Leakage Cell Probe Model: Lower Bounds for Key-Equality Mitigation in Encrypted Multi-Maps

Sarvar Patel* Giuseppe Persiano† Kevin Yeo‡

Abstract

Encrypted multi-maps (EMMs) enable clients to outsource the storage of a multi-map to a potentially untrusted server while maintaining the ability to perform operations in a privacy-preserving manner. EMMs are an important primitive as they are an integral building block for many practical applications such as searchable encryption and encrypted databases. In this work, we formally examine the tradeoffs between privacy and efficiency for EMMs.

Currently, all known dynamic EMMs with constant overhead reveal if two operations are performed on the same key or not; that is, they leak the *global key-equality pattern*. In our main result, we present strong evidence that the leakage of the global key-equality pattern is inherent for any dynamic EMM construction with $O(1)$ efficiency. In particular, we consider the slightly smaller leakage of *decoupled key-equality pattern* where leakage of key-equality between update and query operations is decoupled and the adversary only learns whether two operations of the *same type* are performed on the same key or not. We show that any EMM with at most decoupled key-equality pattern leakage incurs $\Omega(\lg n)$ overhead in the *leakage cell probe model*. This is tight as there exist ORAM-based constructions of EMMs with logarithmic slowdown that leak no more than the decoupled key-equality pattern (and actually, much less). Furthermore, we present stronger lower bounds that encrypted multi-maps leaking at most the decoupled key-equality pattern but are able to perform one of either the update or query operations in the plaintext still require $\Omega(\lg n)$ overhead. Finally, we extend our lower bounds to show that dynamic, *response-hiding* searchable encryption schemes must also incur $\Omega(\lg n)$ overhead even when one of either the document updates or searches may be performed in the plaintext.

1 Introduction

In this work, we study *encrypted multi-maps* [17, 40], an example of structured encryption (see Chase and Kamara [16]). Structured encryption considers the problem of a client that wishes to outsource the storage of an encrypted data structure to an untrusted server in a privacy-preserving manner. In addition, the structured encryption scheme must enable the client to perform operations over the encrypted, outsourced data structure in an efficient manner. For privacy, the goal is simply to reveal as little information as possible about the data structure as well as the performed operations.

Encrypted multi-maps (EMMs) are a specific structured encryption scheme for outsourcing *multi-maps*. For multi-maps, a client is able to update the tuple of values associated with a key as well as query for the value tuple associated with any key. In this paper, we focus on encrypted multi-maps due to its many important practical applications. Two examples of applications are searchable encryption and encrypted databases. The construction of private and efficient encrypted multi-maps is an important problem to enable the deployment of these privacy-preserving applications in the real-world.

Searchable encryption (also known as encrypted search) was first introduced by Song *et al.* [65] and has been a well studied topic in the past couple decades (see [28, 7, 17, 4, 43, 14, 13, 15, 66, 2, 10, 54, 11, 19, 38, 58, 3, 18, 41, 40] as some examples). The representative scenario for searchable encryption considers

*`sarvar@google.com`. Google.
†`giuper@gmail.com`. Università di Salerno.
‡`kwlyeo@google.com`. Google.

a client that owns a large corpus of documents and an untrusted server with large amounts of available storage. The goal of searchable encryption is to enable the client to outsource the storage of the document corpus to the server. For functionality, the client wishes to maintain the ability to efficiently search over the corpus and retrieve the identifiers of all documents containing a specific keyword as well as update documents by inserting, deleting and/or modifying keywords. In terms of privacy, the client wishes to keep any information related to the contents of the document corpus and the queries hidden from the server. In many works, searchable encryption schemes utilize encrypted multi-maps as their main building block to map keywords to documents that contain the keyword. We note that various searchable encryptions schemes have utilized encrypted multi-maps in other, more sophisticated, manners as well.

Another important application is encrypted databases. In this problem, the goal is to encrypt and outsource a database while enabling the database owner to privately perform database operations. Earlier works on encrypted databases [62] utilized property-preserving encryption schemes such as deterministic [4] and order-preserving encryption [5, 6, 8, 52]. It has been shown that encrypted databases built from property-preserving encryption may have security vulnerabilities [55]. In the most recent work, a scheme for encrypting SQL databases was presented by Kamara and Moataz [39] utilizing encrypted multi-maps instead of property-preserving encryption.

Due to these applications, the problem of constructing both efficient and private encrypted multi-maps is very important. Unfortunately, the only way that is currently known to achieve very strong levels of privacy is using very expensive cryptographic primitives such as oblivious RAM [30] and/or fully homomorphic encryption [26]. These schemes only leak the size of inputs and outputs of operations, which can also be mitigated by using techniques from recent volume-hiding schemes [40, 60]. However, the large performance overheads of these expensive cryptographic primitives preclude them from being used in practical applications. Instead, structured encryption schemes take a different approach by slightly relaxing privacy requirements with the hope of improving efficiency. In particular, the privacy of searchable encryption schemes is parameterized by a *leakage* function. The leakage function is an upper bound on the information revealed to the adversarial server when processing queries over a stored document corpus. Therefore, the design of encrypted multi-map schemes consists of minimizing the leakage function while ensuring the overhead is as small as possible. Using this relaxed variant of privacy, several dynamic encrypted multi-map schemes such as [17, 43] with constant overhead have been presented. However, all these schemes have shown to have non-trivial leakage including the *global key-equality pattern* that enables the adversary to learn whether two multi-map operations are performed on the same key or not.

On the other hand, there has been a long line of work starting with the paper of Islam *et al.* [36] that evaluate the negative privacy consequences of various leakage profiles. Using various and continuously improving frequency analysis and statistical learning methods [12, 55, 63], it has been shown that the contents of documents and/or the queried keywords may be compromised by using *access pattern leakage* that shows whether a specific memory location is accessed by different queries or not. These ideas are further extended to present attacks on schemes that enable clients to perform range queries in [46, 32]. In another line of work, Zhang *et al.* [70] consider the scenario where adversaries may inject files into encrypted search schemes. By carefully arranging keywords in the injected files, it is shown that viewing the identifiers of matching injected documents of any query enables the adversary to determine the queried keyword with perfect accuracy. Finally, a recent work by Kornaropoulos *et al.* [45] show new non-parameteric estimation techniques to utilize global key-equality pattern leakage to compromise privacy in certain settings.

Therefore, it is important to ensure that encrypted multi-map constructions are both efficient (to be deployable in practice) as well as only leak small amounts of information (to ensure privacy is not compromised). To our knowledge, all previous work either present a construction with some efficiency and a specific leakage profile or present an attack that compromises privacy with a specific leakage profile. There has been no work that formally analyzes the tradeoffs in efficiency and leakage that appear in encrypted multi-maps. In this work, we present the first formal tradeoffs between privacy and efficiency for encrypted multi-maps.

1.1 Our Results

In this section, we present our lower bounds in the *leakage cell probe model*. We start by focusing on encrypted multi-maps. Afterwards, we move onto dynamic searchable encryption schemes.

To start, we briefly describe how the efficiency of schemes in the leakage cell probe model is measured. Typically, data structures measure efficiency amortized over the number of operations. This approach cannot be used for data structures that may return outputs of varying sizes. As a concrete example, let us consider *multi-maps*. Roughly speaking, a multi-map is a data structure that maintains a sequence of pairs $(\texttt{key}, \texttt{vals})$, where $\texttt{key}$ is taken from a *key universe* $\mathcal{K}$ and $\texttt{vals}$ is a tuple of varying length of values from a *value universe* $\mathcal{V}$. A multi-map supports $\mathsf{Get}(\texttt{key})$ operations, that return the tuple associated with $\texttt{key}$, and $\mathsf{Add}(\texttt{key}, \texttt{val})$ operations, that add value $\texttt{val} \in \mathcal{V}$ to the tuple associated with $\texttt{key}$. So two Get operations might return tuples of values of vastly different sizes and thus cannot be expected to incur the same costs. So, we measure the *query efficiency* as the amount of server computation per *returned value*. The problem does not occur for Add updates operations as they operate on a single value and thus we can consider the *update efficiency* as the amount of server computation per Add operation. The efficiency of a dynamic scheme is the maximum of the update and query efficiency.

Encrypted multi-maps. We start by describing our results for encrypted multi-maps and we note our results also apply to encrypted arrays (which can be interpreted as oblivious RAMs with larger leakage). The efficiency of encrypted multi-maps crucially depends on the leakage one is willing to tolerate. If no security is sought and each operation may completely leak its inputs, the multi-map problem is identical to the classic dynamic dictionary problem (see [56] for a survey). One can obtain constructions of *plaintext* multi-maps with constant amortized efficiency by utilizing, for example, the optimal dynamic perfect hashing scheme in [21]. In this case, all operations are performed in the plaintext and the inputs and outputs of all operations are revealed.

At the other hand of the leakage spectrum, there exist folklore solutions of encrypted multi-maps with minimal leakage that can be obtained by using efficient ORAMs [57, 1] while achieving logarithmic overhead for each updated value in update operations and for each returned value in query operations. In particular, these folklore solutions only leak the number of values (volume) associated with the queried key and nothing else. For completeness, we present a formal definition of this minimal leakage function as well as a description and a proof of the folklore solution in Appendix A.

In this work, we are interested in understanding the transition from constant to logarithmic amortized efficiency as a function of the leakage allowed. In particular, we attempt to identify the smallest leakage where $O(1)$ overhead solutions still exist. Furthermore, we want to find the largest leakage where constructions must incur asymptotically larger than constant overhead. Specifically, we start by observing that non-trivial leakage can be obtained with constant amortized efficiency by using a simple *hash-and-encrypt* approach. We start from the construction of plaintext multi-maps based on any dynamic perfect hashing scheme such as the one by Dietzfelbinger *et al.* [21]. During the initialization of the encrypted multi-map, the client randomly selects a key K_1 for a collision resistant hash function $\mathcal{H}$ and a random encryption key K_2 for an IND-CPA symmetric encryption scheme $(\mathcal{E}, \mathcal{D})$. For each $\mathsf{Add}(\texttt{key}, \texttt{val})$ operation, the client executes the algorithm for the insertion operation for the dynamic perfect hashing scheme with the hashed value $\mathcal{H}(K_1, \texttt{key})$ as the key and an encryption $\mathcal{E}(K_2, \texttt{val})$ of the value being added. A query operation $\mathsf{Get}(\texttt{key})$ is implemented by executing the query algorithm of the dynamic perfect hashing scheme using $\mathcal{H}(K_1, \texttt{key})$ as a key and then decrypting all the returned values with the IND-CPA key K_2. As a result, the client is successfully able to retrieve all plaintext values associated with the queried $\texttt{key}$. We note that the hash-and-encrypt method is not novel and implicitly appeared in many previous works such as [17, 43] to name a few examples.

The above implementation provides some privacy for the inserted and queried keys and values. In particular, the hash-and-encrypt version of dynamic perfect hashing does not leak the keys and values in the plaintext. However, the adversarial server learns the type of operation performed as well as the number of encrypted values returned by a Get operation. Additionally, the server learns whether two different operations are performed on the same key or not as the server learns the value $\mathcal{H}(K_1, \texttt{key})$ when either performing a Get or Add operation. We denote this leakage, $\mathcal{L}_{\mathsf{glob}}$, as the *global key-equality pattern* that describes whether two operations are given the same key as input or not. We refer readers to Appendix B for a formal description and analysis of the hash-and-encrypt compiler when applied to dynamic perfect hashing.

The above simple hash-and-encrypt construction provides a baseline of what privacy may be efficiently implemented with $O(1)$ overhead. A natural next step is to try and improve the privacy of the above scheme without incurring significantly larger overhead. A slight improvement in privacy would be to consider the leakage function $\mathcal{L}_{\mathsf{dec}}$ which allows the adversary to learn the equality pattern on keys but only for operations of the same type. In more detail, the adversary still learns whether two Get operations are on the same key or not as well as whether two Add operations are on the same key or not. However, the adversary cannot link an Add operation and a Get operation as operating on the same key. We denote this leakage $\mathcal{L}_{\mathsf{dec}}$ as the *decoupled key-equality pattern* (see Section 3 for a formal definition) as it *decouples* the Add key-equality pattern from the Get key-equality pattern. From a quick glance, this small improvement in privacy seems insignificant. In the main result of our work, we show that any encrypted multi-map that leaks at most the decoupled key-equality pattern must incur logarithmic overhead.

Theorem 1 (Informal)**.** *Let* $\mathbf{DS}$ *be a* $\mathcal{L}_{\mathsf{dec}}$*-leakage encrypted multi-map that leaks at most the decoupled key-equality pattern. Then the amortized efficiency of* $\mathbf{DS}$ *must be* $\Omega(\lg n)$ *per updated and/or returned value.*

In other words, our results show that the global key-equality pattern is an inherent and seemingly necessary leakage for any $O(1)$ efficiency encrypted multi-map. By attempting to mitigate the global key-equality pattern even in an extremely small (seemingly meaningless) manner, the resulting encrypted multi-maps must incur logarithmically higher efficiency. As a result, one must either tolerate the leakage of the global key-equality pattern or at least logarithmic overhead when implementing encrypted multi-maps. Furthermore, if the mitigation of global key-equality pattern leakage is necessary or logarithmic overhead is tolerable, then the encrypted multi-map construction using oblivious RAMs may be used resulting in minimal leakage. We also note that the bound in Theorem 1 (with formal statement in Theorem 3) is tight in view of the upper bound provided by the ORAM-based construction (see Appendix A).

The proof of the lower bound for $\mathcal{L}_{\mathsf{dec}}$ only relies on the fact that an adversary cannot link an Add and a Get operation as operating on the same key. Note that this property is guaranteed even if one of the two operations completely leaks the inputs on which it operates. For example, the leakage function $\mathcal{L}_{\mathsf{add}}$, that for any Add(`key`, `val`) operation leaks both `key` and `val`, can still be considered as decoupling the Get and Add key-equality patterns. We can strengthen the proof of our main result to show that encrypted multi-maps that only leak the decoupled key-equality pattern but are allowed to perform all Add operations in plaintext must also incur logarithmic overhead. The same holds also for leakage function $\mathcal{L}_{\mathsf{get}}$ in which Get operations are performed in the clear while keeping the key-equality patterns decoupled. These results further reinforce the difficulty of mitigating the global key-equality pattern leakage even when willing to sacrifice privacy in other areas. We refer the reader to Section 5 for more details.

Theorem 2 (Informal)**.** *Let* $\mathbf{DS}$ *be a* $\{\mathcal{L}_{\mathsf{add}}, \mathcal{L}_{\mathsf{get}}\}$*-leakage encrypted multi-map that leaks at most the decoupled key-equality pattern but may perform one of either the* Add *or* Get *operations in the plaintext. Then the amortized efficiency of* $\mathbf{DS}$ *must be* $\Omega(\lg n)$ *per updated and/or returned value.*

Searchable encryption. We can further prove lower bounds for searchable encryption schemes. In particular, one can use a searchable encryption scheme to construct an encrypted multi-map. As a result, the lower bounds follow directly by interpreting the encrypted multi-map leakage functions as searchable encryption leakage functions.

We start by interpreting the notion of the decoupled key-equality pattern for searchable encryption scheme. For this scenario, the adversary may learn whether two distinct searches are performed for the same keyword or not. For two different document insertions, the adversary may learn the number of keywords that appear in the intersection of the two inserted documents (a generalization of key-equality for documents with multiple keywords). However, this keyword-equality knowledge is limited to operations of the same type. The adversary should not learn whether a queried keyword appears in an inserted document or not. As a result, we refer to these searchable encryption schemes as *response-hiding* where the adversary cannot learn the identity of documents that match a queried keyword.

For the static searchable encryption problem where documents are given during initialization and the documents are immutable, there exists response-hiding schemes with $O(1)$ overhead such as [17]. On the other hand, our lower bounds show that the dynamic version of response-hiding schemes require logarithmic overhead. Furthermore, our lower bounds still hold for searchable encryption schemes even when the construction may perform one of either document updates or searches in the plaintext. In more detail, plaintext updates mean the construction can reveal the entirety of the updated document in plaintext. Similarly, plaintext searches mean the scheme can reveal the queried keyword in plaintext. As a result, our results show that dynamic, response-hiding searchable encryption schemes must either leak the matching documents for any search or incur logarithmic efficiency. For more information, see Section 6.

1.2 Overview of Our Techniques

We present an overview of the techniques used to prove our lower bounds. Our lower bounds are proven in the cell probe model which only measures running time by the number of server memory accesses. We refer the reader to Section 2.1 for more details on the cell probe model. We will utilize the *information transfer* of Pătraşcu and Demaine [64], which Larsen and Nielsen [49] used to prove lower bounds for ORAMs. We review their proof which will be our starting point.

The information transfer technique starts by constructing the *information transfer tree* for a given sequence of n operations. The information transfer tree is a complete tree with one leaf node for each of the n operations. Operations are assigned to the leaves in chronological order: the first operation is assigned to the leftmost leaf node, the second operation is assigned to the second leftmost leaf node and so forth. Each cell probe is assigned to at most one node in the tree in the following manner. First, we determine the operation performing the probe and the associated leaf and then the most recent operation that overwrote the probed cell and its associated leaf. If this is the first probe for the cell then the probe is not assigned to any node; otherwise, the probe is assigned to the lowest common ancestor of the two leaves.

Having defined the information transfer tree, we move onto the hard distribution for the ORAM lower bounds in [49]. Fix any internal node v in the tree and consider the subtree rooted at v. The hard distribution for v consists of writing uniformly random strings to unique array indices in the leaves of the left subtree and, subsequently, querying for these array indices in the leaves of the right subtree. To answer the queries correctly, significant amounts of information must be transferred from the left subtree to the right subtree. For sufficiently large subtrees, it can be shown that the majority of this information must be transferred by query operations in the right subtree performing many probes to cells last overwritten by operations in the left subtree. As a result, these probes will be uniquely assigned to the root of the tree, v.

To complete the proof, Larsen and Nielsen [49] use the obliviousness requirements of ORAM. Suppose there exists another sequence of operations of the same length that assigns significantly less cell probes to the internal node v compared to the hard distribution described above. Note, there exists polynomial time algorithms to compute the number of probes assigned to v. Therefore, a computationally bounded adversary can distinguish between the hard distribution for v and the sequence that does not assign enough probes to v. This contradicts obliviousness. Therefore, a large number of probes must be assigned to each node in the tree. As each probe is uniquely assigned to a node, adding the counts over all nodes gives the desired lower bound.

There are two major obstacles for using the information transfer technique to prove lower bounds for multi-maps. The first problem appears because the lower bounds for oblivious RAMs of [49], as well the one for differentially private RAMs of [61], assumes that the stored array entries are chosen as uniformly random strings. Recall that the crux of the information transfer argument shows that the large entropy of the random strings generated independently in the left subtree of a node v must be retrieved by the query operations in the right subtree of v. The natural extension for encrypted multi-maps would be to assume that all values are truly random strings. While this assumption might be appropriate for multi-maps, it is unreasonable for the application of searchable encryption as it would force either the keywords or the document identifiers to be truly random. It is well known in practice that the entropy of keywords is not too large. Similarly, there is no reason that document identifiers are required to be very random. For example, document identifiers could be titles of documents or just generated by a counter. Instead, our lower bounds will derive entropy from the random distribution of values into keys for multi-maps (or, the random distribution of keywords into documents for the searchable encryption application). As an example, let us consider an arbitrary set of values V and keys K. We can view the distribution of the values V to keys K as a bipartite graph with K as the left partition and V as the right partition. An edge exists between a $\texttt{key} \in K$ and $\texttt{val} \in V$ if and only if $\texttt{val}$ is associated with $\texttt{key}$ in the multi-map. The edges are drawn randomly such that the resulting graph is l-left-regular so each key is associated with exactly l values. Consider the scenario where all values in V are inserted according to this randomly chosen bipartite graph. Suppose that queries are performed to all keys in K. The answers to these queries allows one to correctly retrieve the randomly chosen edges of the graph. In other words, the queries perfectly retrieve the entropy of the update operations. Furthermore, our lower bounds do not make assumptions that the keys in K or the values in V are random.

The other and more serious problem arises from the fact that we are attempting to prove lower bounds for encrypted multi-maps that leak significantly more information to the adversary compared to ORAMs. The ORAM lower bound proof of [49] critically uses the fact that the information transfer tree for any two sequences of the same length must be computationally indistinguishable. On the other hand, we will be proving lower bounds for encrypted multi-maps that leak at least the decoupled equality pattern as well as performing one of either the Add or Get operations in the plaintext. As a result, the overwhelming majority of pairs of sequences of encrypted multi-map operations of the same length will have different leakage and, thus, they will be computationally distinguishable to the adversary.

Therefore, we must choose the hard distributions for each node v such that the decoupled equality pattern leakage is the same for the hard distribution of all nodes in the tree. To do this, we will carefully coordinate Get operations and Add performed on the same key. Recall that $\mathcal{L}_{\mathsf{dec}}$ leaks whether two Add operations are performed on the same key as well as whether two Get operations are performed on the same key. To ensure that leakage incurred by Get operations are identical, we choose our hard distribution such that all Get operations are performed on unique keys. As a result, we are able to swap any two Get operations without changing the leakage as long as the number of values returned by both operations are identical. We will arrange Add operations such that each queried key is always associated with exactly $l \geq 1$ values where l is a parameter (one can achieve encrypted arrays by setting $l = 1$). Using the above properties, we construct our hard distribution for each node v. We assign each leaf node in the information transfer tree with two disjoint equal-sized set of keys K_v^{a} and K_v^{g} and a set of values K_v. Furthermore, all assigned key and value sets are pairwise node disjoint. Each leaf node will be associated with $|K_v^{\mathsf{a}}| \cdot l$ Add operations where each key in K_v^{a} is associated with l uniformly random chosen values from V_v. Recall that we can model these random assignments of values to keys as picking a random l-left-regular bipartite graph with K_v^{a} and V_v acting as the left and right partition respectively. Additionally, each leaf node will perform $|K_v^{\mathsf{g}}|$ Get operations for each key in K_v^{g}. We will use this distribution of sequences as our baseline to construct hard distributions for each internal node v in the information transfer tree. Each of these node-specific hard distributions will have the same leakage with respect to the decoupled leakage function $\mathcal{L}_{\mathsf{dec}}$.

Recall that the goal of a hard distribution for node v is to ensure that a large number of cell probes are assigned to v in the information transfer tree. To do this, we should pick a hard distribution that requires queries in the right subtree of v to retrieve large amounts of entropy generated in the left subtree of v. To start, we denote $K^{\mathsf{a}}, K^{\mathsf{g}}$ and V as the union of the sets $K_{v'}^{\mathsf{a}}, K_{v'}^{\mathsf{g}}, V_{v'}$ that are assigned to leaf nodes v' that appear in the left subtree of v. We keep the identical Add and Get operations that appear in the left subtree of v. We modify the Get operations that appear in the right subtree to query keys in K^{a}, which are all the keys updated in the left subtree of v. As a result, the answers to Get operations in the right subtree of v are able to retrieve the random l-left-regular bipartite graph generated in the left subtree of v forcing a large number of cell probes to be assigned to v. Furthermore, our hard distribution for v only swapped the key parameters of Get operations maintaining the same leakage as the baseline hard distribution. By privacy, it must be that a large number of cell probes are assigned to many nodes of the information transfer tree. As a result, we are able to prove lower bounds for the leakage $\mathcal{L}_{\mathsf{dec}}$ that is significantly larger compared to ORAM leakage. Similar ideas can be used to prove lower bounds for the leakage functions $\mathcal{L}_{\mathsf{add}}$ and $\mathcal{L}_{\mathsf{get}}$

which enable schemes to perform one of either the Add or Get operations in the plaintext. We refer the reader to Section 4 for full details on the lower bound.

1.3 Related Works

The notion of searchable encryption was introduced by Song *et al.* [65]. The notion of adaptive security was first presented by Curtmola *et al.* [17]. They also present the first static schemes with $O(1)$ query efficiency. Chase and Kamara [16] present structured encryption, which is a generalization of searchable encryption. Subsequent works study different variants and topics such as dynamic schemes [28, 43, 42, 66, 13], cache locality [15, 54, 2, 19, 18, 3], forward and backward security [66, 10, 11, 68, 22, 27], expressive queries [16, 14, 23, 38], public-key operations [7] and multiple users [17, 34, 58]. Several works investigate the implications of leakage in searchable encryption [36, 12, 55, 70, 44, 63, 33, 46, 32]. Recently, there have been several searchable encryption schemes presented, which use the oblivious RAM (ORAM) primitive introduced by Goldreich and Ostrovsky [29, 30]. ORAM enables a client to access a server-stored array without revealing either the array contents or the indices updated or retrieved. Garg *et al.* [25] present a static, response-reveaving scheme leveraging ideas from garbled RAMs (an extension of ORAM) first introduced by Lu and Ostrovsky [53]. Kamara *et al.* [41] present a dynamic, response-hiding scheme built from either the original square root ORAM [29] or tree-based ORAMs [67]. Both above schemes hide key-equality of operations. Response-hiding constructions in [11, 27] introduce ORAM-based schemes to achieve backward security. In addition, the use of compression with ORAM-based schemes is considered by Demertzis *et al.* [20] with concrete efficient gains for natural document corpora. We note that many of the above searchable encryption schemes are also implicitly encrypted multi-maps constructions.

The majority of data structure lower bounds are proven in the cell probe model [69]. The chronogram technique was first introduced by Fredman and Saks [24] to prove $\Omega(\lg n/\lg\lg n)$ bounds. Pătraşcu and Demaine [64] present the information transfer technique proving $\Omega(\lg n)$ bounds. Larsen [47] presented the first techniques that proved $\tilde{\Omega}(\lg^2 n)$ bound for dynamic, two-dimensional range counting. This is the current, highest lower bound proven for any data structure with $\Omega(\lg n)$ bit outputs for queries. For dynamic data structures with boolean outputs, the highest lower bound is presented by Larsen *et al.* [51] of $\tilde{\Omega}(\lg^{1.5} n)$. The seminal work by Larsen and Nielsen [49] is the first to show the applicability of the cell probe model for privacy-preserving data structures by giving an $\Omega(\lg n)$ lower bound for ORAMs. Persiano and Yeo [61] extend the $\Omega(\lg n)$ lower bound for differentially private RAMs with weaker privacy. Hubáček *et al.* [35] extend the ORAM lower bounds to the case where the adversary is unaware when operations start and end. Larsen *et al.* [48] present $\tilde{\Omega}(\lg^2 n)$ lower bounds for oblivious near-neighbor search. Lower bounds for ORAMs in the multi-server setting are presented in [50].

2 Definitions and Models

In this section, we define the notation that we will be using throughout the paper. We formalize *the notion of a leakage function* and the *leakage cell probe model*, which is a generalization of the oblivious cell probe model of Larsen and Nielsen [49]. The leakage cell probe model can be used to derive lower bounds on the efficiency of general data structures with respect to a leakage function. We will then describe the *dynamic encrypted multi-map* problem for which we will derive lower bounds. We also consider the dynamic searchable encryption problem whose formal definition can be found in Section 6.1. We start by reviewing the cell probe model.

2.1 Cell Probe Model

The cell probe model was introduced by Yao [69] and has widely been used to prove lower bounds for data structures (see [24, 64, 47, 51] as examples). The goal of the cell probe model is to abstract the interactions of CPUs and word-RAM architectures. Memory in the cell probe model is an array of *cells* where each cell consists of exactly w bits. The operations of a data structure consist of *cell probes* where each probe may read the contents of a cell and/or update the cell's content. The *cost* or *running time* of an operation is measured by the number of cell probes. A data structure in the cell probe model may perform unlimited computation based on the contents of cells that were probed. Note, lower bounds in the cell probe model immediately imply results to more realistic models that measure costs using both memory accesses and computation.

In the context of privacy-preserving data structure, the cell probe model is adapted to a two-party setting: the *client* and the *server*. The client outsources the storage of data to the server and uses the data structure algorithms to perform operations that read and/or update the data stored on the server. For privacy, the client wishes to hide the content of outsourced data and/or the operations performed from the adversarial server. The adversarial server's view consists of the content of all cells on the server and the probes performed by operations. The adversary does not view the content of the client's storage nor the probes performed to the client's storage. In the first work relating the cell probe model to privacy-preserving data structures, Larsen and Nielsen [49] introduced the *oblivious cell probe model* in which any two sequences of operations of the same length are required to induce indistinguishable server's views. This model has been used to prove a lower bound for oblivious RAMs [49] and for other data structures, like stacks and queue [37]. Subsequently, Persiano and Yeo [61] introduced the *differentially private cell probe model*, a generalization of the oblivious cell probe model in which the adversary's view must abide to the standard differential privacy definition for neighboring sequences.

In this work, we define the *leakage cell probe model* which considers data structures with more complex leakage. For a *leakage function* $\mathcal{L}$, we denote the *$\mathcal{L}$-leakage cell probe model* such that the adversary's view when processing two sequences of operations O and O' must be indistinguishable if $\mathcal{L}(O)$ and $\mathcal{L}(O')$ are equal. The leakage cell probe model is a generalization of the oblivious cell probe model as obliviousness can be viewed as privacy with respect to a leakage function that only leaks the number of operations performed. We note that the client-server interaction in the leakage cell probe model is identical to both the oblivious cell probe model [49] and the differentially private cell probe model [61]. The only difference is in the privacy notion.

We next describe the notion of a *data structure problem* in the cell probe model as consisting of a set $\mathbb{U}$ of update operations and a set $\mathbb{Q}$ of queries that return values in the domain $\mathbb{O}$. The response to a query $q \in \mathbb{Q}$ is determined by a function $\mathbb{R} : \mathbb{U}^\star \times \mathbb{Q} \to \mathbb{O}$ based on the choice of the query $q \in \mathbb{Q}$ and the sequence of updates $(u_1, \dots, u_l) \in \mathbb{U}^\star$ that have been executed before the query q. For any **DS** solving a data structure problem in the cell probe model, the server's memory is assumed to consist of w-bit cells. The client's storage consists of c bits. There exists a random string $\mathcal{R}$ accessible by the operations of **DS**. We will assume that $\mathcal{R}$ is finite, but may be arbitrarily large. For cryptographic purposes, $\mathcal{R}$ may act as a private random function or a random oracle. An operation of **DS** is allowed to perform probes to cells in server memory, access bits in the client storage and access bits in $\mathcal{R}$. The data structure is only charged for probes to server cells. Accessing bits in client storage or $\mathcal{R}$ are free. The sequence of cell probes chosen by an operation of **DS** are a deterministic function of the client storage, random string $\mathcal{R}$ and the contents of cells that were previously probed in the current operation. Note, this deterministic function need not be efficiently computable as the cell probe model does not charge for computation. We denote the *failure probability* as the maximum probability that **DS** outputs the incorrect answer over all query operations and preceding sequence of operations. Note that the probability is strictly over the random choice of $\mathcal{R}$. Additionally, we note that the cell probe model assumes that **DS** processes operations in an *online* manner. **DS** must finish processing an operation before receiving the next operation. As a result, each cell probe performed by **DS** may be uniquely associated to an operation. The assumption of online operations is realistic as the majority of practical scenarios consider online operations.

The assumption that $\mathcal{R}$ is finite does not preclude the applicability of our result to algorithms with vanishing failure probabilities that may run infinitely. We show they can be converted into data structures with finite running time but non-zero failure probabilities by a standard reduction. The data structure is run for an arbitrary number of cell probes until the failure probability is sufficiently small. At this point, the data structure must return an answer. Our lower bounds will consider data structures with any constant failure probability strictly less than 1/2. As a result, our lower bounds also apply to data structures whose failure probabilities decrease as the running time increases but have no termination guarantees.

2.2 Leakage Cell Probe Model

In this section, we formalize the privacy notion for data structures in the *leakage cell probe model.* Roughly speaking, we give an upper bound on the maximum amount of information viewed by the adversary when processing a sequence of operations by specifying a *leakage function* $\mathcal{L}$. Concretely, a leakage function $\mathcal{L}$ takes as input any valid sequence of operations, O, of **DS**. For online **DS** and for any sequence $O = (\mathtt{op}_1, \ldots, \mathtt{op}_\ell)$, we can rewrite the leakage $\mathcal{L}(O)$ as:

$$\mathcal{L}(O) = \mathcal{L}(\mathtt{op}_1), \mathcal{L}(\mathtt{op}_1, \mathtt{op}_2), \ldots, \mathcal{L}(\mathtt{op}_1, \ldots, \mathtt{op}_\ell) = \mathcal{L}(O_1), \mathcal{L}(O_2), \ldots, \mathcal{L}(O_\ell),$$

where O_i denotes the prefix $O_i = (\mathtt{op}_1, \ldots, \mathtt{op}_i)$ consisting of all operations up to and including the i-th operation. We formalize the notion that **DS** leaks at most $\mathcal{L}$ by means of an *indistinguishability-based* definition in which we require that, for any two sequences O and O' such that $\mathcal{L}(O) = \mathcal{L}(O')$, no efficient adversary $\mathcal{A}$ can distinguish a sequence of cell probes executed by **DS** while performing O from one executed while performing sequence O'. For two sequences O and O', we say that $\mathcal{L}(O) = \mathcal{L}(O')$ if and only if $\mathcal{L}(O_i) = \mathcal{L}(O'_i)$ for every $i = 1, \ldots, \ell$.

Let us now proceed more formally. For any sequence of operations $O = (\mathtt{op}_1, \ldots, \mathtt{op}_\ell)$, the *adversary's view* $\mathcal{V}_{\mathbf{DS}}(O)$ of **DS** processing O consists of the sequence of probes performed by **DS** while processing sequence O. The randomness of $\mathcal{V}_{\mathbf{DS}}(O)$ is over the choice of the random string $\mathcal{R}$. For online **DS**, each cell probe is uniquely assigned to an operation. So, we can rewrite $\mathcal{V}_{\mathbf{DS}}(O) = (\mathcal{V}_{\mathbf{DS}}(O_1), \ldots, \mathcal{V}_{\mathbf{DS}}(O_\ell))$.

The formal definition of *non-adaptively $\mathcal{L}$-IND* is given below.

Definition 1 (Non-adaptively $\mathcal{L}$-IND)**.** **DS** *is* ν-non-adaptively $\mathcal{L}$-IND *if for every pair of sequences O and O' such that $\mathcal{L}(O) = \mathcal{L}(O')$ and any deterministic polynomial time algorithm $\mathcal{A}$, then*

$$|\Pr[\mathcal{A}(\mathcal{V}_{\mathbf{DS}}(O)) = 1] - \Pr[\mathcal{A}(\mathcal{V}_{\mathbf{DS}}(O')) = 1]| \leq \nu$$

The acute reader might notice several differences between the above security notion and previous definitions (for example, see [17, 16, 43, 13]). First, our definition uses the weaker indistinguishability notion as opposed to the stronger simulation paradigm. Secondly, many previous works consider *adaptive* security where the adversary is allowed to view the leakage by **DS** on previous operations before picking the next operation. Our definition does not allow the operations to be picked depending on the adversary's view. For the third point, our definition considers deterministic, polynomial time adversaries unlike the randomized PPT adversaries of previous security notions. All three differences result in a weaker security notion. However, a lower bound for a scheme satisfying this weaker security notion also implies a lower bound for the normal, stronger security notion. In other words, by assuming a weaker security notion, we improve the strength and applicability of our lower bound.

Finally, we formally define a *$\mathcal{L}$-leakage cell probe model data structure.*

Definition 2. *A* **DS** *is a* $\mathcal{L}$-leakage cell probe model data structure *if* **DS** *has failure probability strictly less than 1/2 and is 1/4-non-adaptively $\mathcal{L}$-IND.*

Note that, the distinguishing probability only has to be at most 1/4 as opposed to $\mathsf{negl}(\lambda)$ where λ is the security parameter. Once again, we stress that this results in a weaker security notion and a lower bound for any **DS** that is 1/4-non-adaptively $\mathcal{L}$-IND applies for any **DS** satisfying a stronger security notion. Overall, our lower bounds for *$\mathcal{L}$-leakage cell probe model data structure* imply lower bounds to the standard simulation-based, adaptive security notions against PPT adversaries with $\mathsf{negl}(\lambda)$ advantage.

In practice, the assumption of failure probability close to 1/2 is unacceptably large. Once again, this is to improve the strength of our lower bound as it immediately implies results for **DS** with small or zero failure probability.

We also note that leakage cell probe model is a generalization of the oblivious cell probe model [49]. Consider the leakage function, $\mathcal{L}(\mathtt{op}_1, \ldots, \mathtt{op}_i) = i$, that only leaks the number of operations. In the $\mathcal{L}$-leakage cell probe model, all sequences of the same length must be indistinguishable which is identical to the oblivious cell probe model [49].

Comparing leakage functions. In general, leakage functions are not numerical as they encapsulate all the information learned by the adversary and for this reason it is hard to linearly order leakage functions. We can nonetheless define the following partial order on leakage functions.

Definition 3. *Leakage function $\mathcal{L}_1$ is* at least as secure as *leakage function $\mathcal{L}_2$ (in symbols $\mathcal{L}_1 \leq \mathcal{L}_2$) if any* **DS** *that is $\mathcal{L}_1$-IND is also $\mathcal{L}_2$-IND.*

We note the we use $\mathcal{L}_1 \leq \mathcal{L}_2$ as the leakage of $\mathcal{L}_1$ is smaller than the leakage of $\mathcal{L}_2$ and that a lower bound for a **DS** with leakage $\mathcal{L}_2$, also applies to any $\mathbf{DS}'$ with leakage $\mathcal{L}_1 \leq \mathcal{L}_2$. The following lemma gives a sufficient condition for $\mathcal{L}_1 \leq \mathcal{L}_2$.

Lemma 1. *If there exists an efficient function F such that for all sequences O of operations it holds that $\mathcal{L}_1(O) = F(\mathcal{L}_2(O))$, then $\mathcal{L}_1 \leq \mathcal{L}_2$.*

2.3 Encrypted Multi-Maps

In this section, we present the *dynamic multi-map problem* where we consider the *multi-map* data structure that maintains m pairs $\mathsf{MM} = \{(\mathtt{key}_i, \mathtt{vals}_i)\}_{i\in[m]}$ where each $\mathtt{key}_i$ is from the *key universe* $\mathcal{K}$ and $\mathtt{vals}_i$ is a tuple of values from the *value universe* $\mathcal{V}$. We assume that all keys are unique (that is, $\mathtt{key}_i \neq \mathtt{key}_j$ for all $i \neq j$). This assumption is without loss of generality as any multi-map with duplicate keys can merge the associated tuples of values. For any $\mathtt{key}_i$, we denote by the number of values associated with $\mathtt{key}_i$ by $\ell(\mathtt{key}_i)$ (that is, $\ell(\mathtt{key}_i) := |\mathtt{vals}_i|$). Note, different keys can be associated with tuples of different length. We denote the total number of values by $n := \sum_{i\in[m]} \ell(\mathtt{key}_i) = \sum_{i\in[m]} |\mathtt{vals}_i|$. Additionally, we introduce the following notation for convenience. For any $\mathtt{key}$, $\mathtt{vals}(\mathsf{MM}, \mathtt{key})$ is the tuple of values associated with $\mathtt{key}$. Whenever the multi-map MM is clear from the context, we will omit MM and write $\mathtt{vals}(\mathtt{key})$ instead of $\mathtt{vals}(\mathsf{MM}, \mathtt{key})$.

We consider dynamic multi-maps with Create, Get and Add operations.

1. Create returns an empty $\mathsf{MM} := \emptyset$.
2. $\mathsf{Get}(\mathtt{key})$ takes as input $\mathtt{key} \in \mathcal{K}$ and outputs $\mathtt{vals}(\mathtt{key})$, the tuple of values associated with $\mathtt{key}$.
3. $\mathsf{Add}(\mathtt{key}, \mathtt{val})$ adds value $\mathtt{val}$ to the tuple associated with $\mathtt{key}$.

Note that we only allow a very simple type of insertions in which only one value is added for each operation. By proving a lower bound on a multi-map with only a simple insertion operation, our lower bound will also apply to more general multi-maps with more complex insertions and update operations.

Definition 4. *The* dynamic encrypted multi-map *problem is parameterized by $\mathcal{K}$, the* key universe*, and by $\mathcal{V}$, the* value universe*. The problem is defined by the tuple $(\mathbb{U}, \mathbb{Q}, \mathbb{R})$ where*

- $\mathbb{U} = \{\mathsf{Add}(\mathtt{key}, \mathtt{val}) \mid \mathtt{key} \in \mathcal{K}, \mathtt{val} \in \mathcal{V}\} \cup \{\mathsf{Create}\}$;
- $\mathbb{Q} = \{\mathsf{Get}(\mathtt{key}) \mid \mathtt{key} \in \mathcal{K}\}$;

and for any sequence $O = (\mathsf{Create}, \mathsf{Add}(\mathtt{key}_1, \mathtt{val}_1), \ldots, \mathsf{Add}(\mathtt{key}_m, \mathtt{val}_m))$,

$$\mathbb{R}(O, \mathsf{Get}(\mathtt{key})) = \{\mathtt{val} \mid \exists\, 1 \leq i \leq m \text{ s.t. } \mathtt{key}_i = \mathtt{key} \text{ and } \mathtt{val}_i = \mathtt{val}\}.$$

In other words, $\mathsf{Get}(\mathtt{key})$ *returns* $\mathtt{vals}(\mathsf{MM}, \mathtt{key})$*, where* MM *is the instance obtained by executing the sequence O of update operations.*

Efficiency measure. For a data structure **DS** solving the dynamic encrypted multi-map problem, we denote $\mathsf{Cost}_{\mathbf{DS}}(O)$ as the expected number of cell probes needed by **DS** to perform the sequence of operations O where the expectation is taken over the random coin tosses of **DS**. We note that, unlike ORAMs, $\mathsf{Cost}_{\mathbf{DS}}$ is not a good measure of the efficiency of the data structure **DS**. For example, some Get operations might return an extremely long tuple while others only a few values and it would be unreasonable to expect these vastly different operations to perform the same number of cell probes. We thus define the *amortized efficiency* $\mathsf{Eff}_{\mathbf{DS}}$ of a data structure $\mathbf{DS}$ solving the dynamic encrypted multi-map problem with respect to a sequence of operations $O = (\mathtt{op}_1, \ldots, \mathtt{op}_\ell)$ as the expected value of the total number of cell probes executed by $\mathbf{DS}$ divided by the total number of values returned by Get or taken as inputs as by Add. More precisely, the add $\mathtt{op} = \mathsf{Add}(\mathtt{key}, v)$ operation will receive a single value tuple as input as in our setting only one value can be added to a key. Therefore, $\mathsf{Eff}_{\mathbf{DS}}(\mathtt{op}) := \mathsf{Cost}_{\mathbf{DS}}(\mathtt{op})$. For each get $\mathtt{op} = \mathsf{Get}(\mathtt{key})$, we consider the length of the returned tuple $\mathtt{vals}(\mathtt{key})$ as the length of the output and thus $\mathsf{Eff}_{\mathbf{DS}}(\mathtt{op}) := \mathsf{Cost}_{\mathbf{DS}}(\mathtt{op})/|\mathtt{vals}(\mathtt{key})|$.

In this paper, we prove lower bounds on $\mathsf{Eff}_{\mathbf{DS}}(n)$ for all probabilistic $\mathbf{DS}$ where $\mathsf{Eff}_{\mathbf{DS}}(n)$ is defined to be the maximum over all possible sequences O of n operations of the total expected amortized efficiency of all n operations where the expectation is taken over the random coin tosses of $\mathbf{DS}$.

3 Leakage Profiles

In this section, we formally define the leakage profile $\mathcal{L}_{\mathsf{dec}}$ for which we prove our main result. As stated before, the efficiency of encrypted multi-maps crucially depends on its leakage. For strong privacy, there exist several solutions of encrypted multi-maps with minimal leakage using efficient oblivious RAMs [1, 57] while achieving logarithmic efficiency. Minimal leakage $\mathcal{L}_{\mathsf{min}}$ refers to the adversary learning only the size of inputs and outputs of operations and nothing else. We formally define $\mathcal{L}_{\mathsf{min}}$ and present a simplified version of a folklore construction in Appendix A.

To understand the transition from constant to logarithmic efficiency as a function of the leakage allowed, we consider the smallest leakage achieved by constant efficiency encrypted multi-maps. In particular, these schemes leak the *global key-equality pattern*, $\mathcal{L}_{\mathsf{glob}}$, where adversaries learn whether two operations use the same key as input or not. We formally define $\mathcal{L}_{\mathsf{glob}}$ and present a straightforward *hash-and-encrypt* compiler that achieves $\mathcal{L}_{\mathsf{glob}}$ leakage in Appendix B.

The next step up in security would be to still allow the adversary to learn which operations are on the same key but to limit this ability to operations of the same type. That is, the adversary still learns whether two Get operations are on the same key or not and whether two Add operations are on the same key but it cannot link an Add and a Get that receive the same key as input. This is captured by the following leakage function.

Definition 5 (Decoupled Key-Equality Leakage $\mathcal{L}_{\mathsf{dec}}$)**.** *For sequence $O = (\mathtt{op}_0 = \mathsf{Create}, \mathtt{op}_1, \ldots, \mathtt{op}_\ell)$ of operations where $\mathtt{key}_1, \ldots, \mathtt{key}_\ell$ are the input keys to each non-create operation, then the* decoupled key-equality leakage *$\mathcal{L}_{\mathsf{dec}}(O)$ associated with O consists of $\mathcal{L}_{\mathsf{dec}}(O) = (\mathcal{L}_{\mathsf{dec}}(O_0), \ldots, \mathcal{L}_{\mathsf{dec}}(O_\ell))$ where $O_i = (\mathtt{op}_0, \ldots, \mathtt{op}_i)$ and MM^{O_i} is the multi-map resulting from the first i operations. Then, $\mathcal{L}_{\mathsf{dec}}(O_i)$ is defined as:*

1. *if $\mathtt{op}_i = \mathsf{Create}$ then $\mathcal{L}_{\mathsf{dec}}(O_i) = (\mathsf{Create})$;*
2. *if $\mathtt{op}_i = \mathsf{Add}(\mathtt{key}_i, \mathtt{val}_i)$ then $\mathcal{L}_{\mathsf{dec}}(O_i) = (\mathsf{Add}, \mathsf{ep}^{\mathsf{dec}}{}_i)$;*
3. *if $\mathtt{op}_i = \mathsf{Get}(\mathtt{key}_i)$ then $\mathcal{L}_{\mathsf{dec}}(O_i) = (\mathsf{Get}, |\mathtt{vals}(\mathsf{MM}^{O_{i-1}}, \mathtt{key}_i)|, \mathsf{ep}^{\mathsf{dec}}{}_i)$.*

The decoupled key-equality pattern $\mathsf{ep}^{\mathsf{dec}}{}_i := (\mathsf{ep}^{\mathsf{dec}}{}_{i,1}, \ldots, \mathsf{ep}^{\mathsf{dec}}{}_{i,i-1})$ *is:*

$$\mathsf{ep}^{\mathsf{dec}}{}_{i,j} = \begin{cases} \perp, & \textit{if } \mathtt{op}_i \textit{ and } \mathtt{op}_j \textit{ are not of the same type.} \\ 0, & \textit{if } \mathtt{op}_i \textit{ and } \mathtt{op}_j \textit{ are of the same type and } \mathtt{key}_i \neq \mathtt{key}_j. \\ 1, & \textit{if } \mathtt{op}_i \textit{ and } \mathtt{op}_j \textit{ are of the same type and } \mathtt{key}_i = \mathtt{key}_j. \end{cases}$$

We note that the above leakage still leaks the number of returned values for each Get operation. Using Add key-equality leakage, the adversary can observe the number of values added for a pseudonymous representation of a key. If the number of values added is unique for any key, then the adversary will learn the global key-equality pattern about this specific key that leaks whether specific Add and Get operations operate on this key with a unique number of associated values. In particular, $\mathcal{L}_{\mathsf{dec}}$ hides key-equality patterns between Add and Get operations when there exist multiple keys with the same number of associated values when Get is executed. In other words, $\mathcal{L}_{\mathsf{dec}}$ is a very minimal increase in privacy over $\mathcal{L}_{\mathsf{glob}}$. The main result of this paper is that $\mathcal{L}_{\mathsf{dec}}$-IND security for encrypted multi-maps (and arrays) incurs $\Omega(\lg n)$ overhead even though it is minimally more secure than $\mathcal{L}_{\mathsf{glob}}$-IND schemes.

We can further extend our lower bounds to **DS** with even larger leakage functions. We define leakage functions $\mathcal{L}_{\mathsf{add}}$ and $\mathcal{L}_{\mathsf{get}}$, which leak the decoupled key-equality pattern like $\mathcal{L}_{\mathsf{dec}}$. Additionally, $\mathcal{L}_{\mathsf{add}}$ leaks the keys and values that are input to all Add operations while $\mathcal{L}_{\mathsf{get}}$ leaks the keys that are input to all Get operations. In other words, $\mathcal{L}_{\mathsf{add}}$ enables the multi-map to perform Add operations in the plaintext while $\mathcal{L}_{\mathsf{get}}$ enables the multi-map to perform Get operations in the plaintext. It turns out our lower bounds still apply as long as the encrypted multi-map performs at most one of either Get or Add operations are performed in the plaintext. We formally define $\mathcal{L}_{\mathsf{add}}$ and $\mathcal{L}_{\mathsf{get}}$ in Section 5. The counterparts of $\mathcal{L}_{\mathsf{add}}$ and $\mathcal{L}_{\mathsf{glob}}$ for dynamic searchable encryption can be found in Section 6.

4 Lower Bounds for Decoupled Key-Equality Leakage

In this section, we present our main result that any encrypted multi-map with leakage at most $\mathcal{L}_{\mathsf{dec}}$ must incur logarithmic overhead.

Theorem 3. *Let* **DS** *be a $\mathcal{L}_{\mathsf{dec}}$-leakage cell probe model dynamic encrypted multi-map implemented over w-bit cells and a client with c bits of storage. Then*

$$\mathsf{Eff}_{\mathbf{DS}}(n) = \Omega\left(\lg\left(\frac{n}{c}\right) \cdot \frac{\lg(n)}{w}\right).$$

In the natural setting that $c = O(n^{\alpha})$, for some constant $0 \leq \alpha < 1$, and cell sizes of $w = \Theta(\lg n)$ bits, the above bound simplifies to $\Omega(\lg n)$.

This result will be proven using the information transfer technique [64]. Throughout the proof, we will assume that **DS** has error probability at most $1/128$ (instead of strictly smaller than $1/2$) and this is without loss of generality as we can apply a standard reduction of executing a constant number of independent copies and returning the majority answer without affecting the asymptotic efficiency.

4.1 Hard Distribution

We start by formalizing the hard distribution and random variables used in our proof. Fix positive integers n and l such that $l < n$. Fix a small constant $0 < \epsilon < 1$ and set $p := n^{1-\epsilon}$. The hard distribution will use the following $p + 1$ disjoint sets of values:

1. V_0 consisting of l values;
2. $V_1, \ldots, V_p$ each consisting of n^{ϵ} values;

Additionally, we define the following $2p$ pairwise disjoint sets of keys:

1. Sets K_j^{a}, for $j = 1, \ldots, p$, each of size n^{ϵ};
2. Sets K_j^{g}, for $j = 1, \ldots, p$, each of size n^{ϵ}.

The hard distribution is generated by a process consisting of $p + 1$ phases where each phase is uniquely indexed by $j \in \{0, \ldots, p\}$. After phase 0 has been completed, each of the n keys in the sets $K_1^{\mathsf{g}}, \ldots, K_p^{\mathsf{g}}$ will be associated with the l values from the set V_0. In each phase $j \in \{1, \ldots, p\}$, a Get operation for each key in the set K_j^{g} is executed and l Add operations for each key in the set K_j^{a}. The values associated with each of the inserted keys in K_j^{a} will receive a random and independently chosen subset containing exactly l values chosen from the set of values V_j. We formally define this probabilistic process which generates sequences of encrypted multi-map operations in Table 1. The resulting distribution of operations will be our

```
$\mathsf{Hard}_{n,l,\epsilon}(V_0, V_1, \ldots, V_p, K^{\mathsf{a}}_1, \ldots, K^{\mathsf{a}}_p, K^{\mathsf{g}}_1, \ldots, K^{\mathsf{g}}_p)$

• Phase 0:
  Execute SubPhase $\mathsf{Init_i}$ for each $i \in \{1, \ldots, p\}$:
      For each key $\in K^{\mathsf{g}}_i$:
          For each val $\in V_0$:
              output: Add(key, val).
• Phase j for each $j \in \{1, \ldots, p\}$:
  Execute SubPhase $\mathsf{A_j}$ of add operations and SubPhase $\mathsf{G_j}$ of get operations.
    1. SubPhase $\mathsf{A_j}$
       For each key $\in K^{\mathsf{a}}_j$:
           Select subset $V_{\mathtt{key}} \subset V_j$ of $k$ values uniformly at random.
           For each val $\in V_{\mathtt{key}}$:
               output: Add(key, val).
    2. SubPhase $\mathsf{G_j}$
       For each key $\in K^{\mathsf{g}}_j$:
           output: Get(key).
```

Table 1: Generation of hard distribution.

hard distribution, denoted by $\mathsf{Hard}(V_0, V_1, \ldots, V_p, K^{\mathsf{a}}_1, \ldots, K^{\mathsf{a}}_p, K^{\mathsf{g}}_1, \ldots, K^{\mathsf{g}}_p)$, used in our lower bound. For convenience, we will assume that all of n, l, ϵ as well as the sets $V_0, V_1, \ldots, V_p, K^{\mathsf{a}}_1, \ldots, K^{\mathsf{a}}_p, K^{\mathsf{g}}_1, \ldots, K^{\mathsf{g}}_p$ are fixed going forward. As a result, we will denote our hard distribution simply by Hard.

As described in Table 1, a sequence in the support of our hard distribution consists of $p+1$ phases. Phase 0 consists of adding the tuple of values V_0 for each key in K^{g}_i for all $i \in \{1, \ldots, p\}$. For all $j \in \{1, \ldots, p\}$, phase j consists of two sub-phases: sub-phase A_j that consists of n^ϵ Add operations that is directly followed by sub-phase G_j that consists of n^ϵ Get operations. The Add operations of phase j consist of associating each key in K^{a}_j with a subset of l values chosen uniformly at random from the set V_j. We show that this is naturally defined by a bipartite graph $B_j = (K^{\mathsf{a}}_j, V_j, E_j)$ where the set of key K^{a}_j appear in the left partition, the set of values V_j appear in the right partition, and E_j represents the edge set. An edge $(\mathtt{val}, \mathtt{key})$ appears in E_j if and only if val is added to the tuple of values associated with key, that is $\mathtt{val} \in \mathtt{vals}(\mathtt{key})$. We note that our choice of adding l randomly chosen values to each $\mathtt{key} \in K^{\mathsf{a}}_j$ is equivalent to choosing B_j uniformly at random from the set of all left l-regular bipartite graphs. Furthermore, this bipartite graph B_j uniquely identifies the sequence of Add operations that will appear in phase j.

Leakage of the hard sequence. Let us consider any fixed sequence H in the support of our hard distribution. We analyze the leakage available to the adversary when executing H on an encrypted multi-map with leakage at most $\mathcal{L}_{\mathsf{dec}}$.

We observe that each of the Get operations returns the l values in V_0 as all K^{g}_i are pairwise disjoint by definition. Furthermore, all Get operations occur for unique keys. Thus, the query leakage of $\mathcal{L}_{\mathsf{add}}(H)$ for each Get operation in H will be identical where the adversary learns the size of the tuple associated with each query key is always l and that the queried key is unique from all other queried keys. In other words, the Get key-equality leakage pattern will be empty.

For the leakage incurred by Add operations, we observe that the $2p$ sets $\{K^{\mathsf{g}}_i\}_{i \in \{1,\ldots,p\}}$ and $\{K^{\mathsf{a}}_i\}_{i \in \{1,\ldots,p\}}$ are pairwise disjoint by definition. H will perform exactly n^ϵ consecutive Add operations to each key in $\{K^{\mathsf{g}}_i\}_{i \in \{1,\ldots,p\}}$ during phase 0. In phase j, H will perform exactly n^ϵ consecutive Add operations to each key in K^{a}_j. Therefore, the Add key-equality leakage pattern will reveal to the adversary that Add operations to the same key always occurs in consecutive blocks of n^ϵ operations.

From the above, it is not hard to see that $\mathcal{L}_{\mathsf{dec}}$ on any pair of sequences H_1 and H_2 in the support of the hard distribution are the same. The leakage during Get operations depends only on the choice of l. Similarly, the leakage during Add operations depends only on the choice of n^ϵ. As both l and n^ϵ are fixed, the leakage $\mathcal{L}_{\mathsf{dec}}(H_1)$ and $\mathcal{L}_{\mathsf{dec}}(H_2)$ for any H_1 and H_2 in the support of the hard sequence will always be identical.

Information transfer tree. Next, we define an abstract model of data flow called the information transfer tree, which will be integral in our lower bound proofs. For each sequence H in the support of the hard distribution, we will denote the information transfer tree of H by $\mathcal{T}(H)$. $\mathcal{T}(H)$ is a binary tree whose nodes contain the cell probes performed by **DS** when executing H. Without loss of generality, we assume that p is a power of 2 and construct a complete binary tree with p leaves. For all $j \in \{1, \ldots, p\}$, we assign phase j, consisting of subphases A_j and G_j, to the j-th leftmost leaf. Phase 0 is ignored in the construction of the information transfer tree.

Next, we proceed by uniquely assigning cell probes to nodes of the information transfer tree. Consider a probe to cell address x that occurs as part of an operation of the phase j. If this is the first probe to cell address x, then the probe is not assigned to any node. Otherwise, pick the most recent phase i that precedes phase j $(i \leq j)$ such that an operation in phase i overwrote the contents at cell address x. The probe is then assigned to the least common ancestor of the leaf nodes associated with phase j and phase i. Note that the assignment of probes to nodes is probabilistic and depends on the random coin tosses $\mathcal{R}$ of **DS**. So, $\mathcal{T}(H)$ is also a random variable over $\mathcal{R}$. For each node v, we define $\mathcal{C}_v(H)$ as the the set of probes assigned to v when executing H over the choice of $\mathcal{R}$. We denote $\mathcal{T}(\mathsf{Hard})$ and $\mathcal{C}_v(\mathsf{Hard})$ as probability distributions over the random choices of both Hard and $\mathcal{R}$.

4.2 Bounding Probes Assigned to Internal Nodes

To prove our lower bound, we will show that for many nodes v, the expected size of $\mathcal{C}_v(\mathsf{Hard})$ must be large. Since each probe is assigned to at most one node, the sum of the number of probes assigned over all the nodes v will result in a lower bound on the expected number of cell probes needed to process a random sequence generated by Hard.

Denote depth(v) as the distance of v from the root. As there are $p = n^{1-\epsilon}$ leaf nodes, the leaf nodes have depth $\lg(p) = (1-\epsilon)\lg(n)$ where all logarithms are base 2. We will prove the following lemma which states that a large number of cells must be assigned to nodes in expectations for all nodes that are not too close to either the root node or the leaf nodes.

Lemma 2. *Let* **DS** *be a* $\mathcal{L}_{\mathsf{dec}}$*-leakage cell probe model dynamic encrypted multi-map scheme that errs with probability at most* $1/128$. *For any* $1 \leq l \leq n^{\epsilon/2}$, *there exists a constant* $\gamma_1 > 0$ *such that for every node* v *of depth* $8 \leq d \leq \frac{1-\epsilon}{2}\lg(\frac{n}{c})$, *it must be that*

$$\mathsf{E}\left[|\mathcal{C}_v(\mathsf{Hard})|\right] \geq \gamma_1 \cdot \frac{n}{2^d} \cdot \frac{l \lg n}{w}.$$

We now show that Lemma 2 would complete the proof of Theorem 3.

Proof of Theorem 3. Recall that each probe is assigned to a most one node of the tree. So, counting the cell probes assigned to a subset of the trees gives a lower bound on the number of cell probes. A complete binary tree has 2^d nodes at depth d. By Lemma 2, all nodes v such that $8 \leq \text{depth}(v) \leq \frac{1-\epsilon}{2}\lg(\frac{n}{c})$ have $\Omega(\frac{n}{2^d}\frac{l\lg n}{w})$ assigned cell probes in expectation. Therefore, each level in this range contributes $\Omega(n \cdot \frac{l \lg n}{w})$ cell probes in expectation and by multiplying by the number of levels for which Lemma 2 holds we obtain $\Omega(n\lg(\frac{n}{c})\frac{l\lg n}{w})$ cell probes. Recall that we are considering both the Get and Add operations and the efficiency is measured as running time per response of a query and per value added. Note, a hard sequence performs $\Theta(n)$ queries with exactly l responses each and performs $\Theta(n \cdot l)$ Add each of exactly one value. So, we get the expected amortized running time is $\Omega(\lg(\frac{n}{c}) \cdot \frac{\lg n}{w})$. □

4.3 Using the Privacy Guarantees

Therefore, it remains to prove Lemma 2 to finish the proof of our main result. To do this, we will prove a weaker lemma which shows that for a large number of nodes v there exists a probability distribution

Hard_v (specifically built for node v) that forces the number of probes assigned to v, $\mathcal{C}_v(\mathsf{Hard}_v)$, to be large in expectation. This lemma is significantly weaker than Lemma 2 which states that there exists a *single* distribution, Hard, that simultaneously assigns many probes to the sets $\mathcal{C}_v(\mathcal{H})$ for a large number of nodes v. We note that our proof must critically use the privacy guarantees of **DS** as there exist constructions with $O(1)$ efficiency that do not provide any privacy such as the dynamic perfect hashing solutions [21]. By leveraging the privacy guarantees of **DS**, we can show the two statements are equivalent. First, we formally state our weaker lemma.

Lemma 3. *Fix integers n and l and $0 \leq \epsilon \leq 1$ such that $1 \leq l \leq n^{\epsilon/2}$. Let* **DS** *be a $\mathcal{L}_{\mathsf{dec}}$-leakage cell probe model dynamic encrypted multi-map scheme that errs with probability at most $1/128$. Then, there exists a constant $\gamma_2 > 0$ such that, for every node v with $8 \leq depth(v) \leq \frac{1-\epsilon}{2} \lg \frac{n}{c}$, there exists a probability distribution Hard_v such that $\mathcal{L}_{\mathsf{dec}}(\mathsf{Hard}) = \mathcal{L}_{\mathsf{dec}}(\mathsf{Hard}_v)$ and*

$$\Pr\left[|\mathcal{C}_v(\mathsf{Hard}_v)| \geq \gamma_2 \cdot \frac{n}{2^d} \cdot \frac{l \lg n}{w}\right] \geq \frac{1}{2}.$$

By combining Lemma 3 with the privacy guarantees of **DS**, we show that we can prove Lemma 2. By Lemma 3, there exists a distribution Hard_v that forces any **DS** with at most $1/128$ failure probability to assign many cell probes to $\mathcal{C}_v(\mathsf{Hard}_v)$ in expectation. Furthermore, Hard_v and Hard have the same leakage with respect to leakage function $\mathcal{L}_{\mathsf{dec}}$. Since the size of $\mathcal{C}_v(O)$ can be computed by a deterministic, polynomial time algorithm for any sequence O, it must be that the expected sizes of $\mathcal{C}_v(\mathsf{Hard})$ and $\mathcal{C}_v(\mathsf{Hard}_v)$ cannot differ significantly. Otherwise, a deterministic, polynomial time adversary will be able to distinguish whether **DS** is executing a sequence randomly drawn from Hard or Hard_v. As a result, it can be shown that the size of $\mathcal{C}_v(\mathsf{Hard})$ for all nodes v must be large in expectation. We proceed to formalize these ideas.

Proof of Lemma 2. Pick $\gamma_1 < \gamma_2/4$ and suppose, for the sake of contradiction, that there exists a node v of depth $8 \leq \text{depth}(v) \leq \frac{1-\epsilon}{2} \lg \frac{n}{c}$, such that $\mathsf{E}[|\mathcal{C}_v(\mathsf{Hard})|] < \frac{\gamma_2}{4} \frac{n}{2^d} \cdot \frac{l \lg n}{w}$. By Markov's inequality, we have that

$$\Pr\left[|\mathcal{C}_v(\mathsf{Hard})| \geq \gamma_2 \cdot \frac{n}{2^d} \cdot \frac{l \lg n}{w}\right] < 1/4.$$

On the other hand, by Lemma 3 we know that

$$\Pr\left[|\mathcal{C}_v(\mathsf{Hard}_v)| \geq \gamma_2 \cdot \frac{n}{2^d} \cdot \frac{l \lg n}{w}\right] \geq 1/2.$$

Therefore a deterministic, polynomial time adversary that computes the number of probes assigned to v and outputs 1 if and only if the number of cell probes assigned to v is less than $\gamma_2 \cdot \frac{n}{2^d} \cdot \frac{l \lg n}{w}$. This adversary successfully distinguishes whether **DS** is processing Hard or Hard_v. Thus, this contradicts that **DS** is non-adaptively $\mathcal{L}_{\mathsf{dec}}$-IND. □

4.4 An Encoding Argument

Finally, we present the proof of Lemma 3 that requires finding a distribution Hard_v with the properties that $\mathcal{C}_v(\mathsf{Hard}_v)$ is large in expectation and that Hard_v has the same leakage as Hard with respect to $\mathcal{L}_{\mathsf{dec}}$. We start by describing simple modifications to Hard that are used to construct Hard_v while keeping $\mathcal{L}_{\mathsf{dec}}$ unchanged.
$\mathcal{L}_{\mathsf{dec}}$-invariant swaps. Let us start with a simple example and consider distribution $\mathsf{Hard}^{(s,s')}$ defined as follows for indices $1 \leq s \leq s' \leq p$. Recall that in our definition of Hard, phase $1 \leq j \leq p$ consists of subphase $\mathsf{A_j}$ where Add operations are performed on the keys in $K^{\mathsf{a}}_{s'}$ and subphase $\mathsf{G_j}$ where Get operations are performed on the keys in $K^{\mathsf{g}}_{s'}$. In distribution $\mathsf{Hard}^{(s,s')}$ where $s \leq s'$, subphase $\mathsf{A}_{s'}$ still consists of Add operations performed on the keys in $K^{\mathsf{a}}_{s'}$ but the Get operations of subphase $\mathsf{G}_{s'}$ are performed on the keys in K^{a}_{s} instead of $K^{\mathsf{g}}_{s'}$. We show that this swap does not change the leakage with respect to $\mathcal{L}_{\mathsf{dec}}$.

Lemma 4. *For any $1 \leq s \leq s' \leq p$, $\mathcal{L}_{\mathsf{dec}}(\mathsf{Hard}) = \mathcal{L}_{\mathsf{dec}}\left(\mathsf{Hard}^{(s,s')}\right)$.*

Proof. Since no Add operation is affected by the swap, the leakage generated by the Add operations remains the same. For the Get operations, observe that the Get operations in $\mathsf{Hard}^{(s,s')}$ are always performed on distinct keys, just as in Hard and thus the key-equality pattern does not change. Moreover, since $s \le s'$, when the keys in K^{a}_s are queried in phase s', l values have already been added to them. Therefore the Get operations of $\mathsf{Hard}^{(s,s')}$ return l values just as in Hard and thus the volume pattern does not change either. ☐

The same argument applies to any set $S := \{(s_1, s'_1), \dots, (s_\ell, s'_\ell)\}$ of swaps provided that $s_i \le s'_i$ for all $i \in \{1, \dots, \ell\}$ and that each index is involved in at most one swap. We call such a set S of swaps, a *legal* set of swaps and we will denote by Hard^S the distribution resulting from performing all swaps in the legal set S of swaps. The following lemma follows by applying swaps one at a time and by invoking Lemma 4 for each application.

Lemma 5. *For any legal set S of swaps where $S := \{s_1, s'_1), \dots, (s_\ell, s'_\ell)\}$ such that $s_i \le s'_i$ for all $i \in \{1, \dots, \ell\}$ and that each s_i and s'_i appears at most once in S, then $\mathcal{L}_{\mathsf{dec}}(\mathsf{Hard}) = \mathcal{L}_{\mathsf{dec}}(\mathsf{Hard}^S)$.*

Defining Hard_v. Distribution Hard_v is designed to make the set of cell probes assigned to v, $\mathcal{C}_v(\mathsf{Hard}_v)$ large in expectation for any **DS** with a bounded failure probability while ensuring the leakages of Hard and Hard_v remain identical according to $\mathcal{L}_{\mathsf{dec}}$. Recall that $\mathcal{C}_v(\mathsf{Hard}_v)$ contains only probes that occur in the right subtree of v to a cell last overwritten in the left subtree of v. Suppose we design Hard_v so that the Add operations in the left subtree of v insert a large amount of random information that is independent from all other operations and that this information must be extracted by Get operations in the right subtree of v. For **DS** to answer the queries with low failure probability, a lot of the information inserted in the left subtree of v must be transferred to the answers of the queries in the right subtree of v. We show that there are only two ways to transfer information between the left and right subtree. First, the client can store information in the c bits of client storage. The other option is that queries in the right subtree of v must probe cells that were last overwritten in the left subtree of v. If the information required to transfer is much larger than the c bits of client storage, it must be the number of probes performed by queries in the right subtree of v to cells that were last overwritten by operations in left subtree of v much be sufficiently large. Note all these probes will be assigned to $\mathcal{C}_v(\mathsf{Hard}_v)$ as desired.

Let us be more precise. Fix any node v of depth d and denote by 2ℓ the number of leaves in the tree rooted at v so that each of the left and right subtree has exactly $\ell := p/2^{d+1}$. Let i be the index of the leftmost leaf of the subtree rooted at v. Then, the Add operations performed in the left subtree of v add values to keys in $K^{\mathsf{a}}_i, \dots, K^{\mathsf{a}}_{i+\ell-1}$ according to the bipartite graphs $B_i, \dots, B_{i+\ell-1}$. Recall that each of the bipartite graphs B_j where $j \in \{i, \dots, i+\ell-1\}$ arrange the keys K^{a}_j in the left subtree and the values V_j in the right subtree. An edge occurs between a key $\mathtt{key} \in K^{\mathsf{a}}_j$ and value $\mathtt{val} \in V_j$ if and only if $\mathtt{val}$ is added to the tuple of values associated with $\mathtt{key}$. In other words, the operation $\mathsf{Add}(\mathtt{key}, \mathtt{val})$ was executed in the left subtree of v. The Get operations performed in the right subtree of v are for keys in $K^{\mathsf{g}}_{i+\ell}, \dots, K^{\mathsf{g}}_{i+2\ell-1}$. Each of these keys has been associated with the l values of V_0 by the Add operations of phase 0. We construct Hard_v by modifying the Get operations in the right subtree of v to query the keys that were used as inputs by the Add operations of the left subtree of v. Specifically, the leaves in the right subtree of v will contain Get operations to the keys in $K^{\mathsf{a}}_i, \dots, K^{\mathsf{a}}_{i+\ell-1}$. This corresponds to the set of swaps $\mathsf{swap}_v = \{(i, i+\ell), \dots, (i+\ell-1, i+2\ell-1)\}$ which is easily seen to be legal. By invoking Lemma 5, we get the following lemma.

Lemma 6. *Leakage distributions $\mathcal{L}_{\mathsf{dec}}(\mathsf{Hard}_v)$ and $\mathcal{L}_{\mathsf{dec}}(\mathsf{Hard})$ are identical.*

We remind the reader that in phase j, each keyword of K^{a}_j is assigned a random subset of exactly l values from the set of values V_j. These chosen values are uniquely defined by a left l-regular bipartite graph B_j that is chosen uniformly at random. The entropy of the left subtree of v in Hard_v originates from the chosen bipartite graphs B_j that are chosen uniformly and independently at random for all $j \in \{i, \dots, i+\ell-1\}$. For each key that appears in the left partition of B_j, there are $\binom{|V_j|}{l} = \binom{n^\epsilon}{l}$ possible choices for the l edges (corresponding to the l values that will be associated with the key). Therefore, the choice of the l edges adjacent to each key in the left partition of B_j has entropy $\lg \binom{n^\epsilon}{l}$. By picking $l \in \{1, \dots, n^{\epsilon/2}\}$, the choice of the edges adjacent to each key in the left partition of B_j generates $\Omega(l \lg n)$ bits of entropy by applying Stirling's approximation. We note our lower bound do not assume any entropy for the actual values as done in previous lower bound results [49, 61].

As the right subtree of v will query for all keys that were input to Add operations in the left subtree and **DS** has low failure probability, most of this entropy must be retrieved by **DS** from the left subtree of v. Note, there are a total of $\Theta(\frac{n}{2^d})$ queries performed in the right subtree of v. As a result, $\Omega(\frac{n}{2^d} \cdot l \lg n)$ bits of entropy must be transferred from the left subtree. Each cell probe can transfer at most w bits of entropy and, intuitively, this implies that $\Omega(\frac{n}{2^d} \cdot \frac{l \lg n}{w})$ cell probes must be assigned to v. We now formalize these arguments using an encoding argument.

Lemma 7. *Fix integers n and l and $0 \le \epsilon \le 1$ such that $1 \le l \le n^{\epsilon/2}$. Let* **DS** *be a $\mathcal{L}_{\mathsf{dec}}$-leakage cell probe model dynamic encrypted multi-map that errs with probability at most $1/128$. For every node v of depth $8 \le d \le \frac{1-\epsilon}{2} \lg \frac{n}{c}$,*

$$\Pr\left[|\mathcal{C}_v(\mathsf{Hard}_v)| \ge \frac{1}{100} \cdot \frac{n}{2^d} \cdot \frac{l \lg n}{w}\right] \ge \frac{1}{2}.$$

Proof. Fix any vertex v with depth $8 \le d \le \frac{1-\epsilon}{2} \lg \frac{n}{c}$. We consider the one-way communication problem between Alice and Bob in which a sequence O of operations is sampled according to Hard_v. The entirety of O is given to Alice whereas Bob receives all of O except the operations performed in the left subtree of O. That is, the operations of phases $i, \ldots, i + \ell - 1$ in O are only given to Alice and not to Bob for some i where $\ell = \frac{n}{2^{d+1}}$. Both Alice and Bob receive common randomness $\mathcal{R}$ used by **DS**. Furthermore, they have agreed on an arbitrary, but fixed ordering for each of the value and key sets. The goal of the one-way communication is for Alice to allow Bob to reconstruct the missing operations which are uniquely defined by the bipartite graphs $B_i, \ldots, B_{i+\ell-1}$. We observe that the entropy of the missing bipartite graphs is $\ell \cdot \lg \binom{n^\epsilon}{l} = \Theta((n/2^d) \cdot \lg \binom{n^\epsilon}{l})$ even when conditioned on Bob's input as all the graphs are chosen independently of $\mathcal{R}$ and all other operations that appear in O. By Shannon's source coding theorem, the expected length of Alice's message must be at least as large as the entropy of the graphs.

Towards a contradiction, we will assume that there exists **DS** with error probability at most $1/128$ such that $\Pr[|\mathcal{C}_v(\mathsf{Hard}_v)| \ge \frac{1}{100} \cdot \frac{1}{w} \cdot \frac{n}{2^{d+1}} \cdot \lg \binom{n^\epsilon}{l}] < \frac{1}{2}$. We will use **DS** to construct an impossible encoding contradicting Shannon's source coding theorem. Note this assumption contradicts the statement of Lemma 7 for any $1 \le l \le n^{\epsilon/2}$ as by Stirling's approximation it implies that $\lg \binom{n^\epsilon}{l} = \Omega(l \lg(n))$.

Alice's Encoding. Alice receives the sequence O sampled according to Hard_v and $\mathcal{R}$ as input and produces the following encoding:

1. Alice executes **DS** using $\mathcal{R}$ as the randomness and performs all operations in sequence O up to, but not including, phase $i + \ell$. Note that phase $i + \ell$ is the first phase in O that belongs to the right subtree of v. At this point, Alice takes a `snapshot` of the contents of all memory cells on the server as well as the contents of client storage.

2. Alice executes the remaining operations in v's right subtree. That is, all operations of phases $i + \ell, \ldots, i + 2\ell - 1$ in O. Alice collects the set F of all query operations in v's right subtree where **DS** fails to return the correct answer. Additionally, Alice collects the set $C_v(O)$ of the cell probes that are assigned to v along with the addresses of the probed cells.

3. If either $|F| \ge \frac{1}{32} \cdot \frac{n}{2^{d+1}}$ or $|C_v(O)| \ge \frac{1}{100} \cdot \frac{n}{2^{d+1}} \cdot \lg \binom{n^\epsilon}{l} / w$, then Alice's encoding will start with a 0 followed by the response to each of the queries in the right subtree of v. Specifically, for $j \in \{i, \ldots, i + \ell - 1\}$, Alice iterates through all $\mathtt{key} \in K_j^{\mathsf{a}}$ in the order agreed upon with Bob and encodes the subset of l values from V_j associated with each $\mathtt{key}$ using $\lg \binom{n^\epsilon}{l}$ bits. This completes Alice's encoding for this case.

4. Suppose instead that $|F| < \frac{1}{32} \cdot \frac{n}{2^{d+1}}$ and $|C_v(O)| < \frac{1}{100} \cdot \frac{n}{2^{d+1}} \cdot \lg \binom{n^\epsilon}{l} / w$. In this case, Alice's encoding will start with a 1-bit and continues by encoding the following information:

(a) The c bits of client storage recorded in snapshot.
(b) The number $|F|$ of failed query using $\Theta(\lg n)$ bits as $|F| \leq n$.
(c) The index and the answer of the $|F|$ keys in $K_i^{\mathsf{a}} \cup \ldots \cup K_{i+\ell-1}^{\mathsf{a}}$ for which **DS** fails to return the correct answer. The indices are encoded using $\lg \binom{n/2^{d+1}}{|F|}$ bits and the answer to each of the failing queries are encoded using $\lg \binom{n^\epsilon}{l}$.
(d) The number $|C_v(O)|$ of the probes assigned to v using $\Theta(\lg n)$ bits.
(e) The address and content of each cell probe in $C_v(O)$ where w bits are used to encode the address and another w bits to encode the contents.

Bob's Decoding. Bob receives Alice's encoding, the sequence of operations O except for the operations of that occur phases $i, \ldots, i+\ell-1$ and the random string $\mathcal{R}$. Bob decodes in the following manner:

1. If Alice's encoding starts with a 0-bit then the answers to all Get queries of Phases $i+\ell, \ldots, i+2\ell-1$ are explicitly encoded in Alice's message and thus Bob proceeds as follows. For $j \in \{i, \ldots, i+\ell-1\}$ and for each `key` $\in K_j^{\mathsf{a}}$ in the agreed upon order, Bob reads the $\lg \binom{n^\epsilon}{l}$ bits that encode which l values of V_j have been assigned to `key`. This directly provides the l edges of the vertex in B_j corresponding to `key`. Repeating this process for all keywords allows Bob to completely retrieve B_j completing the decoding when Alice's message starts with a 0-bit.

2. From now on, we suppose Alice's encoding starts with a 1-bit.

   (a) Bob simulates **DS** using $\mathcal{R}$ for phases $0, \ldots, i-1$. That is, all operations up to, but not including, the first operation of v's left subtree. The result of this execution is identical to Alice's execution as they both use the same random string $\mathcal{R}$. Bob will record the contents of all cells in snapshot$'$.
   (b) Bob skips phases $i, \ldots, i+\ell-1$ consisting of operations in the left subtree of v.
   (c) Bob retrieves the following information from Alice's encoding:
      i. The contents of client storage in snapshot where snapshot is the state of **DS** just before any operations in the right subtree of v.
      ii. The set F of keywords for which **DS** will fail to return the correct answer. For each of these failed keywords, Bob will also retrieve the correct answer from Alice's encoding using the same algorithm as the one where Alice's encoding started with a 0-bit described above.
      iii. The address and content of each of the cells in $\mathcal{C}_v(O)$.
   (d) Bob simulates **DS** on the operations in the right subtree of v. That is, all phases $j \in \{i+\ell, \ldots, i+2\ell-1\}$ using $\mathcal{R}$. Specifically, for each cell probe performed by **DS**, Bob checks if the probed cell was last overwritten by any of the preceding operations in the right subtree of v. If so, Bob will use the most recent contents of the cell. Otherwise, checks if the cell belongs to $C_v(O)$ in which case Bob will use the contents of the cell that were encoded by Alice. Finally if the cell was last overwritten before any operations in the left subtree of v, Bob will use the cell content as reported by snapshot$'$. After Bob completes the simulation, Bob successfully decodes the answer for all queries where **DS** returns the correct answer. As a result, Bob successfully decodes all bipartite graphs $B_i, \ldots, B_{i+\ell-1}$.

We now argue that Bob's simulation of **DS** for operations in the right subtree of v (phases $j \in \{i+\ell, \ldots, i+2\ell-1\}$) is identical to Alice's execution. Consider the first time any cell is probed during Bob's execution of operations in v's right subtree. Either the cell is read from Alice's encoding of $C_v(O)$ or the cell is read from snapshot$'$. Bob's execution will be different from Alice if and only if Bob uses the incorrect contents of a cell when first probed. This only happens if Bob uses the contents of a cell from snapshot$'$ yet that cell was overwritten by an operation in the left subtree of v (phases $j \in \{i, \ldots, i+\ell-1\}$). If this were the case, this cell probe is assigned to v and, thus, the cell contents would have been encoded by Alice in $C_v(O)$. As a result, we know both executions by Alice and Bob are identical and Bob successfully decodes all answers in v's right subtree.

Analysis. We now analyze the expected length of the encoding and show that the expected size of Alice's encoding is smaller than the entropy of the bipartite graphs decoded by Bob contradicting Shannon's source coding theorem.

We distinguish two cases. In the case that Alice's encoding starts with a 0-bit, the length is exactly $1 + \frac{n}{2^{d+1}} \cdot \lg \binom{n^\epsilon}{l}$ bits. Let us upper bound probability that Alice produces an encoding that starts with 0. There are two cases in which this happens. In the first case, it is because F is large and thus **DS** made too many errors. Since **DS** has error probability at most 1/128, we know that $\mathsf{E}[|F(\mathsf{Hard}_v)|] \leq (1/128)n/2^{d+1}$ by linearity of expectation. By Markov's inequality, it follows that $\Pr[|F(\mathsf{Hard}_v)| \geq (1/32)n/2^{d+1}] \leq 1/4$. In the second case, $C_v(O)$ is too large and, by our assumption towards a contradiction, this happens with probability at most 1/2. Therefore, Alice's encoding starts with a 0-bit with probability at most 3/4. Let us now analyze the expected length of an encoding that starts with a 1-bit.

(a) Client storage is encoded using c bits. Recall that we chose $8 \leq d \leq (1/2)(1-\epsilon)\lg(n/c)$. As a result, we know that
$$c \leq \frac{n}{2^{2d}} \leq \frac{1}{2^{d-1}} \cdot \frac{nl}{2^{d+1}} \leq \frac{1}{128} \cdot \frac{n}{2^{d+1}} \cdot \lg \binom{n^\epsilon}{l}.$$

(b) $|F| \leq n$ and thus $\Theta(\lg n)$ bits are needed;

(c) The indices and the answers for the failed queries are encoded using
$$\Theta(\lg n) + \lg \binom{\frac{n}{2^{d+1}}}{|F|} + |F| \lg \binom{n^\epsilon}{l}$$
bits. The above encoding size increases as a function of $|F|$. The largest encoding occurs when $|F| = (1/32)n/2^{d+1}$. By substituting and adding the $\Theta(\lg n)$ bits from above items, we obtain
$$\Theta(\lg n) + \frac{1}{32} \cdot \frac{n}{2^{d+1}} \left(\lg(32e) + \lg \binom{n^\epsilon}{l} \right) \leq \frac{1}{16} \cdot \frac{n}{2^{d+1}} \cdot \lg \binom{n^\epsilon}{l}.$$

(d) $|C_v(O)| = O(\frac{n}{2^{d+1}} \lg \binom{n^\epsilon}{l}/w)$ and thus $\Theta(\lg n)$ bits are needed;

(e) By our contradiction assumption, the expected length of the encoding of $C_v(O)$ requires at most $(1/100) \cdot \frac{n}{2^{d+1}} \cdot \lg \binom{n^\epsilon}{l}$ bits. If we sum the $\Theta(\lg n)$ bits from (d) we obtain a total of
$$\frac{1}{100} \cdot \frac{n}{2^{d+1}} \cdot \lg \binom{n^\epsilon}{l}.$$

Altogether, the expected length of the encoding starting with a 1-bit is at most
$$\left(\frac{1}{128} + \frac{1}{16} + \frac{1}{50} \right) \cdot \frac{n}{2^{d+1}} \cdot \lg \binom{n^\epsilon}{l} < \frac{1}{8} \cdot \frac{n}{2^{d+1}} \cdot \lg \binom{n^\epsilon}{l}.$$

By putting the two cases together, we can conclude that the expected length of the encoding is at most $1 + (3/4 + 1/8) \cdot \frac{n}{2^{d+1}} \cdot \lg \binom{n^\epsilon}{l} < \frac{n}{2^{d+1}} \cdot \lg \binom{n^\epsilon}{l}$ which contradicts Shannon's source coding theorem thus completing the proof. □

We note the proof of Lemma 3 follows directly from Lemma 6 and Lemma 7. Thus, the proof of Theorem 3 is complete. See Section 5 for extending our results to even larger leakage functions. Also, see Section 6 for applying our results to searchable encryption.

Discussion 1. Previous works in the ORAM literature consider *passive* servers that act exclusively as storage that may only retrieve or update server memory. In this model, a cell probe corresponds to one cell of bandwidth. As a result, the above lower bounds can be interpreted as bandwidth lower bounds for passive servers. For servers with general computation (like we assumed in our work), cell probe lower bounds apply to server computation.

Discussion 2. As noted above, our lower bounds can be applied to the encrypted array primitive that is much closer to the ORAM primitive. One can interpret our leakage cell probe model with respect to the $\Omega(\lg n)$ ORAM lower bounds that appear in [49, 61]. In particular, our lower bounds show that the $\Omega(\lg n)$ overhead necessarily incurred by ORAMs is caused by mitigating the global key-equality pattern leakage. After mitigating global key-equality pattern leakage, other leakage mitigation by ORAMs such as operation type do not cost additional asymptotic overhead.

Discussion 3. We note that the efficiency of some previous schemes are evaluated for specific scenarios. For example, the schemes in [41] are evaluated assuming queries are drawn according to the Zipf's distribution. We note that our lower bounds do not apply to any scenarios where our hard distribution is not a valid input. Our lower bounds can be interpreted as if one wishes to leak at most $\mathcal{L}_{\mathsf{dec}}$, then one must either incur $\Omega(\lg n)$ overhead or only accept specific input distributions. We leave it as an interesting and important open question to study the efficiency schemes assuming specific distributions.

5 Stronger Lower Bounds

In this section, we describe two extensions of the lower bound of Section 4 to two more leakage functions, $\mathcal{L}_{\mathsf{get}}$ and $\mathcal{L}_{\mathsf{add}}$, that leak the same information as $\mathcal{L}_{\mathsf{dec}}$ and, additionally, $\mathcal{L}_{\mathsf{get}}$ reveals the keys that are used as inputs for all Get operations while $\mathcal{L}_{\mathsf{add}}$ reveals the keys and values that are used as inputs for all Add operations. In other words, $\mathcal{L}_{\mathsf{get}}$ enables **DS** to essentially perform Get operations in the plaintext. Similarly, $\mathcal{L}_{\mathsf{add}}$ enables **DS** to perform Add operations in the plaintext. As both functions $\mathcal{L}_{\mathsf{get}}$ and $\mathcal{L}_{\mathsf{add}}$ might have larger leakage than $\mathcal{L}_{\mathsf{dec}}$ (that is, $\mathcal{L}_{\mathsf{dec}} \leq \mathcal{L}_{\mathsf{get}}$ and $\mathcal{L}_{\mathsf{dec}} \leq \mathcal{L}_{\mathsf{add}}$), the lower bounds are not implied by the lower bound on $\mathcal{L}_{\mathsf{dec}}$. We formally define $\mathcal{L}_{\mathsf{add}}$ and $\mathcal{L}_{\mathsf{get}}$ below.

Definition 6 (Leakage function $\mathcal{L}_{\mathsf{add}}$)**.** *For a sequence $O = (\mathsf{op}_0 = \mathsf{Create}, \mathsf{op}_1, \ldots, \mathsf{op}_\ell)$ of operations where $\mathtt{key}_1, \ldots, \mathtt{key}_\ell$ are the input keys to each non-create operation, then the leakage $\mathcal{L}_{\mathsf{add}}(O)$ associated with O consists of $\mathcal{L}_{\mathsf{add}}(O) = (\mathcal{L}_{\mathsf{add}}(O_0), \ldots, \mathcal{L}_{\mathsf{add}}(O_\ell))$ where $O_i = (\mathsf{op}_0, \ldots, \mathsf{op}_i)$ and MM^{O_i} is the multi-map resulting from the first i operations. Then, $\mathcal{L}_{\mathsf{add}}(O_i)$ is defined as follows:*

1. *if $\mathsf{op}_i = \mathsf{Create}$ then $\mathcal{L}_{\mathsf{add}}(O_i) = (\mathsf{Create})$;*
2. *if $\mathsf{op}_i = \mathsf{Add}(\mathtt{key}_i, \mathtt{val}_i)$ then $\mathcal{L}_{\mathsf{add}}(O_i) = (\mathsf{Add}, \mathtt{key}_i, \mathtt{val}_i)$;*
3. *if $\mathsf{op}_i = \mathsf{Get}(\mathtt{key}_i)$ then $\mathcal{L}_{\mathsf{add}}(O_i) = (\mathsf{Get}, |\mathtt{vals}(\mathsf{MM}^{O_{i-1}}, \mathtt{key}_i)|, {\mathsf{ep}^{\mathsf{Get}}}_i)$.*

The get key-equality pattern ${\mathsf{ep}^{\mathsf{Get}}}_i := ({\mathsf{ep}^{\mathsf{Get}}}_{i,1}, \ldots, {\mathsf{ep}^{\mathsf{Get}}}_{i,i-1})$ *is defined as follows:*

$$
{\mathsf{ep}^{\mathsf{Get}}}_{i,j} = \begin{cases} \bot, & \textit{if } \mathsf{op}_j \textit{ is an } \mathsf{Add} \textit{ operation;} \\ 0, & \textit{if } \mathsf{op}_j \textit{ is a } \mathsf{Get} \textit{ operation and } \mathtt{key}_i \neq \mathtt{key}_j; \\ 1, & \textit{if } \mathsf{op}_j \textit{ is a } \mathsf{Get} \textit{ operation and } \mathtt{key}_i = \mathtt{key}_j; \end{cases}
$$

Definition 7 (Leakage function $\mathcal{L}_{\mathsf{get}}$)**.** *For a sequence $O = (\mathsf{op}_0 = \mathsf{Create}, \mathsf{op}_1, \ldots, \mathsf{op}_\ell)$ of operations where $\mathtt{key}_1, \ldots, \mathtt{key}_\ell$ are the input keys to each non-create operation, then the leakage $\mathcal{L}_{\mathsf{get}}(O)$ associated with O consists of $\mathcal{L}_{\mathsf{get}}(O) = (\mathcal{L}_{\mathsf{get}}(O_1), \ldots, \mathcal{L}_{\mathsf{get}}(O_l))$ where $O_i = (\mathsf{op}_0, \ldots, \mathsf{op}_i)$ and MM^{O_i} is the multi-map resulting from the first i operations. Then, $\mathcal{L}_{\mathsf{get}}(O_i)$ is defined as follows:*

1. *if $\mathsf{op}_i = \mathsf{Create}$, then $\mathcal{L}_{\mathsf{get}}(O_i) = (\mathsf{Create})$;*
2. *if $\mathsf{op}_i = \mathsf{Add}(\mathtt{key}_i, \mathtt{val}_i)$ then $\mathcal{L}_{\mathsf{get}}(O_i) = (\mathsf{Add}, \mathsf{ep}^{\mathsf{Add}})$;*
3. *if $\mathsf{op}_i = \mathsf{Get}(\mathtt{key}_i)$ then $\mathcal{L}_{\mathsf{get}}(O_i) = (\mathsf{Get}, \mathtt{key}_i, |\mathtt{vals}(\mathsf{MM}^{O_{i-1}}, \mathtt{key}_i)|)$.*

The add key-equality pattern $\mathsf{ep^{Add}}_i := (\mathsf{ep^{Add}}_{i,1}, \ldots, \mathsf{ep^{Add}}_{i,i-1})$ *is defined as follows:*

$$\mathsf{ep^{Add}}_{i,j} = \begin{cases} \perp, & \textit{if } \mathtt{op}_j \textit{ is a } \mathsf{Get} \textit{ operation.} \\ 0, & \textit{if } \mathtt{op}_j \textit{ is an } \mathsf{Add} \textit{ operation and } \mathtt{key}_i \neq \mathtt{key}_j. \\ 1, & \textit{if } \mathtt{op}_j \textit{ is an } \mathsf{Add} \textit{ operation and } \mathtt{key}_i = \mathtt{key}_j. \end{cases}$$

We provide some intuition why our lower bounds from Section 4 also apply to these larger leakage functions. Recall that the $\mathcal{L}_{\mathsf{dec}}$ leakage function decouples the Get key-equality pattern and the Add key-equality pattern which allows an adversary to label each key according to the order in which it is used as input for a Get operation and the Add. In other words, the $\mathcal{L}_{\mathsf{dec}}$ leakage functions assigns each key two *pseudonyms*: one used to associate Get operations and another for Add operations. If we replace one of the two pseudonyms with the real key, the leakage of $\mathcal{L}_{\mathsf{dec}}$ should not be greatly changed. Therefore, if the actual keys of one of the Get or Add operations (but not both) are leaked, the key-equality patterns of the two operations remain decoupled. In this section, we will make this intuition formal by extending our lower bounds to $\mathcal{L}_{\mathsf{get}}$ and $\mathcal{L}_{\mathsf{add}}$.

5.1 Lower Bounds for $\mathcal{L}_{\mathsf{add}}$

Theorem 4. *Let* **DS** *be a* $\mathcal{L}_{\mathsf{add}}$*-leakage cell probe model dynamic encrypted multi-map implemented over* w*-bit cells and a client with* c *bits of storage. Then*

$$\mathsf{Eff}_{\mathbf{DS}}(n) = \Omega\left(\lg\left(\frac{n}{c}\right) \cdot \frac{\lg(n)}{w}\right).$$

Proof. The proof follows almost immediately from the proof of Theorem 3 with one extra observation. Note, that the swaps of Get operations performed in the proof of Theorem 3 to obtain Hard_v from Hard do not affect the leakage incurred by Add operations. As the leakage incurred by Get operations by $\mathcal{L}_{\mathsf{dec}}$ and $\mathcal{L}_{\mathsf{add}}$ are identical, we get that $\mathcal{L}_{\mathsf{add}}(\mathsf{Hard}_v)$ and $\mathcal{L}_{\mathsf{add}}(\mathsf{Hard})$ are identical; that is, a variant of Lemma 6 for $\mathcal{L}_{\mathsf{add}}$. As this is the only part of the proof of Theorem 3 that uses the leakage of **DS**, the proof follows similarly. □

5.2 Lower Bounds for $\mathcal{L}_{\mathsf{get}}$

Theorem 5. *Let* **DS** *be a* $\mathcal{L}_{\mathsf{get}}$*-leakage cell probe model dynamic encrypted multi-map implemented over* w*-bit cells and a client with* c *bits of storage. Then*

$$\mathsf{Eff}_{\mathbf{DS}}(n) = \Omega\left(\lg\left(\frac{n}{c}\right) \cdot \frac{\lg(n)}{w}\right).$$

Proof. We present the modifications of the proof of Theorem 3 needed to prove the result for $\mathcal{L}_{\mathsf{get}}$.

Unlike the proof of Theorem 4 for leakage function $\mathcal{L}_{\mathsf{add}}$, we need to redefine the notion $\mathcal{L}_{\mathsf{get}}$-invariant swaps used to obtain the hard distribution for each node v, Hard_v. It is not hard to see that the swaps of Get operations in the proof of Theorem 3 used to obtain Hard and Hard_v would result in the leakage distributions $\mathcal{L}_{\mathsf{get}}(\mathsf{Hard})$ and $\mathcal{L}_{\mathsf{get}}(\mathsf{Hard}_v)$ being different as $\mathcal{L}_{\mathsf{get}}$ leaks the inputs of Get operations in the plaintext.

For $s \leq s'$, we re-define distribution $\mathsf{Hard}^{(s,s')}$ by modifying Phase s and SubPhase $\mathsf{Init}_{s'}$ as follows:

- In Phase 0, SubPhase $\mathsf{Init}_{s'}$ will associated each key in K^{a}_s (instead of keys in $K^{\mathsf{g}}_{s'}$) with l uniformly random values from V_0.
- In Phase s, Add operations in SubPhase $\mathsf{A_j}$ will add l values from V_s chosen uniformly at random to each key in $K^{\mathsf{g}}_{s'}$ (instead of keys in K^{a}_s). The Get operations in SubPhase $\mathsf{G_s}$ query all keys in K^{g}_s just like in Hard.

Let us now convince ourselves that the swaps defined above are $\mathcal{L}_{\mathsf{get}}$-invariant. Indeed, observe that the Get operations are not affected by the swaps and that the values queried in phase s' are inserted in phase $s \leq s'$. Therefore the $\mathcal{L}_{\mathsf{get}}$ leakage derived from Get operations stays the same. For the Add operations instead observe that, both in Hard and $\mathsf{Hard}^{(s,s')}$, the same number k of consecutive Add operations are performed to distinct keys and thus the key-equality pattern does not change (even though the actual keys do differ).

The $\mathcal{L}_{\mathsf{get}}$-invariance continues to hold when performing any sequence of swaps $S := \{(s_1, s_1'), \dots, (s_\ell, s_\ell')\}$ such that $s_i \leq s_i'$, for $i = 1, \dots, \ell$ and each index appears at most once in S.

We proceed to define distribution Hard_v as the distribution resulting from performing swaps as defined by $\mathsf{swap}_v := \{(i, i+\ell), \dots, (i+\ell-1, i+2\ell-1)\}$ where $\{i, \dots, i+\ell-1\}$ ($\{i+\ell, \dots, i+2\ell-1\}$) refer to the subphases appearing the left (right) subtree of v.

The identical encoding argument of Theorem 3 may be used as the Get operations in the right subtree must retrieve values inserted randomly by Add operations in the left subtree in the identical manner. As a result, the proof is complete with the above modification of Hard_v. $\square$

6 Lower Bounds for Dynamic Searchable Encryption

In this section, we show that our lower bound for encrypted multi-maps can directly be interpreted as lower bounds for searchable encryption. In particular, our lower bounds apply to *response-hiding* searchable encryption schemes which do not reveal information about the matching documents to a queried keyword (except the number of matching documents). To do this, we show there exists a simple reduction that allows one to implement an encrypted multi-map using a searchable encryption scheme. Recall that encrypted multi-maps have two functions: $\mathsf{Get}(\mathtt{key})$ that retrieves all values associated with $\mathtt{key}$ and $\mathsf{Add}(\mathtt{key}, \mathtt{val})$ that adds value $\mathtt{val}$ to the tuple of values associated with $\mathtt{key}$. On the other hand, dynamic searchable encryption implements two functions: $\mathsf{Query}(\mathsf{kw})$ that retrieves all documents with keyword kw as well as $\mathsf{Insert}(d, \mathsf{Kws})$ that inserts all keywords in Kws into document d. We start by formally defining dynamic searchable encryption.

6.1 Definition of Dynamic Searchable Encryption

In this section, we present the problem of *dynamic searchable encryption*. This data structure problem abstracts the scenario in which a client wishes to outsource the storage of a corpus of *documents* in an encrypted manner. In addition, the client wishes to be able to search for *keywords* and retrieve the list of the identifiers of the documents containing the searched keyword. In the dynamic variant, the client is also able to update the corpus of documents stored by the server. This could include adding new documents, removing old documents and modifying documents. For privacy, the client's goal is to minimize the information that the server learns about stored documents and queried keywords.

We now formally describe the dynamic searchable encryption problem. A *document* is defined as a pair $(d, \mathsf{Kws}(d))$ where d is the identifier of the document and $\mathsf{Kws}(d)$ is the set of all unique keywords that appear in the document. The set of identifiers of all documents that are in the system is defined by Docs and we define the set of all document identifiers that contain keyword w as $\mathsf{Docs}(w) := \{d \in \mathsf{Docs} \mid w \in \mathsf{Kws}(d)\} \subseteq \mathsf{Docs}$. We extend $\mathsf{Kws}(D)$ to subsets $D \subseteq \mathsf{Docs}$ of documents in the natural way. That is, $\mathsf{Kws}(D) := \cup_{d \in D} \mathsf{Kws}(d)$. The goal of the dynamic searchable encryption problem is to construct a data structure that enables a client to perform queries for keywords w and obtain the set $\mathsf{Docs}(w)$. In this paper, we will consider the searchable encryption schemes that provide only minimal functionality for updating the stored corpus of documents. Specifically, we only consider schemes that support only the insertion of new keywords into a document mirroring our definitions of encrypted multi-maps. Since we are interested in proving lower bounds, assuming minimal functionality makes our results stronger as they will also hold for any schemes that provide a wider range of operations.

Definition 8. *The* dynamic searchable encryption *problem is parameterized by a finite alphabet Σ and an integer $t > 0$. Let $A = \Sigma^{\leq t}$ be the set of all strings of length at most t with letters from Σ. Let $S = 2^A$ be the power set over A. The problem is defined by the tuple $(\mathbb{U}, \mathbb{Q}, \mathbb{R})$ where*

- $\mathbb{U} = \{\mathsf{Insert}(d, \mathsf{Kws}(d)) \mid d \in A, \mathsf{Kws}(d) \in S\} \cup \{\mathsf{Create}\}$;
- $\mathbb{Q} = \{\mathsf{Query}(\mathsf{kw}) \mid \mathsf{kw} \in A\}$;

and for any sequence $O = (\mathsf{Create}, \mathsf{Insert}(d_1, \mathsf{Kws}(d_1)), \ldots, \mathsf{Insert}(d_m, \mathsf{Kws}(d_m)))$,

$$\mathbb{R}(O, \mathsf{Query}(\mathsf{kw})) = \{d_i \mid \mathsf{kw} \in \mathsf{Kws}(d_i), i \in [m]\} = \mathsf{Docs}(\mathsf{kw}).$$

Efficiency measure. We define our measures of efficiency for the dynamic searchable encryption problem similar to encrypted multi-maps. We measure *insert efficiency* as the number of cell probes performed per unique keyword appear in the inserted document. In other words, the insert efficiency is the number of cell probes performed to insert document d_i divided by the $|\mathsf{Kws}(d_i)|$. We will measure *query efficiency* as the number of cell probes performed per identifier in the query's answer. Formally, the query efficiency is the number of cell probes performed to answer a query for keyword kw divided by $|\mathsf{Docs}(\mathsf{kw})|$.

6.2 Lower Bounds

To show that our lower bounds from Section 5 extend to searchable encryption, we need to be able to answer both MM.Get and MM.Add encrypted multi-map operations given only SSE.Query and SSE.Insert encrypted search operations. To do this, we can simulate $\mathsf{MM.Add}(\mathtt{key}, \mathtt{val})$ by inserting `key` as a keyword into the document identified by `val`, $\mathsf{SSE.Insert}(\mathtt{val}, \{\mathtt{key}\})$. For simulating $\mathsf{MM.Get}(\mathtt{key})$, we execute and return the answer of $\mathsf{SSE.Query}(\mathtt{key})$. As a result, $\mathsf{MM.Get}(\mathtt{key})$ returns all values `val` such that $\mathsf{Add}(\mathtt{key}, \mathtt{val}) = \mathsf{SSE.Insert}(\mathtt{val}, \{\mathtt{key}\})$ was executed. As one can simulate an encrypted multi-map using searchable encryption schemes, the encrypted multi-map lower bounds also apply to searchable encryption. In particular, we can directly interpret the lower bounds from Section 5 into searchable encryption.

We start by interpreting the leakage of $\mathcal{L}_{\mathsf{add}}$ with respect to encrypted multi-maps as leakage $\mathcal{L}^{\mathsf{SSE}}_{\mathsf{add}}$ for a searchable encryption scheme. In particular, $\mathcal{L}^{\mathsf{SSE}}_{\mathsf{add}}$ enables a searchable encryption scheme to perform all Insert operations that insert new documents completely in the plaintext. That is, the document identifier as well as the inserted keywords may be revealed to the adversary in the plaintext. For Query operations, the searchable encryption scheme may only reveal the number of documents containing the queried keyword as well as the identity of previous Query operations that used the same keyword as input (that is, keyword-equality leakage). We now formally define $\mathcal{L}^{\mathsf{SSE}}_{\mathsf{add}}$:

Definition 9 (Leakage function $\mathcal{L}^{\mathsf{SSE}}_{\mathsf{add}}$)**.** *For a sequence* $O = (\mathsf{op}_0 = \mathsf{Create}, \mathsf{op}_1, \ldots, \mathsf{op}_\ell)$ *of operations, then the leakage* $\mathcal{L}^{\mathsf{SSE}}_{\mathsf{add}}(O)$ *associated with* O *consists of* $\mathcal{L}^{\mathsf{SSE}}_{\mathsf{add}}(O) = (\mathcal{L}^{\mathsf{SSE}}_{\mathsf{add}}(O_0), \ldots, \mathcal{L}^{\mathsf{SSE}}_{\mathsf{add}}(O_\ell))$ *where* $O_i = (\mathsf{op}_0, \ldots, \mathsf{op}_i)$ *and* SSE^{O_i} *is the multi-map resulting from the first* i *operations. Then,* $\mathcal{L}^{\mathsf{SSE}}_{\mathsf{add}}(O_i)$ *is defined as follows:*

1. *if* $\mathsf{op}_i = \mathsf{Create}$ *then* $\mathcal{L}^{\mathsf{SSE}}_{\mathsf{add}}(O_i) = (\mathsf{Create})$;
2. *if* $\mathsf{op}_i = \mathsf{Insert}(d_i, \mathsf{Kws}_i)$ *then* $\mathcal{L}^{\mathsf{SSE}}_{\mathsf{add}}(O_i) = (\mathsf{Insert}, d_i, \mathsf{Kws}_i)$;
3. *if* $\mathsf{op}_i = \mathsf{Query}(\mathsf{kw}_i)$ *then* $\mathcal{L}^{\mathsf{SSE}}_{\mathsf{add}}(O_i) = (\mathsf{Query}, |\mathsf{SSE}^{O_{i-1}}.\mathsf{Query}(\mathsf{kw}_i)|, {\mathsf{ep}^{\mathsf{Query}}}_i)$.

The query keyword-equality pattern ${\mathsf{ep}^{\mathsf{Query}}}_i := ({\mathsf{ep}^{\mathsf{Query}}}_{i,1}, \ldots, {\mathsf{ep}^{\mathsf{Query}}}_{i,i-1})$ *is defined as follows:*

$$\mathsf{ep}^{\mathsf{Query}} i, j = \begin{cases} \perp, & \text{if } \mathsf{op}_j \text{ is an Insert operation.} \\ 0, & \text{if } \mathsf{op}_j \text{ is a Query operation and } \mathsf{kw}_i \neq \mathsf{kw}_j. \\ 1, & \text{if } \mathsf{op}_j \text{ is a Query operation and } \mathsf{kw}_i = \mathsf{kw}_j. \end{cases}$$

Theorem 6. *Let* SSE *be a* $\mathcal{L}^{\mathsf{SSE}}_{\mathsf{add}}$*-leakage cell probe model searchable encryption scheme implemented over* w*-bit cells and a client with* c *bits of storage. Then,*

$$\mathsf{Eff}_{\mathsf{SSE}}(n) = \Omega\left(\lg\left(\frac{n}{c}\right) \cdot \frac{\lg(n)}{w}\right).$$

Similarly, we may also interpret $\mathcal{L}_{\mathsf{get}}$ with respect to the leakage of searchable encryption schemes, $\mathcal{L}_{\mathsf{get}}^{\mathsf{SSE}}$. As a dual of $\mathcal{L}_{\mathsf{add}}^{\mathsf{SSE}}$, schemes with leakage $\mathcal{L}_{\mathsf{get}}^{\mathsf{SSE}}$ are able to perform Query operations that query for keywords completely in the plaintext. That is, the queried keyword may be revealed in the plaintext. As for the results of Query operations, the adversary may only learn the number of documents that match the queried keyword. In terms of leakage incurred by Insert operations, the adversary learns the number of inserted keywords. Additionally, for each pair of Insert operations, the adversary learns the number of identical keywords inserted in both operations (that is, a generalization of keyword-equality leakage). The formal definition of $\mathcal{L}_{\mathsf{get}}^{\mathsf{SSE}}$ is below:

Definition 10 (Leakage function $\mathcal{L}_{\mathsf{get}}^{\mathsf{SSE}}$)**.** *For a sequence $O = (\mathsf{op}_0 = \mathsf{Create}, \mathsf{op}_1, \ldots, \mathsf{op}_\ell)$ of operations, then the leakage $\mathcal{L}_{\mathsf{get}}^{\mathsf{SSE}}(O)$ associated with O consists of $\mathcal{L}_{\mathsf{get}}^{\mathsf{SSE}}(O) = (\mathcal{L}_{\mathsf{get}}^{\mathsf{SSE}}(O_0), \ldots, \mathcal{L}_{\mathsf{get}}^{\mathsf{SSE}}(O_\ell))$ where $O_i = (\mathsf{op}_0, \ldots, \mathsf{op}_i)$ and SSE^{O_i} is the multi-map resulting from the first i operations. Then, $\mathcal{L}_{\mathsf{get}}^{\mathsf{SSE}}(O_i)$ is defined as follows:*

1. *if $\mathsf{op}_i = \mathsf{Create}$ then $\mathcal{L}_{\mathsf{get}}^{\mathsf{SSE}}(O_i) = (\mathsf{Create})$;*
2. *if $\mathsf{op}_i = \mathsf{Insert}(d_i, \mathsf{Kws}_i)$ then $\mathcal{L}_{\mathsf{get}}^{\mathsf{SSE}}(O_i) = (\mathsf{Insert}, |\mathsf{Kws}(d_i)|, {\mathsf{ep}^{\mathsf{Insert}}}_i)$;*
3. *if $\mathsf{op}_i = \mathsf{Query}(\mathsf{kw}_i)$ then $\mathcal{L}_{\mathsf{get}}^{\mathsf{SSE}}(O_i) = (\mathsf{Query}, \mathsf{kw}_i, |\mathsf{SSE}^{O_{i-1}}.\mathsf{Query}(\mathsf{kw}_i)|)$.*

The insert keyword-equality pattern ${\mathsf{ep}^{\mathsf{Insert}}}_i := ({\mathsf{ep}^{\mathsf{Insert}}}_{i,1}, \ldots, {\mathsf{ep}^{\mathsf{Insert}}}_{i,i-1})$ *is defined as follows:*

$${\mathsf{ep}^{\mathsf{Insert}}}_{i,j} = \begin{cases} \bot, & \textit{if } \mathsf{op}_j \textit{ is a } \mathsf{Query} \textit{ operation.} \\ |\mathsf{Kws}_i \cap \mathsf{Kws}_j|, & \textit{if } \mathsf{op}_j \textit{ is an } \mathsf{Insert} \textit{ operation.} \end{cases}$$

Theorem 7. *Let SSE be a $\mathcal{L}_{\mathsf{get}}^{\mathsf{SSE}}$-leakage cell probe model searchable encryption scheme implemented over w-bit cells and a client with c bits of storage. Then,*

$$\mathsf{Eff}_{\mathsf{SSE}}(n) = \Omega\left(\lg\left(\frac{n}{c}\right) \cdot \frac{\lg(n)}{w}\right).$$

7 Conclusions

To summarize, our work presents the first formal tradeoff analysis of efficiency and privacy for encrypted multi-maps as well as searchable encryption schemes. In particular, we show that mitigating the global key-equality pattern leakage (even in a very small manner) fundamentally incurs an $\Omega(\lg n)$ overhead. We show our lower bounds hold even when the encrypted multi-map is able to perform one of the Add or Get operations in plaintext. These results may be applied to the setting of searchable encryption where we show that dynamic schemes that are response-hiding also must use $\Omega(\lg n)$ overhead even when one of the document updates or searches may be performed in the plaintext.

In terms of techniques, our paper introduces several new ideas that may be widely applicable. First, we introduce the notion of the leakage cell probe model that allows proving lower bounds for structured encryption with arbitrary leakage profiles. Next, our lower bounds apply to the setting where the data structure contents do not necessarily have to be random such as the keywords that appear in documents. Finally, we present new methods to construct hard distributions even when considering much larger leakage profiles than previous results. We believe these techniques may be helpful in analyzing the efficiency and privacy tradeoffs for many other primitives.

References

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

[2] G. Asharov, M. Naor, G. Segev, and I. Shahaf. Searchable symmetric encryption: Optimal locality in linear space via two-dimensional balanced allocations. In *STOC '16*, 2016.

[3] G. Asharov, G. Segev, and I. Shahaf. Tight tradeoffs in searchable symmetric encryption. In *CRYPTO '18*, 2018.

[4] M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In *CRYPTO '07*, 2007.

[5] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill. Order-preserving symmetric encryption. In *EUROCRYPTO '09*, 2009.

[6] A. Boldyreva, N. Chenette, and A. O'Neill. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In *CRYPTO '11*, 2011.

[7] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *EUROCRYPT '04*, pages 506–522, 2004.

[8] D. Boneh, K. Lewi, M. Raykova, A. Sahai, M. Zhandry, and J. Zimmerman. Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation. In *EUROCRYPT '15*, 2015.

[9] D. Boneh, D. Maziéres, and R. A. Popa. Remote oblivious storage: Making oblivious RAM practical. Technical report, CSAIL, MIT, 2011.

[10] R. Bost. Sophos: Forward secure searchable encryption. In *CCS '16*, 2016.

[11] R. Bost, B. Minaud, and O. Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. In *CCS '17*, 2017.

[12] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart. Leakage-abuse attacks against searchable encryption. In *CCS '15*, 2015.

[13] D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner. Dynamic searchable encryption in very-large databases: data structures and implementation. In *NDSS*, volume 14, pages 23–26, 2014.

[14] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *CRYPTO '13*, pages 353–373. Springer, 2013.

[15] D. Cash and S. Tessaro. The locality of searchable symmetric encryption. In *EUROCRYPT '14*, pages 351–368. Springer, 2014.

[16] M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *EUROCRYPT '10*.

[17] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. *Journal of Computer Security*, 2011.

[18] I. Demertzis, D. Papadopoulos, and C. Papamanthou. Searchable encryption with optimal locality: Achieving sublogarithmic read efficiency. In *CRYPTO '18*, 2018.

[19] I. Demertzis and C. Papamanthou. Fast searchable encryption with tunable locality. In *SIGMOD '17*, 2017.

[20] I. Demertzis, R. Talapatra, and C. Papamanthou. Efficient searchable encryption through compression. *Proc. VLDB Endow.*, 11(11):1729–1741, July 2018.

[21] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert, and R. E. Tarjan. Dynamic perfect hashing: Upper and lower bounds. *SIAM J. Comput.*, 23(4):738–761, Aug. 1994.

[22] M. Etemad, A. Küpçü, C. Papamanthou, and D. Evans. Efficient dynamic searchable encryption with forward privacy. *PETS '18*, 2018.

[23] B. A. Fisch, B. Vo, F. Krell, A. Kumarasubramanian, V. Kolesnikov, T. Malkin, and S. M. Bellovin. Malicious-client security in blind seer: a scalable private dbms. In *Security and Privacy (SP), 2015 IEEE Symposium on*, 2015.

[24] M. Fredman and M. Saks. The cell probe complexity of dynamic data structures. In *Proceedings of the 21st annual ACM symposium on Theory of computing*, 1989.

[25] S. Garg, P. Mohassel, and C. Papamanthou. TWORAM: Round-optimal oblivious RAM with applications to searchable encryption. In *CRYPTO '16*, 2016.

[26] C. Gentry. Fully Homomorphic Encryption Using Ideal Lattices. In *STOC '09*.

[27] J. Ghareh Chamani, D. Papadopoulos, C. Papamanthou, and R. Jalili. New constructions for forward and backward private symmetric searchable encryption. In *CCS '18*, 2018.

[28] E.-J. Goh. Secure indexes. *IACR Cryptology ePrint Archive*, 2003:216, 2003.

[29] O. Goldreich. Towards a theory of software protection and simulation by oblivious RAMs. In *STOC '87*, 1987.

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

[31] M. T. Goodrich, M. Mitzenmacher, O. Ohrimenko, and R. Tamassia. Practical oblivious storage. In *Proceedings of the second ACM conference on Data and Application Security and Privacy*, 2012.

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

[33] P. Grubbs, T. Ristenpart, and V. Shmatikov. Why your encrypted database is not secure. In *HotOS '17*, 2017.

[34] A. Hamlin, A. Shelat, M. Weiss, and D. Wichs. Multi-key searchable encryption, revisited. In *PKC '18*, 2018.

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

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

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

[38] S. Kamara and T. Moataz. Boolean searchable symmetric encryption with worst-case sub-linear complexity. In *EUROCRYPT '17*, pages 94–124, 2017.

[39] S. Kamara and T. Moataz. SQL on structurally-encrypted databases. In *ASIACRYPT '18*, 2018.

[40] S. Kamara and T. Moataz. Computationally volume-hiding structured encryption. In *EUROCRYPT '19*, 2019.

[41] S. Kamara, T. Moataz, and O. Ohrimenko. Structured encryption and leakage suppression. In *CRYPTO '18*, 2018.

[42] S. Kamara and C. Papamanthou. Parallel and dynamic searchable symmetric encryption. In *FC '13*.

[43] S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *CCS '12*, 2012.

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

[45] E. M. Kornaropoulos, C. Papamanthou, and R. Tamassia. The state of the uniform: Attacks on encrypted databases beyond the uniform query distribution. Cryptology ePrint Archive, Report 2019/441, 2019. `https://eprint.iacr.org/2019/441`.

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

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

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

[49] K. G. Larsen and J. B. Nielsen. Yes, there is an Oblivious RAM lower bound! In *CRYPTO '18*, 2018.

[50] K. G. Larsen, M. Simkin, and K. Yeo. Lower bounds for multi-server oblivious rams. Cryptology ePrint Archive, Report 2019/1108, 2019. `https://eprint.iacr.org/2019/1108`.

[51] K. G. Larsen, O. Weinstein, and H. Yu. Crossing the logarithmic barrier for dynamic boolean data structure lower bounds. In *STOC '18*, 2018.

[52] K. Lewi and D. J. Wu. Order-revealing encryption: New constructions, applications, and lower bounds. In *CCS '16*, 2016.

[53] S. Lu and R. Ostrovsky. How to garble RAM programs? In *EUROCRYPT '13*.

[54] I. Miers and P. Mohassel. IO-DSSE: Scaling dynamic searchable encryption to millions of indexes by improving locality. *IACR Cryptology ePrint Archive*, 2016.

[55] M. Naveed, S. Kamara, and C. V. Wright. Inference attacks on property-preserving encrypted databases. In *CCS '15*, 2015.

[56] R. Pagh. *Hashing, randomness and dictionaries.* BRICS, 2002.

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

[58] S. Patel, G. Persiano, and K. Yeo. Symmetric searchable encryption with sharing and unsharing. In *European Symposium on Research in Computer Security*, 2018.

[59] S. Patel, G. Persiano, and K. Yeo. What storage access privacy is achievable with small overhead? In *PODS'19*, 2019.

[60] S. Patel, G. Persiano, K. Yeo, and M. Yung. Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing. In *CCS '19*, 2019.

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

[62] R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: protecting confidentiality with encrypted query processing. In *SOSP '11*, 2011.

[63] D. Pouliot and C. V. Wright. The shadow nemesis: Inference attacks on efficiently deployable, efficiently searchable encryption. In *CCS '16*, 2016.

[64] M. Pătraşcu and E. D. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM Journal on Computing*, 2006.

[65] D. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *Proceedings of IEEE Symposium on Security and Privacy*, 2000.

[66] E. Stefanov, C. Papamanthou, and E. Shi. Practical dynamic searchable encryption with small leakage. In *NDSS*, volume 71, pages 72–75, 2014.

[67] E. Stefanov, M. Van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas. Path ORAM: an extremely simple oblivious RAM protocol. In *CCS '13*, 2013.

[68] S.-F. Sun, X. Yuan, J. K. Liu, R. Steinfeld, A. Sakzad, V. Vo, and S. Nepal. Practical backward-secure searchable encryption from symmetric puncturable encryption. In *CCS '18*, 2018.

[69] A. C.-C. Yao. Should tables be sorted? *Journal of the ACM*, 1981.

[70] Y. Zhang, J. Katz, and C. Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In *USENIX Security Symposium*, pages 707–720, 2016.

A Folklore ORAM-based Minimal Leakage Construction

We define following minimal leakage function $\mathcal{L}_{\mathsf{min}}$ as follows:

Definition 11 (Minimal Leakage $\mathcal{L}_{\mathsf{min}}$)**.** *For a sequence $O = (\mathtt{op}_0 = \mathsf{Create}, \mathtt{op}_1, \ldots, \mathtt{op}_\ell)$ of operations where $\mathtt{key}_1, \ldots, \mathtt{key}_\ell$ are the input keys to each non-create operation, then the leakage $\mathcal{L}_{\mathsf{min}}(O)$ associated with O consists of $\mathcal{L}_{\mathsf{min}}(O) = (\mathcal{L}_{\mathsf{min}}(O_0), \ldots, \mathcal{L}_{\mathsf{min}}(O_\ell))$, where $O_i = (\mathtt{op}_0, \ldots, \mathtt{op}_i)$ and MM^{O_i} is the multi-map resulting from the first i operations. Then, $\mathcal{L}_{\mathsf{min}}(O_i)$ is defined as follows:*

1. *If $\mathtt{op}_i = \mathsf{Create}$, then $\mathcal{L}_{\mathsf{min}}(O_i) = (\mathsf{Create})$;*
2. *If $\mathtt{op}_i = \mathsf{Add}(\mathtt{key}_i, \mathtt{val}_i)$ then $\mathcal{L}_{\mathsf{min}}(O_i) = (\mathsf{Add})$;*
3. *If $\mathtt{op}_i = \mathsf{Get}(\mathtt{key}_i)$ then $\mathcal{L}_{\mathsf{min}}(O_i) = (\mathsf{Get}, |\mathtt{vals}(\mathsf{MM}^{O_{i-1}}, \mathtt{key}_i)|)$.*

We present the folklore construction of encrypted multi-maps with leakage $\mathcal{L}_{\mathsf{min}}$ using oblivious RAMs (ORAMs) [30]. To do this, we first note that recent logarithmic ORAM constructions [57, 1] are also constructions of oblivious key-value storage without any modifications or increase in overhead. Oblivious key-value storage were considered previously under the name of oblivious storage in [9, 31].

A.1 Construction

We now present OptMM using any oblivious key-value storage implementation OKVS as a blackbox. We assume that we have an upper bound, n, on the sum of the lengths of the tuples associated with the keys. OptMM uses two independent instances of OKVS denoted by OKVS`key` and OKVScnt. OKVScnt maintains the count of the number of values associated with each key. OKVS`key` will store all values associated with each key. Both OKVS are initialized with the capacity to store at most n unique keys and each initialization returns an handle for the instance and a key.

OptMM.Add receives as input a key `key` and a value `val`. First, the number of tuples already associated with `key`, denoted by $\mathtt{cnt}_{\mathtt{key}}$, is retrieved from OKVScnt. If `key` does not exist in $\mathtt{cnt}_{\mathtt{key}}$, then set $\mathtt{cnt}_{\mathtt{key}}$

equal to 1; otherwise increment it by one and store the pair key-value $(\texttt{key}, \textsf{cnt}_{\texttt{key}})$ in $\textsf{OKVScnt}$. Finally, the key-value pair $(\texttt{key}||\textsf{cnt}_{\texttt{key}}, \texttt{val})$ is inserted into $\textsf{OKVSkey}$. Note, this key-value pair does not exist in $\textsf{OKVSkey}$ as all previous occurrences of $\texttt{key}$, if any, are inserted with smaller counts.

$\textsf{OptMM.Get}$ will be given a key $\texttt{key}$ as input. First, the number of documents containing, $\textsf{cnt}_{\texttt{key}}$, is retrieved from $\textsf{OKVScnt}$. Next, the keys $\{\texttt{key}||1, \texttt{key}||2, \ldots, \texttt{key}||\textsf{cnt}_{\texttt{key}}\}$ are queried from $\textsf{OKVSdoc}$. The associated identifiers are returned. Note, they correspond exactly to the values associated with $\texttt{key}$.

We now present the $\textsf{OptMM}$ formally. We also describe the initialization function $\textsf{OptMM.Create}$ that takes as input the bound n on the maximum number of document-keyword pairs and initializes both $\textsf{OKVS}$ instances.

```
OptMM.Create($n$) :
  1. Initialize $(\textsf{K}_{\textsf{cnt}}, \textsf{OKVScnt}) \leftarrow \textsf{OKVS.Create}(1^n)$.
  2. Initialize $(\textsf{K}_{\textsf{doc}}, \textsf{OKVSdoc}) \leftarrow \textsf{OKVS.Create}(1^n)$.
OptMM.Add($\textsf{K}_{\textsf{cnt}}, \textsf{K}_{\textsf{doc}}, \texttt{key}, \texttt{val}$) :
  1. Retrieve $\textsf{cnt}_{\texttt{key}} \leftarrow \textsf{OKVScnt}.\texttt{get}(\textsf{K}_{\textsf{cnt}}, \texttt{key})$.
  2. If $\textsf{cnt}_{\texttt{key}} \neq \perp$, set $\textsf{cnt}_{\texttt{key}} = \textsf{cnt}_{\texttt{key}} + 1$. Otherwise, set $\textsf{cnt}_{\texttt{key}} = 1$.
  3. Execute $\textsf{OKVScnt}.\texttt{update}(\textsf{K}_{\textsf{cnt}}, \texttt{key}, \textsf{cnt}_{\texttt{key}})$.
  4. Execute $\textsf{OKVSdoc}.\texttt{update}(\textsf{K}_{\textsf{doc}}, \texttt{key}||\textsf{cnt}_{\texttt{key}}, \texttt{val})$.
OptMM.Query($\textsf{K}_{\textsf{cnt}}, \textsf{K}_{\textsf{doc}}, \texttt{key}$) :
  1. Set $R \leftarrow \emptyset$.
  2. Retrieve $\textsf{cnt}_{\texttt{key}} \leftarrow \textsf{OKVScnt}.\texttt{get}(\textsf{K}_{\textsf{cnt}}, \texttt{key})$. If $\textsf{cnt}_{\texttt{key}} = \perp$, return $R$.
  3. For each $i \in \{1, \ldots, \textsf{cnt}_{\texttt{key}}\}$:
      (a) Set $R \leftarrow R \cup \{\textsf{OKVSkey}.\texttt{get}(\textsf{K}_{\textsf{doc}}, \texttt{key}||i)\}$.
  4. Return $R$.
```

Let us now investigate the leakage of $\textsf{OptMM}$. Each $\textsf{OKVS}$ leaks the number of operations performed and the maximum capacity of unique keys. The latter is exactly n and thus it is not additional leakage. Therefore, we only focus on the number of executions of each $\textsf{OKVS}$. We note that $\textsf{OKVSkey}$ is invoked twice during $\textsf{OptMM.Add}$ and once during $\textsf{OptMM.Get}$. This leaks the type of operations which also appears in $\mathcal{L}_{\textsf{min}}$. Similarly, the executions of $\textsf{OKVSkey}$ reveals the number of values returned during $\textsf{OptMM.Get}$.

Theorem 8. $\textsf{OptMM}$ *is a dynamic searchable encryption scheme with leakage* $\mathcal{L}_{\textsf{min}}$ *and* $O(\lg n)$ *amortized efficiency consisting of* $O(1)$ $\textsf{OKVS}$ *invocations.*

As a result, $\textsf{OptMM}$ achieves optimal efficiency for a leakage function $\mathcal{L}_{\textsf{min}}$ that is strictly smaller than the leakage profiles $\mathcal{L}_{\textsf{dec}}$, $\mathcal{L}_{\textsf{add}}$ and $\mathcal{L}_{\textsf{get}}$.

B Hash-and-Encrypt Compiler with Global Key-Equality Leakage

In this section, we present a simple *hash-and-encrypt* compiler that can compile plaintext arrays and multi-maps into encrypted versions with at most *global key-equality pattern* leakage. We start by formally defining the global key-equality pattern leakage, $\mathcal{L}_{\textsf{glob}}$.

Definition 12 (Global Leakage $\mathcal{L}_{\textsf{glob}}$)**.** *For a sequence* $O = (\textsf{op}_0 = \textsf{Create}, \textsf{op}_1, \ldots, \textsf{op}_\ell)$ *of operations where* $\texttt{key}_1, \ldots, \texttt{key}_\ell$ *are the input keys to each non-create operation, then the* global leakage function $\mathcal{L}_{\textsf{glob}}(O)$ *associated with* O *consists of* $\mathcal{L}_{\textsf{glob}}(O) = (\mathcal{L}_{\textsf{glob}}(O_0), \ldots, \mathcal{L}_{\textsf{glob}}(O_\ell))$ *where* $O_i = (\textsf{op}_0, \ldots, \textsf{op}_i)$ *and* $\textsf{MM}^{O_i}$ *is the multi-map resulting from the first i operations. Then,* $\mathcal{L}_{\textsf{glob}}(O_i)$ *is defined as follows:*

1. *If* $\mathsf{op}_i = \mathsf{Create}$ *then* $\mathcal{L}_{\mathsf{glob}}(O_i) = (\mathsf{Create})$*;*
2. *If* $\mathsf{op}_i = \mathsf{Add}(\mathtt{key}_i, \mathtt{val}_i)$ *then* $\mathcal{L}_{\mathsf{glob}}(O_i) = (\mathsf{Add}, \mathsf{ep^{glob}}_i)$*;*
3. *If* $\mathsf{op}_i = \mathsf{Get}(\mathtt{key}_i)$ *then* $\mathcal{L}_{\mathsf{glob}}(O_i) = (\mathsf{Get}, |\mathtt{vals}(\mathsf{MM}^{O_{i-1}}, \mathtt{key}_i)|, \mathsf{ep^{glob}}_i)$*.*

The global key-equality pattern $\mathsf{ep^{glob}}_i := (\mathsf{ep^{glob}}_{i,1}, \ldots, \mathsf{ep^{glob}}_{i,i-1})$ *is defined as follows:*

$$\mathsf{ep^{glob}}_{i,j} = \begin{cases} 1, & \textit{if } \mathtt{key}_i = \mathtt{key}_j. \\ 0, & \textit{if } \mathtt{key}_i \neq \mathtt{key}_j. \end{cases}$$

The main idea of the *hash-and-encrypt* compiler is to replace the plaintext keys and values for an encrypted multi-map with hashed keys and encrypted values. Then, the plaintext version just operates over the hashes and encryptions. One could define this compiler in general. Instead, we use a concrete example for simplicity. Consider any $O(1)$ overhead encrypted multi-map such as dynamic perfect hashing in [21] that has operations $DPH.\mathsf{Get}$ and $DPH.\mathsf{Add}$ operations. We generate a key K_1 for a collision-resistant hash function and a key K_2 for an IND-CPA encryption scheme. All $\mathsf{Get}(\mathtt{key})$ operations of the encrypted multi-map scheme are implemented by simply executing $DPH.\mathsf{Get}(\mathcal{H}(K_1, \mathtt{key}))$. Similarly, $\mathsf{Add}(\mathtt{key}, \mathtt{val})$ operations are implemented by executing $DPH.\mathsf{Add}(\mathcal{H}(K_1, \mathtt{key}), \mathcal{E}(K_2, \mathtt{val}))$. We get the following:

Theorem 9. *If one-way functions and collision resistant hash functions exist then there exists a non-adaptively* $\mathcal{L}_{\mathsf{glob}}$*-IND* **DS** *that solves the dynamic encrypted multi-map problem and has constant amortized efficiency.*

Proof. The efficiency follows immediately. For privacy, we note that the adversarial server sees $\mathcal{H}(K_1, \mathtt{key})$. As a result, the adversary learns the global key-equality pattern. Additionally, the server learns the operation type as well as the number of encrypted values associated with each `key`. $\square$

We note that the hash-and-encrypt is not novel as it has appeared implicitly in many previous works [17, 43, 40, 59].