\left(\frac{\log(nb/c)}{1 + \log((\omega + \log\log n)/b)}\right).$$

Proof. We will use the framework of Theorem 3 to prove our lower bound. For our random update sequence $\mathbf{U} = (\mathbf{u}_1, \ldots, \mathbf{u}_n)$, each $\mathbf{u}_i$ is generated by picking a uniformly random $\mathbf{B}_i$ from $\{0, 1\}^b$ and overwriting the i-th entry with $\mathbf{B}_i$. For any sequence of updates $\mathbf{u}_a, \ldots, \mathbf{u}_{a+\ell-1}$, we can construct the query set $\{a, \ldots, a + \ell - 1\}$. As the queries successfully read back the entries $\mathbf{B}_a, \ldots, \mathbf{B}_{a+\ell-1}$, we get that $H(\mathbf{A}(\mathbf{U}, Q) \mid \mathbf{u}_1, \ldots, \mathbf{u}_{a-1}, \mathbf{u}_{a+\ell}, \ldots, \mathbf{u}_n) \geq \ell \cdot b$. The second condition of Theorem 4 is satisfied by Lemma 5. By applying Theorem 4 with $b = v$, we obtain the desired lower bound. □

Discussion about k and δ. We note that for the setting of $k \geq 2$ servers and one compromised server, we can only prove non-trivial lower bounds when $\delta < 1/k$. To see this, note that there is a trivial algorithm that picks one of the random k servers and performs a plaintext data structure. An adversary will only see anything with probability at most $1/k$. Therefore, this is a $(0, 1/k)$-DP data structure. Our lower bound shows that anything with stronger security parameters results in the identical lower bound as the single-server model. As an extreme example, if $k = n^{O(1)}$ and $\delta = \mathsf{negl}(n)$, our lower bound still holds.

5.2 Set Membership

Next, we move onto proving lower bounds for other data structures. In general, previous lower bounds have focused on "key-value" types of data structures. For example, RAMs are essentially arrays with keys from $[n]$ and b-bit values. Prior lower bounds relied upon the fact that the b-bit value is truly random.

We show that our lower bound framework can also used to prove lower bounds for data structures without associated values. For the first such problem, we will consider the simple dynamic set data structure that maintains a subset $S \subseteq [n]$ that enables the following two operations:

1. $\texttt{add}(i)$: Adds item $i \in U$ into subset S.
2. $\texttt{query}(i)$: Returns 1 if $i \in S$ and 0 otherwise.

Note that the set problem is a natural problem where the query output size is only a single bit that will most likely be much smaller than the word size ω.

In the non-oblivious setting, it is clear that the dynamic set problem over the universe $[n]$ can be solved with $O(1)$ time using a bit vector of length n. Using our framework, we will show that the dynamic set membership problem with differential privacy requires logarithmic overhead.

Theorem 6. *Any $(\epsilon, \delta, 1, k)$-DP data structure* DS *that solves the dynamic set problem over $[n]$ with constant $\epsilon > 0$ and $\delta < \beta/k$ for a sufficiently small constant $\beta > 0$, expected update and query time t_u and t_q, client storage c and error probability $\alpha \leq 1/\log^2 n$ in the cell probe model with $\omega \geq 1$ cell size must satisfy:*

$$t_u + t_q = \Omega\left(\frac{\log(n/c)}{1 + \log(\omega + \log\log n)}\right).$$

Proof. To prove this, we will use the framework outlined in Theorem 4. First, we define a random sequence of $n/2$ updates as follows $\mathbf{U} = (\mathbf{u}_1, \ldots, \mathbf{u}_{n/2})$. Each $\mathbf{u}_i$ will add either $2i-2$ or $2i-1$ with probability 1/2 each. Consider any consecutive sequence of $\mathbf{u}_a, \ldots, \mathbf{u}_{a+\ell-1}$ of update operations. We define the query set Q as querying for membership of entries $\{2a-2, 2a, \ldots, 2(a+\ell-1)-2\}$. Let $\mathbf{A}(\mathbf{U}, Q)$ be the correct answers for queries $q \in Q$ with respect to $\mathbf{U}$. Note that

$$H(\mathbf{A}(\mathbf{U}, Q) \mid \mathbf{u}_1, \ldots, \mathbf{u}_{a-1}, \mathbf{u}_{a+\ell}, \ldots, \mathbf{u}_{n/2}) = \ell.$$

To see this, note that each query exactly determines the choice of each update operation. Therefore, we show this satisfies the first condition of Theorem 4. The second condition is satisfied immediately by Lemma 5.

As a result, we can now apply Theorem 4 with values $b = 1$ and $v = 1$ as query outputs are 1 and the above entropy argument completing our proof. □

5.3 Predecessor and Successor

We consider another classic data structure for which sub-logarithmic overhead constructions are known without any privacy requirements. In this section, we will prove lower bounds for the predecessor and successor problem. The predecessor data structure stores subset $S \subseteq U$ of size at most n with the following:

- $\texttt{add}(i)$: Adds item $i \in U$ into subset S.

- query(i): Returns the value $\max\{s \in S : s \leq i\}$. That is, the largest value that is not strictly larger than the value of i.

In the non-oblivious setting, there exists dynamic predecessor and successor data structures with overhead $O(\log\log|U|)$ using van Emde Boas trees [vEB75]. For standard settings of $|U| = n^{O(1)}$, this becomes $O(\log\log n)$. With differentially privacy, the overhead must be logarithmic.

Theorem 7. *Any $(\epsilon, \delta, 1, k)$-DP data structure* DS *that solves the dynamic predecessor (successor) problem over universe U storing at most n items with constant $\epsilon > 0$ and $\delta < \beta/k$ for a sufficiently small constant $\beta > 0$, expected update and query time t_u and t_q, client storage c and error probability $\alpha \leq 1/\log^2 n$ in the cell probe model with $\omega \geq 1$ cell size must satisfy the following:*

$$t_u + t_q = \Omega\left(\frac{\log(n\log(|U|/n)/c)}{1 + \log((\omega + \log\log n)/\log(|U|/n))}\right).$$

Proof. To prove our lower bound, we consider an artificial version of predecessor (successor) with smaller query output sizes. In particular, we consider a variant that returns the output modulo $|U|/n$. In this new problem, the query output size is $\log(|U|/n)$ as opposed to $\log(|U|)$. Clearly, a lower bound for this artificial problem implies a lower bound for the original problem.

We construct a random sequence of n updates $\mathbf{U} = (\mathbf{u}_1, \ldots, \mathbf{u}_n)$ as follows. For each $\mathbf{u}_i$, we add the number $(i-1)|U|/n + \mathbf{R}_i$ where $\mathbf{R}_i$ is a uniformly random integer from the set $[|U|/n]$. Consider any consecutive sequence of updates $\mathbf{u}_a, \ldots, \mathbf{u}_{a+\ell-1}$. Define the query set Q as querying for the predecessor of $\{a|U|/n - 1, (a+1)|U|/n - 1, \ldots, (a+\ell-1)|U|/n - 1\}$. For successor, we can instead use the query set Q of $\{(a-1)|U|/n, \ldots, (a+\ell-2)|U|/n\}$. Then,

$$H(\mathbf{A}(\mathbf{U}, Q) \mid \mathbf{u}_1, \ldots, \mathbf{u}_{a-1}, \mathbf{u}_{a+\ell}, \ldots, \mathbf{u}_n) \geq \ell \cdot \log(|U|/n).$$

It is not hard to see that given the set $\mathbf{A}(\mathbf{U}, Q)$, one can decode the random integers $\mathbf{R}_a, \ldots, \mathbf{R}_{a+\ell-1}$. The second condition is satisfied by Lemma 5. To complete the proof, we apply Theorem 4 with $b = v = \log(|U|/n)$. □

5.4 Disjoint Sets (Union-Find)

Another classic data structure that has very efficient (sub-logarithmic) overhead is the disjoint sets (union-find) data structure. At a high level, the disjoint sets data structure must maintain n items that may be arranged into disjoint sets. Initially, the n items are assumed to be in n individual different sets. Afterwards, the following operations may be performed:

- union(a, b): Given $a, b \in [n]$, merge the two sets containing a and b.
- find(a): Given an item $a \in [n]$, return the identity of the set containing a.

For correctness, it is required that if two items $a, b \in [n]$ are in the same set, then find(a) should be equal to find(b). Also, if a and b are not in the same set, then find(a) should be different from find(b). We will assume that set representations are integers from the set $[n^{O(1)}]$ as done by classic constructions. Thus, the query output size is $O(\log n)$ bits.

There are classic constructions [TVL84] that require only $O(\alpha(n))$ overhead where $\alpha(n)$ is the inverse Ackermann function. In all reasonable settings, $\alpha(n)$ is practically constant. If we enforce differentially privacy, we leverage our framework to prove a logarithmic lower bound.

Theorem 8. *Any $(\epsilon, \delta, 1, k)$-DP data structure* DS *that solves the dynamic disjoint set problem over at most n items with constant $\epsilon > 0$ and $\delta < \beta/k$ for a sufficiently small constant $\beta > 0$, expected update and query time t_u and t_q, client storage c and error probability $\alpha = O(1/\log^2 n)$ in the cell probe model with $\omega \geq 1$ cell size must satisfy the following:*

$$t_u + t_q = \Omega\left(\frac{\log(n/c)}{1 + \log(\omega/\log n)}\right).$$

Proof. We construct the random update sequence $\mathbf{U} = (\mathbf{u}_1, \ldots, \mathbf{u}_{n/2-\sqrt{n}})$ in the following way. We split the set of n items into two halves denoted by (i, j) where $i \in \{0, 1\}$ and $j \in [n/2]$. To generate update $\mathbf{u}_i$, pick a number $\mathbf{R}_i$ uniformly at random from $[\sqrt{n}]$ and merge the items $(0, i)$ and $(1, i + \mathbf{R}_i)$. For any sequence of updates $\mathbf{u}_a, \ldots, \mathbf{u}_{a+\ell-1}$, we can pick the queries to search for the set representation of items $\{(0, a), \ldots, (0, a + \ell - 1), (1, a), \ldots, (1, a + \ell + \sqrt{n} - 1)\}$. Note the total number of queries is $2\ell + \sqrt{n} = O(\ell)$ for any $\ell \geq \sqrt{n}$. Then, we can see that the entropy is

$$H(\mathbf{A}(\mathbf{U}, Q) \mid \mathbf{u}_1, \ldots, \mathbf{u}_{a-1}, \mathbf{u}_{a+\ell}, \ldots, \mathbf{u}_n) \geq \ell \log(\sqrt{n}) = \Omega(\ell \log n).$$

One can see this by simply checking that the queries enable one to accurately retrieve the random values embedded in the updates $\mathbf{R}_a, \ldots, \mathbf{R}_{a+\ell-1}$. Consider update $\mathbf{u}_i$ such that $a \leq i < a + \ell$ and look at the output of the query to $(0, i)$. Next, search from the queries $(1, i), \ldots, (1, i + \sqrt{n} - 1)$ and find the index j such that the query for $(1, j)$ is the same as $(0, i)$. We show that this index j is unique as the query outputs of $(1, x)$ and $(1, y)$ for any $x \neq y$ will always be different as $(1, x)$ and $(1, y)$ will never be merged and each item $(0, z)$ is merged with at most one other item. As a result, we know that $\mathbf{R}_i = j - i$. The second condition is satisfied by Lemma 5. Finally, we can apply Theorem 4 with $b = O(\log n)$ and $v = O(\log n)$ to obtain our desired lower bound. □

5.5 Searchable Encryption (Encrypted Multi-Maps)

Finally, we show that our framework can also be used to prove logarithmic lower bounds for other privacy notions beyond differential privacy and obliviousness. In this section, we consider lower bounds for data structures that provide guarantees on upper bounds on leakage functions. We note this is a standard approach to proving privacy for searchable encryption schemes [CGKO06].

Patel *et al.* [PPY20] proved lower bounds for encrypted multi-maps that guarantee leakage at most the *decoupled key-equality leakage* pattern $\mathcal{L}_{\mathsf{DecKeyEq}}$. This leakage reveals whether two queries (or two updates) operations occur for the same key. However, this leakage does not reveal whether a query and an update operation occur on the same key. In particular, they showed such data structures must have overhead $\Omega(b/\omega \cdot \log(nb/c))$ for multi-maps that can store values of b bits. Once again, there remains the possibility that sub-logarithmic overhead is possible when $b \ll \omega$. Using our framework, we show that logarithmic overhead is still required. We refer to $(\mathcal{L}, \epsilon, 1, k)$-secure as a data structure with leakage at most $\mathcal{L}$, adversarial advantage at most ϵ for a PPT adversary that compromises one of k servers.

We start by presenting a definition of security with respect to leakage. We use the same definition used for the lower bound proofs in [PPY20], which are weaker than standard simulation definitions but result in stronger lower bounds.

Definition 6. *A data structure* DS *is $(\mathcal{L}, \beta, 1, k)$-secure if for any pair of operational sequences O_1 and O_2 with the same leakage, $\mathcal{L}(O_1) = \mathcal{L}(O_2)$, and any PPT adversary $\mathcal{A}$ that compromises one* *of the k servers, the following holds:*

$$|\Pr[\mathcal{A}(\mathcal{T}_{\mathsf{DS}}(O_1)) = 1] - \Pr[\mathcal{A}(\mathcal{T}_{\mathsf{DS}}(O_2)) = 1]| \leq \beta$$

where $\mathcal{T}_{\mathsf{DS}}(O)$ is the transcript seen by the adversary across all compromised servers when the operational sequence O is executed by DS.

Next, we present the definition of decoupled key-equality leakage. At a high level, this leakage specifies reveals two matrices that reveal whether two query (or two update) operations occur for the same key. However, this leakage does not reveal whether a query and an update operation occur on the same key. This is a slight weakening of the key-equality leakage that appears in efficient (sub-logarithmic) searchable encryption schemes. We take the definition from [PPY20].

Definition 7. *The* decoupled key-equality leakage $\mathcal{L}_{\mathsf{DecKeyEq}}$ *for any operational sequence O is the two $|O| \times |O|$ matrices M_U and M_Q defined as follows:*

$$M_U[i][j] = \begin{cases} \perp & \textit{if at least one of the } i\textit{-th or } j\textit{-th operation is a query} \\ 0 & \textit{if the } i\textit{-th and } j\textit{-th update are for different keys} \\ 1 & \textit{if the } i\textit{-th and } j\textit{-th update are for the same key} \end{cases}$$

and M_Q is defined identically swapping the role of updates and queries.

Similar to differential privacy, we present a generic lemma showing that any data structure that is secure with respect to $\mathcal{L}_{\mathsf{DecKeyEq}}$ will satisfy the second conditions of our frameworks in Theorem 3 and Theorem 4.

Lemma 6. *Suppose a data structure $(\mathcal{L}_{\mathsf{DecKeyEq}}, \beta/k, 1, k)$-secure for some sufficiently small constant $\beta > 0$ with k servers and the PPT adversary compromises one server. Let $E(U, q)$ be some event that can be checked whether to have occurred by a PPT adversary for some update sequence U and query q such that $\Pr[E(U, q)] \geq \zeta/k$ for some constant $\zeta > 0$. Then, for any update sequence U and any pair of queries q and q', it must be that*

$$\Pr[E(U, q')] = \Omega(\Pr[E(U, q)])$$

where the probability is over the internal randomness of the data structure.

Proof. By definition of the leakage function $\mathcal{L}_{\mathsf{DecKeyEq}}$ and the fact that U only consists of update sequences, we know that $\mathcal{L}_{\mathsf{DecKeyEq}}((U, q)) = \mathcal{L}_{\mathsf{DecKeyEq}}((U, q'))$. As the data structure is $(\mathcal{L}_{\mathsf{DecKeyEq}}, \beta)$-secure and E is an event observable by a PPT adversary, we know that

$$|\Pr[E(U, q')] - \Pr[E(U, q)]| \leq \beta/k.$$

Suppose that we set $\beta = \zeta/2$, which is a constant since ζ is a constant. Then, we immediately see that $\Pr[E(U, q')] \geq \zeta/(2k) \geq \Pr[E(U, q)]/2$. □

Using the above lemma, we can complete the lower bound proof for encrypted multi-maps. We note the above lemma can also be used to prove lower bounds for any data structures in our paper with respect to $\mathcal{L}_{\mathsf{DecKeyEq}}$ leakage.

Theorem 9. *Any* $(\mathcal{L}_{\mathsf{DecKeyEq}}, \beta/k, 1, k)$*-secure data structure* DS *that solves the dynamic multi-map problem for* n b*-bit entries for a sufficiently small constant* $\beta > 0$*, expected update and query time* t_u *and* t_q*, client storage* c *and error probability* $\alpha \leq 1/\log^2 n$ *in the cell probe model with* $\omega \geq 1$ *cell size must satisfy:*

$$t_u + t_q = \Omega\left(\frac{\log(nb/c)}{1 + \log((\omega + \log\log n)/b)}\right).$$

Proof. Fix any set of n different keys $k_1, \ldots, k_n$. The random update sequence $\mathbf{U} = (\mathbf{u}_1, \ldots, \mathbf{u}_n)$ is generated in the following way. For each $\mathbf{u}_i$, we generate a uniformly random b-bit block $\mathbf{B}_i$ from $\{0,1\}^b$, and $\mathbf{u}_i$ inserts key-value pair $(k_i, \mathbf{B}_i)$. For any sequence of consecutive updates $\mathbf{u}_a, \ldots, \mathbf{u}_{a+\ell-1}$, we construct the query set $\{k_a, \ldots, k_{a+\ell-1}\}$. As the queries successfully read back $\mathbf{B}_a, \ldots, \mathbf{B}_{a+\ell-1}$, we get that

$$H(\mathbf{A}(\mathbf{U}, q) \mid \mathbf{u}_1, \ldots, \mathbf{u}_{a-1}, \mathbf{u}_{a+\ell}, \ldots, \mathbf{u}_n) \geq \ell \cdot b.$$

We apply Theorem 4 as the above satisfies the first condition and Lemma 6 satisfies the second condition. By setting $b = v$, we obtain the theorem. □

$\mathcal{L}_{\mathsf{DecKeyEq}}$ **Lower Bounds.** Similar to differential privacy, we can prove a generic result for $\mathcal{L}_{\mathsf{DecKeyEq}}$ leakage with respect to the Event Transfer Probability property. As a result, we can prove lower bounds for $\mathcal{L}_{\mathsf{DecKeyEq}}$-secure versions for sets, predecessor and union-find. We omit further details as they follow as straightforward applications of our framework.

6 Constructions for Oblivious Stacks and Queues

We show that it is possible to construct an oblivious stack (queue) with sub-logarithmic overhead. by showing one can speed up oblivious stacks (queues) by a multiplicative b/ω factor. This gives a separation result showing that, when $b \ll \omega$, oblivious stacks (queues) are inherently faster than ORAMs. Our construction will match the $\Omega(b/\omega \cdot \log(nb/c))$ lower bound in [JLN19].

Construction. We now describe our oblivious stack construction. It can be modified in a straightforward manner to also obtain oblivious queues or deques. Our construction of an oblivious stack of at most n elements of size b with a server with word size w will make black-box use of any ORAM Π with blocks of length $b' = \omega$. The ORAM will store at most $N = O(n \cdot (b/\omega))$ blocks each containing $L := \omega/b$ stack elements. We can now consider two settings depending on the values of b and ω. When $b < \omega$, $L > 1$ signifies that each ORAM block stores multiple stack elements. For $b \geq \omega$, $L \leq 1$ signifies that a stack element is spread over one or more ORAM blocks. Assuming one-way functions, there exist ORAMs with $O(\log(Nb'/c)) = O(\log(nb/c))$ query overhead and $O(c)$ client storage when the block size is equal to the word size [AKL+20].

At a high level, the client will store an integer counter C describing the total number of blocks currently stored in the stack to keep track of the location of the stack top. For the case when $b \geq \omega$, we can directly use the above ORAM as an oblivious stack and each stack operation will involve b/ω ORAM operations. The value of C keeps a pointer to where these operation must occur.

Let us now focus on the case when $b < \omega$ and each ORAM block thus contains $L = \omega/b$ stack blocks. The idea is to break up stack operations into groups of L operations. To locally handle the L operations of a group, we make sure that, at the start of a group of operations, the client local memory contains the L elements at the top of the stack. As it is easily seen, this is all the information needed to perform a group of L operations and at the end of a group, the local client memory holds at most $2L$ stack elements (this happens if all L operations are push operations). The client thus performs the write of at most $2L$ stack elements back to the ORAM. Since each ORAM block contains $L = \omega/b$ blocks, this can be accomplished by 3 ORAM write operations, as $2L$ stack elements could spread over 3 ORAM blocks. Not to break obliviousness, the client performs 3 writes even if the stack elements found in client memory at the end of a group happen to belong to fewer ORAM blocks. Following this and to prepare for the next group, the client reads the top L elements of the stack from the ORAM and this can be accomplished by reading 2 ORAM blocks. We present the formal construction below.

$\mathsf{Init}(1^\lambda, n)$:

```
1. $\mathsf{st}_\Pi \leftarrow \Pi.\mathsf{Init}(1^\lambda, n)$.
2. Set $S$ to be an empty stack of at most $2\omega/b$ blocks each of size $b$ bits.
3. Set $\mathsf{cnt}_G \leftarrow 0$ to be the number of operations completed within a group.
4. Set $\mathsf{cnt}_\Pi \leftarrow 0$ to be the number of blocks occupied in $\Pi$.
5. Return $\mathsf{st} \leftarrow (\mathsf{st}_\Pi, \mathsf{cnt}_G, \mathsf{cnt}_\Pi, S)$.
```

$\mathsf{Refresh}(\mathsf{st})$:

```
1. If $|Q| \leq \omega/b$:
    (a) If $\mathsf{cnt}_\Pi > 0$:
        i. Set $\mathsf{cnt}_\Pi \leftarrow \mathsf{cnt}_\Pi - 1$.
        ii. Execute $B' \leftarrow \Pi.\mathsf{Query}(\mathsf{st}_\Pi, \mathsf{cnt}_\Pi)$.
        iii. Interpret $B'$ as $\omega/b$ $b$-bit blocks and push them to the bottom of the local stack $S$.
    (b) Else when $\mathsf{cnt}_\Pi = 0$, execute $\Pi.\mathsf{Query}(\mathsf{st}_\Pi, 0)$ and ignore result.
2. Else when $|Q| > \omega/b$:
    (a) Remove the $\omega/b$ block from the bottom of local stack $S$ and arrange into a ORAM block $B'$ of $\omega$ bits.
    (b) If $\mathsf{cnt}_\Pi \geq nb/\omega$:
        i. Return an error as more than $n$ blocks are being stored in the stack.
    (c) Execute $\Pi.\mathsf{Update}(\mathsf{st}_\Pi, \mathsf{cnt}_\Pi, B')$.
    (d) Set $\mathsf{cnt}_\Pi \leftarrow \mathsf{cnt}_\Pi + 1$.
3. Set $\mathsf{cnt}_G \leftarrow 0$.
```

$\mathsf{Push}(\mathsf{st}, B)$:

```
1. Push the $b$-bit block $B$ to the top of the local stack $S$.
2. Set $\mathsf{cnt}_G \leftarrow \mathsf{cnt}_G + 1$.
```

3. If $\mathsf{cnt}_G = \omega/b$, execute $\mathsf{Refresh}(\mathsf{st})$.

$\mathsf{Pop}(\mathsf{st})$:

1. Pop the b-bit block B from the top of the local stack S.
2. Set $\mathsf{cnt}_G \leftarrow \mathsf{cnt}_G + 1$.
3. If $\mathsf{cnt}_G = \omega/b$, execute $\mathsf{Refresh}(\mathsf{st})$.
4. Return B.

Theorem 10. *Assuming one-way functions, the above construction is an oblivious stack for block size $b \geq 1$ and word size $\omega \geq 1$ with client storage $c = O(\omega + \log n)$ bits, server storage $O(n \cdot b)$ bits and amortized overhead $O(b/\omega \cdot \log(nb/c))$.*

Proof. First, we analyze the efficiency of our construction. We note that we perform $O(1)$ ORAM operations every $L = \omega/b$ stack operations. Our underlying ORAM requires $O(\log(nb/c'))$ overhead if we dedicate c' bits of client storage for the ORAM. Therefore, the amortized overhead of the construction is $O(1/L \cdot \log(nb/c')) = O(b/\omega \cdot \log(nb/c'))$. The client memory of the stack stores a constant number of integers using $O(\log n)$ bits to track of the size of the current size of the stack as well as $O(1)$ cells or $O(\omega)$ bits to handle operations within each group of L operations. Therefore, the total client memory is $c = O(\log n + \omega + c')$. If we set $c' = \Theta(\log n + \omega)$, we get that the client storage is $O(\omega + \log n)$ and the amortized overhead is $O(b/\omega \cdot \log(nb/c))$ as $c' = \Theta(c)$.

Next, we consider the obliviousness of our stack. We note the schedule of ORAM operations are pre-determined and independent of the operations performed by the stack. In particular, exactly 2 ORAM operations are performed before every group of L stack operations and exactly 3 ORAM operation following every group of L stack operations. Using the obliviousness of the underlying ORAM, our stack construction is also oblivious. □

Stacks and Our Framework. One way to view this result is that our framework is tight in the sense that data structures that do not satisfy our framework may be implemented more efficiently. In particular, stacks, queues and dequeues are data structures that do not enjoy the Large Information Retrieval property as they cannot arbitrarily retrieve information about updates that occur in the middle of an update sequence and thus Theorem 3 does not apply. This ends up being fundamentally inherent since, as we show in this section, oblivious stacks and queues have running times smaller than the logarithmic lower bounds proved previously.

Sub-Logarithmic and Sub-Constant Overhead. Our construction in Theorem 10 shows that sub-logarithmic overhead is obtainable from oblivious stacks (a similar idea may be adapted for oblivious queues). For example, if we consider the setting where $b = O(1)$ and $\omega = \Theta(\log n)$, then our construction has $O(1)$ overhead. Surprisingly, our construction can even obtain sub-constant amortized overhead. For example, if $b = O(1)$ and $\omega = \Theta(\log^2 n)$, our construction uses amortized overhead $O(1/\log n)$. In other words, the client needs to interact with the server every $O(1/\log n)$ operations on average.

Separation from ORAMs. We note that the above construction allows us to separate RAMs from stacks and queues. When $b = \Theta(\omega)$, the lower bounds in [JLN19] show that logarithmic overhead is tight for all of RAMs, stacks and queues when obliviousness is required. On the other hand, for $b \ll \omega$, our construction shows that sub-logarithmic (and even sub-constant) amortized overhead is possible for oblivious stacks and queues. In contrast, ORAMs still require logarithmic overhead when $b \ll \omega$ as shown in [KL21].

7 Conclusions

In this work, we present logarithmic lower bounds for differentially private data structures for all parameter settings of block sizes b and cell sizes ω. This improves upon the prior lower bounds proved in [PY19] for the setting of $b \ll \omega$ and answers an open question posed in [KL21]. Our lower bounds apply for differentially private RAMs, sets, predecessor and disjoint sets (union-find).

Additionally, we present a framework that can be re-used for different data structure problems and privacy guarantees. To try and make our techniques more accessible, we identify two simple, minimal conditions that are required to prove lower bounds in our framework. We reduce proving logarithmic lower bounds to showing that a specific data structure problem and privacy guarantee satisfy the two conditions of our framework. We hope our framework will make it easier to prove lower bounds without unnecessarily customizing techniques.

Acknowledgements. This research was supported in part by the Algorand Centres of Excellence programme managed by Algorand Foundation. Any opinions, findings, and conclusions or recommendations expressed in this material are solely those of the authors.

References

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

[BCP15] Elette Boyle, Kai-Min Chung, and Rafael Pass. Large-scale secure computation: Multi-party computation for (parallel) RAM programs. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 742–762. Springer, Heidelberg, August 2015.

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

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

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

[BNP+15] Vincent Bindschaedler, Muhammad Naveed, Xiaorui Pan, XiaoFeng Wang, and Yan Huang. Practicing oblivious access on cloud storage: the gap, the fallacy, and the new way forward. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 837–849. ACM Press, October 2015.

[CCMS19] T.-H. Hubert Chan, Kai-Min Chung, Bruce M. Maggs, and Elaine Shi. Foundations of differentially oblivious algorithms. In Timothy M. Chan, editor, *30th SODA*, pages 2448–2467. ACM-SIAM, January 2019.

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

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

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

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

[CLP14] Kai-Min Chung, Zhenming Liu, and Rafael Pass. Statistically-secure ORAM with $\tilde{O}(\log^2 n)$ overhead. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 62–81. Springer, Heidelberg, December 2014.

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

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

[DvF$^+$16] Srinivas Devadas, Marten van Dijk, Christopher W. Fletcher, Ling Ren, Elaine Shi, and Daniel Wichs. Onion ORAM: A constant bandwidth blowup oblivious RAM. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part II*, volume 9563 of *LNCS*, pages 145–174. Springer, Heidelberg, January 2016.

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

[GKW18] S. Dov Gordon, Jonathan Katz, and Xiao Wang. Simple and efficient two-server ORAM. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part III*, volume 11274 of *LNCS*, pages 141–157. Springer, Heidelberg, December 2018.

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

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

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

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

[JLS21] Zahra Jafargholi, Kasper Green Larsen, and Mark Simkin. Optimal oblivious priority queues. In Dániel Marx, editor, *32nd SODA*, pages 2366–2383. ACM-SIAM, January 2021.

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

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

[KS21] Ilan Komargodski and Elaine Shi. Differentially oblivious turing machines. In James R. Lee, editor, *ITCS 2021*, volume 185, pages 68:1–68:19. LIPIcs, January 2021.

[Lar12] Kasper Green Larsen. The cell probe complexity of dynamic range counting. In Howard J. Karloff and Toniann Pitassi, editors, *44th ACM STOC*, pages 85–94. ACM Press, May 2012.

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

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

[LO13] Steve Lu and Rafail Ostrovsky. Distributed oblivious RAM for secure two-party computation. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 377–396. Springer, Heidelberg, March 2013.

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

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

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

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

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

[PT07] Mihai Pătraşcu and Corina E. Tarniţă. On dynamic bit-probe complexity. *Theoretical Computer Science*, 380(1-2):127–142, 2007.

[PTW10] Rina Panigrahy, Kunal Talwar, and Udi Wieder. Lower bounds on near neighbor search via metric expansion. In *51st FOCS*, pages 805–814. IEEE Computer Society Press, October 2010.

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

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

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

[Shi20] Elaine Shi. Path oblivious heap: Optimal and practical oblivious priority queue. In *2020 IEEE Symposium on Security and Privacy*, pages 842–858. IEEE Computer Society Press, May 2020.

[SS13] Emil Stefanov and Elaine Shi. ObliviStore: High performance oblivious cloud storage. In *2013 IEEE Symposium on Security and Privacy*, pages 253–267. IEEE Computer Society Press, May 2013.

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

[TVL84] Robert E Tarjan and Jan Van Leeuwen. Worst-case analysis of set union algorithms. *Journal of the ACM (JACM)*, 1984.

[vEB75] Peter van Emde Boas. Preserving order in a forest in less than logarithmic time. In *Symposium on Foundations of Computer Science*, 1975.

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

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

[WNL+14] Xiao Shaun Wang, Kartik Nayak, Chang Liu, T.-H. Hubert Chan, Elaine Shi, Emil Stefanov, and Yan Huang. Oblivious data structures. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014*, pages 215–226. ACM Press, November 2014.

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

[Yeo23] Kevin Yeo. Lower bounds for (batch) pir with private preprocessing. In *Eurocrypt 2023 (to appear)*, 2023.

A Proof of Theorem 4

We start by defining some new random variables with respect to each of the k individual servers where $j \in [k]$ denotes the j-th server. These are summarized below:

- $\mathbf{X}_u^{\geq p_{i-1},j}$: This is the probes of $\mathbf{X}_u^{\geq p_{i-1}}$ that occur on the j-th server.
- $\mathbf{X}_q^{i,j}$: This is the probes of $\mathbf{X}_q^i$ that occur on the j-th server.
- $\mathcal{Z}^j(i,q)$: This is defined identically to $\mathcal{Z}(i,q)$ except using the above random variables for the j-th server. Formally,

$$\mathcal{Z}^j(i,q) = \min\left\{v, \frac{\mathbb{E}[\mathbf{X}_u^{p_{i-1}}](\omega + \log\log n)}{|Q|} + b \cdot \mathbb{E}[\mathbf{X}_q^i]\right\}.$$

Using the above, we can quickly see to that the following are true:

$$\mathbf{X}_u^{\geq p_{i-1}} = \sum_{j=1}^{k} \mathbf{X}_u^{\geq p_{i-1},j} \text{ and } \mathbf{X}_q^{i} = \sum_{j=1}^{k} \mathbf{X}_q^{i,j}.$$

As an aside, we note that these style of equalities do not hold for $\mathcal{Z}^j(i,q)$ due to its non-linearity.

Lemma 7. *Consider a data structure* DS *that satisfies the conditions of Theorem 3. There exists a query q such that the following holds for all large epochs i where $\mathbb{E}[\mathbf{X}_u^{p_{i-1}}] = O(t_u \ell_i/(rK))$:*

$$\sum_{j=1}^{k} \mathcal{Z}^j(i,q) = \Omega(v).$$

Proof. To prove this, we will leverage the second condition of Theorem 4. First, we consider consider a logarithmic number of events for each of the k servers and each query $q \in Q$. In particular, for each $j \in [k]$, $\beta \in [\log(v/48)]$ and $q \in Q$, we define the following event:

$$\mathbf{E}_q^{i,j,\beta} = 1 \iff \frac{\mathbf{X}_u^{\geq p_{i-1},j}}{|Q|}(\omega + \log\log n) + b \cdot \mathbf{X}_q^{i,j} \in [2^\beta, 2^{\beta+1}).$$

Additionally, we define the following event for $\beta = \log(v/48)$:

$$\mathbf{E}_q^{i,j,\beta} = 1 \iff \frac{\mathbf{X}_u^{\geq p_{i-1},j}}{|Q|}(\omega + \log\log n) + b \cdot \mathbf{X}_q^{i,j} \geq 2^\beta = v/48.$$

We will split the proof into two cases. For each large epoch i where $\mathbb{E}[\mathbf{X}_u^{p_{i-1}}] = O(t_u \ell_i/(rK))$, suppose there exists a query $q_i \in Q$ such that

$$\sum_{j=1}^{k} \Pr[\mathbf{E}_{q_i}^{i,j,\log(v/48)} = 1] \geq p$$

for some probability $p > 0$ that we will set later. Using the second condition of Theorem 4, we note that for all $j \in [k]$ and any query $q \in Q$,

$$\Pr[\mathbf{E}_q^{i,j,\log(v/48)} = 1] \geq \nu \cdot \Pr[\mathbf{E}_{q_i}^{i,j,\log(v/48)} = 1])$$

for some constant $\nu > 0$ whenever $\Pr[\mathbf{E}_{q_i}^{i,j,\log(v/48)} = 1] \geq \zeta/k$. Therefore, the loss is at most ζ/k whenever the probability is less than ζ/k and at most a ν fraction whenever the probability is greater than ζ/k when moving to q for each event. We can obtain the following inequality

$$\sum_{j=1}^{k} \Pr[\mathbf{E}_q^{i,j,\log(v/48)} = 1] \geq \sum_{j=1}^{k} \Pr[\mathbf{E}_{q_i}^{i,j,\log(v/48)} = 1] - k \cdot (\zeta/k) \geq \nu/2 - \zeta > \nu/4$$

assuming $\zeta \leq \nu/4$. Then, we can apply linearity of expectation to get that

$$\mathbb{E}\left[\sum_{j=1}^{k} \mathcal{Z}^j(i,q)\right] \geq (\nu/4) \cdot v/48 = \Omega(v)$$

that completes the proof for this case.

We now consider the other case. In this setting, for every large epoch i where $\mathbb{E}[\mathbf{X}_u^{p_{i-1}}] = O(t_u \ell_i/(rK))$, we get that for every query $q_i \in Q$,

$$\sum_{j=1}^{k} \Pr[\mathbf{E}_{q_i}^{i,j,\log(v/48)}] < p$$

for some probability $p > 0$ that we will set later. By the first part of Lemma 3 before applying the Event Probability Transfer property, we know that there exists some $q_i \in Q$ such that

$$\mathbb{E}\left[\frac{\mathbf{X}_u^{\geq p_{i-1}}(\omega + \log\log n)}{|Q|} + b \cdot \mathbf{X}_{q_i}^{i}\right] = \Omega(v).$$

By linearity of expectation, this immediately implies that

$$\sum_{j=1}^{k} \left(\frac{\mathbb{E}[\mathbf{X}_u^{\geq p_{i-1},j}](\omega + \log\log n)}{|Q|} + b \cdot \mathbb{E}[\mathbf{X}_{q_i}^{i,j}] \right) = \Omega(v) \geq \eta v$$

for some constant $\eta > 0$. Towards a contradiction, suppose that for every $\beta \in [\log(v/48)]$:

$$\sum_{j=1}^{k} \Pr[\mathbf{E}_{q_i}^{i,j,\beta} = 1] < \eta.$$

Then, we get the following by setting $p \leq \eta/48$:

$$\begin{aligned}
&\sum_{j=1}^{k} \left(\frac{\mathbb{E}[\mathbf{X}_u^{\geq p_{i-1},j}](\omega + \log\log n)}{|Q|} + b \cdot \mathbb{E}[\mathbf{X}_{q_i}^{i,j}] \right) \\
&\quad \leq \left(\sum_{j=1}^{k} \sum_{\beta=0}^{\log(v/48)-1} 2^{\beta+1} \cdot \Pr[\mathbf{E}_{q_i}^{i,j,\beta} = 1] \right) + \left(\sum_{j=1}^{k} \Pr[\mathbf{E}_{q_i}^{i,j,\log(v/48)}] \cdot v \right) \\
&\quad \leq \left(\sum_{\beta=0}^{\log(v/48)-1} \eta \cdot 2^{\beta+1} \right) + \eta \cdot (v/48) \\
&\quad \leq \eta \cdot (v/24) + \eta \cdot (v/48) \\
&\quad \leq \eta \cdot (v/12).
\end{aligned}$$

This contradicts the previous inequality meaning that it must be that for every $\beta \in [\log(v/48)]$:

$$\sum_{j=1}^{k} \Pr[\mathbf{E}_{q_i}^{i,j,\beta} = 1] \geq \eta.$$

For the last step, we use the second condition of Theorem 4 in a similar way as the previous case. If we consider any query $q \in Q$ and any $\beta \in [\log(v/48)]$, then

$$\sum_{j=1}^{k} \Pr[\mathbf{E}_{q_i}^{i,j,\beta} = 1] \geq \eta - k \cdot (\zeta/k) \geq \nu\eta - \zeta \geq \nu\eta/2$$

assuming that $\zeta < \nu\eta/2$. Finally, we complete the proof by seeing that

$$\begin{aligned}\sum_{j=1}^{k}\left(\frac{\mathbb{E}[\mathbf{X}_u^{\geq p_{i-1},j}](\omega+\log\log n)}{|Q|}+b\cdot\mathbb{E}[\mathbf{X}_{q_i}^{i,j}]\right) &\geq \sum_{j=1}^{k}\sum_{\beta=0}^{\log(v/48)-1}\Pr[\mathbf{E}_{q_i}^{i,j,\beta}=1]\cdot 2^{\beta}\\ &\geq \sum_{\beta=0}^{\log(v/48)-1}(\nu\eta/2)\cdot 2^{\beta}\\ &\geq (\nu\eta/2)\cdot(v/96).\end{aligned}$$

In other words, the sum of the second arguments of $\mathcal{Z}^j(i,q)$ for all $j \in [k]$ is at least $\Omega(v)$ as both ν and η are positive constants. This immediately implies the statement of the lemma to complete the proof. □

Proof of Theorem 4. By Lemma 7, we get that for all large epochs i such that $\mathbb{E}[\mathbf{X}_u^{p_{i-1}}] = O(t_u\ell_i/(rK))$ the following holds:

$$\mathbb{E}[\mathcal{Z}(i,q)] \geq \mathbb{E}\left[\sum_{j=1}^{k}\mathcal{Z}^j(i,q)\right] = \Omega(v).$$

The first argument of each $\mathcal{Z}^j(i,q)$ is at least $\Omega(v)$. The above inequality implies the sum of the second arguments must be at least $\Omega(v)$. Therefore, we get that, by linearity of expectation,

$$\frac{\mathbb{E}[\mathbf{X}_u^{p_{i-1}}]}{|Q|}(\omega+\log\log n)+b\cdot\mathbb{E}[\mathbf{X}_q^{i}] = \sum_{j=1}^{k}\left(\frac{\mathbb{E}[\mathbf{X}_u^{p_{i-1},j}]}{|Q|}(\omega+\log\log n)+b\cdot\mathbb{E}[\mathbf{X}_q^{i,j}]\right) = \Omega(v).$$

This is the exact inequality that was required to complete the proof of Theorem 3. Therefore, by repeating the same steps, we can obtain the same lower bound to complete the proof. □