Injection-Secure Structured and Searchable Symmetric Encryption

Ghous Amjad*
Google

Seny Kamara†
MongoDB & Brown University

Tarik Moataz‡
MongoDB

Abstract

Recent work on dynamic structured and searchable symmetric encryption has focused on achieving the notion of forward-privacy. This is mainly motivated by the claim that forward privacy protects against adaptive file injection attacks (Zhang, Katz, Papamanthou, *Usenix Security, 2016*). In this work, we revisit the notion of forward-privacy in several respects. First, we observe that forward-privacy does not necessarily guarantee security against adaptive file injection attacks if a scheme reveals other leakage patterns like the query equality. We then propose a notion of security called *correlation security* which generalizes forward privacy. We then show how correlation security can be used to formally define security against different kinds of injection attacks. We then propose the first injection-secure multi-map encryption encryption scheme and use it as a building block to design the first injection-secure searchable symmetric encryption (SSE) scheme; which solves one of the biggest open problems in the field. Towards achieving this, we also propose a new fully-dynamic volume-hiding multi-map encryption scheme which may be of independent interest.

*gamjad@google.com. Work done while at Brown University.
†seny@brown.edu.
‡tarik.moataz@mongodb.com. Work done while at Brown University and Aroki Systems.

Contents

1 Introduction

Structured encryption (STE) schemes encrypt data structures in such a way that they can be privately queried. Roughly speaking, STE schemes are secure if they leak nothing about the structure and queries beyond a well-specified and "reasonable" leakage function. Encrypted data structures are one of the main building blocks in the design of *sub-linear* algorithms on encrypted data, [1] for example, sub-linear encrypted search, graph and database algorithms [16, 15, 32]. Many aspects of STE have been studied and improved over the years including its expressiveness [12, 38, 24, 30, 28], its locality [14, 3, 20, 5, 18] and its leakage profiles [32, 31, 39]. A special case of STE is multi-map encryption which is a fundamental building block in the design of almost all sub-linear encrypted algorithms. Roughly speaking, encrypted multi-maps store label/tuple pairs and support get operations which, given a label, return the associated tuple.

Dynamism. An STE scheme is *static* if it supports queries over an encrypted data structure that never changes and it is *dynamic* if it supports queries over an encrypted structure that can be modified using, e.g., add, delete, or edit operations. Sub-linear dynamic EMMs were first achieved in [34] but in such a way that update operations could be correlated with search operations. In other words, while the adversary could not tell which label was being updated or queried, it could tell that a particular update was for a label that was queried in the past. This motivated Stefanov, Papamanthou and Shi to propose the notion of *forward-privacy* which, intuitively, guarantees that update operations cannot be linked to previous search operations [43]. Stefanov et al. also described the first forward-private EMM; achieving sub-linear query complexity with sub-linear client storage. A few years later, Bost proposed a formal definition of forward-privacy and the first forward-private EMM with optimal query complexity and client storage linear in the number of labels [8]. In [30], however, Kamara and Moataz pointed out that Bost's definition does not necessarily capture the notion of forward-privacy and suggested a different formalization.

Searchable symmetric encryption from EMMs. As mentioned above, EMMs are the main building block needed to achieve sub-linear and optimal searchable symmetric encryption (SSE) [16].[2] Given a document collection, the client builds a multi-map that indexes the document collection, i.e., its labels are keywords and the tuples are the identifiers of the documents that contain that keyword. It then encrypts the multi-map and the documents with a multi-map encryption scheme and a standard symmetric encryption scheme, respectively, and sends the EMM and encrypted documents to the server. To search for a keyword w, the client privately queries the EMM on w which reveals to the server the identifiers of the (encrypted) documents that need to be returned. EMMs that reveal the response to a query are called *response-revealing* and ones that do not are called response-hiding. [3]

Injection attacks. Injection attacks were introduced by Zhang, Katz and Papamanthou [44] and one of the main motivations to achieve forward-privacy (besides simply minimizing leakage) is that

[1] Another approach relies on property-preserving encryption (PPE) which also achieves sub-linear efficiency but with qualitatively different leakage profiles.

[2] Note that there exist several ways to build an SSE scheme from an EMM and each approach leads to a different efficiency vs. security tradeoff. For simplicity, we only describe the most natural approach to do so.

[3] Response-hiding EMMs can also be used but at the cost of an extra communication round.

it prevents *adaptive* injection attacks. In a standard/non-adaptive injection attack the adversary inserts files into the client's document collection and combines the EMM' s leakage with knowledge of its chosen files to recover the client's queries. More precisely, in the first phase of the attack, the server finds a way to insert documents with carefully-chosen subsets of the keyword space. [4] This results in the new documents being indexed and encrypted, and the EMM being updated to account for the new keywords. Later, when the client searches for a keyword w the server will learn the identifiers of the documents that contain w. If some of those documents are injected documents then it has learned some information about the client's query.

In an adaptive injection attack, the server injects files *after* the client makes its queries and uses the leakage of the EMM's update operation to correlate the update (and its contents) to previous queries. Forward privacy prevents such attacks because it guarantees that the updates caused by the adversary's adaptive file injections cannot be linked to previous queries.

The cost of injection attacks is measured in the number of documents that need to be injected and their size. Zhang et al. describe a non-adaptive attack that can recover all of a client's queries at the cost of injecting $\log \#\mathbb{W}$ files each of size $\#\mathbb{W}/2$, where $\mathbb{W}$ is the keyword space. Note that this attack—and others given in [44]—crucially rely on response identity leakage which can be hidden using, e.g., response-hiding EMMs or ORAM-based solutions. However, Blackstone, Kamara and Moataz [7] described file injection attacks that rely only on volume leakage which makes them applicable to almost all constructions including ORAM-based solutions.

Limitations of forward-privacy. While forward privacy is an important notion it has limitations. For conceptual clarity, we propose an alternative view on forward privacy and the security guarantees it provides against injection attacks. Instead of considering adaptive vs. non-adaptive injection attacks we will say that an injection attack is one where the adversary injects a file (and therefore causes an update) at any time during a sequence of client operations. We then ask whether the scheme reveals *correlations* between the update and the queries, where a correlation is leakage that reveals whether two operations are for the same label. Injection attacks essentially cause an update with an adversarially-known label and then use correlations between that update and adversarially-unknown queries to learn the unknown query labels.

The first limitation of forward privacy is that it only prevents correlations between updates and pre-injection queries. This was already pointed out in [44] as the authors explained that forward privacy only prevents adaptive injection attacks. To see why, consider the following sequence of operations on an EMM:

$$\mathbf{op} = (\mathsf{op}_1, \mathsf{op}_2, \mathsf{op}_3, \mathsf{op}_4) = ((\mathtt{qry}, \ell_5), (\mathtt{qry}, \ell_2), (\mathtt{app}, \ell_5, \mathbf{v}), (\mathtt{qry}, \ell_5)),$$

where $\mathsf{op}_i = (\mathtt{qry}, \ell_i)$ is a query operation on label ℓ_i and $\mathsf{op}_3 = (\mathtt{app}, \ell_5, \mathbf{v})$ is an adversarially-chosen append operation on ℓ_5 (i.e., append the tuple $\mathbf{v}$ to ℓ_5's pre-existing tuple) that results from a file injection. If the EMM is not forward-private, then its leakage on the append reveals that op_3 and op_1 are for the same label. If, on the other hand, the EMM is forward-private this correlation is not revealed when the op_3 occurs. Note, however, that the correlation between op_3 and op_4 could be revealed when op_4 occurs. Forward-privacy does not explicitly prevent this and, indeed, most forward-private constructions [43, 8, 9, 22, 1] leak this information and provide no guarantees for post-injection queries.

[4] This can be achieved in various ways depending on the application scenario.

The second limitation is that forward privacy only provides a relatively weak form of security in the sense that it only prevents *direct* correlations between updates and pre-injection queries but not *indirect* correlations which could occur if additional patterns are leaked. This means that forward privacy doesn't *necessarily* protect pre-injection queries. To see why, consider a setting where the sequence of operations above is executed with a scheme that is forward private but leaks the query equality. In this case, forward privacy guarantees that op_3 cannot be directly correlated with op_1. But an adversary can still learn op_1's label by observing that op_4 is correlated with op_3 (which is not prevented by forward privacy) and then using the query equality to learn that op_4 is correlated with op_1. The combination of these two correlations mean that op_1's label is ℓ_5.

1.1 Our Contributions

In this work, we focus on the security of dynamic structured encryption schemes. We make several contributions including new security definitions and constructions.

Correlation attacks. As illustrated by the discussion above, forward-privacy has several limitations including that it provides no guarantees for post-injection queries and that it does not necessarily protect pre-injection queries against injection attacks because it does not prevent indirect correlations. We also observe that the attacks considered in [44] capture only a fraction of how injections can be used to attack STE schemes. This motivates us to consider a broader class of attacks we call *correlation attacks* that work as follows. First, the adversary learns the labels/keywords associated to a subset of *operations*. Note that as opposed to injection attacks where the adversary chooses the label/keyword for an update operation, in a correlation attack the adversary could learn the label/keyword of any operation, e.g., queries, inserts, deletes etc. Furthermore, this can be achieved using injections but not necessarily; it could also be achieved using inference attacks or a known-data attack [27, 10, 7].[5] In the second phase, the adversary uses the leakage to correlate known operations to to unknown operations. Note that here we are concerned with correlations to *any* operation not just queries. In other words, the adversary's goal is not necessarily to learn the labels/keywords of unknown queries but could be to learn the labels/keywords of unknown updates, deletes etc.

Equality patterns and correlation graphs. Correlation attacks exploit leakage patterns that reveal correlations between operations; the most immediate examples of such patterns are *equality patterns* like the *query equality* which reveals if and when queries are for the same label/keyword, the *operation equality* which reveals if and when operations are for the same label/keyword or the *backward query equality* which reveals only if and when past queries are for the same label/keyword. Equality patterns can be defined in various ways but in this work we introduce a simple and useful representation we call *correlation graphs*. The correlation graph of a given equality pattern on a sequence of operations is a graph with operations as vertices and edges between equal operations. Correlation graphs can be composed to describe the correlations that result from leaking multiple equality patterns. Specifically, given two (or more) equality patterns patt_1 and patt_2 with correlation graphs $\mathcal{G}_1$ and $\mathcal{G}_2$, their union $\mathcal{G}_1 \cup \mathcal{G}_2$ captures all the correlations revealed by the two patterns.

[5] When a correlation attack is used after an inference or known-data attack it is effectively "boosting" that attack.

Correlation and injection security. We introduce and formalize the notion of *correlation security* which, intuitively, guarantees that a leakage profile does not reveal certain correlations between operations. In the context of correlation graphs we ask that no path exist between certain types of operations. We formalize this intuition using a game-based definition that guarantees that operations are indistinguishable given the correlation graph of the leakage on an adversarially-chosen sequence of operations. By constraining exactly how the adversary can choose the sequence, we can capture security against various kinds of correlation attacks. In particular, we show how to define security against injection attacks by which we mean protection of both pre- and post-injection queries. We also prove that if a scheme leaks *only* the query equality then it is injection-secure. While this might seem counter-intuitive given the example above, note that in the example pre-injection queries were correlated with updates by exploiting both the query equality between op_4 and op_1 and the correlation between op_4 and op_3 together.

An injection-secure EMM. We describe a dynamic multi-map encryption scheme called FIX that only leaks the query equality. What this means intuitively is that, given a sequence of queries and updates, the only thing leaked is the correlation between queries. Note that achieving this leakage profile is quite surprising considering the scheme is dynamic. In fact, most dynamic constructions—even forward-private ones—leak more; including correlations between queries and past updates.

At a very high level, the scheme works by handling update operations (appends and deletes) on a fixed but random schedule. This is accomplished by storing update operations in a stash at the client and "pushing" them to the remotely-stored EMM only according to the schedule. If a query occurs for a label whose updates have not been pushed yet, the information in the stash is combined with the "stale" results from the remote EMM to provide a correct answer. Because the schedule is fixed and independent of the update operations, the leakage is as well. We stress that this is a very high level description of our approach and that it does not capture many of the subtleties and challenges involved.

FIX is the first injection-secure multi-map; i.e., the first to protect both pre- and post-injection queries which solves an important problem left open since [44]. It has query complexity $O(\log \#\mathbb{L})$ and append and delete complexity $O(\log^2 \#\mathbb{L})$, where $\#\mathbb{L}$ is the size of the label space, under reasonable assumptions and parameterization which we detail in Section 4.

A dynamic volume-hiding EMM. Our construction makes black-box use of a static volume-hiding multi-map encryption scheme and of a dynamic volume-hiding multi-map encryption scheme. The former can be instantiated using many well-known constructions [31, 40]. To instantiate the latter, we design a new fully-dynamic volume-hiding multi-map encryption scheme we call DVLH (Section 5.1). DVLH is a dynamic variant of the static volume-hiding encrypted multi-map of Kamara and Moataz [31]. Similarly to their scheme, DVLH is lossy but we show that under natural assumptions (i.e., updates are Zipf-distributed) the lossiness can be bounded to be a reasonable amount with high probability (Section 5.2).[6]

Injection-secure SSE. An important application of EMMs is to the design of optimal SSE schemes. An SSE scheme encrypts a document collection in such a way that it can support keyword search; that is, given a keyword w, return the encrypted documents that include w. We show how

[6]Recent work by Amjad et al. [2] can also be used to instantiate the underlying dynamic volume-hiding EMM.

to use FIX to design the first injection-secure SSE scheme. Under the same assumptions and parameterization of FIX mentioned above, the SSE scheme has $O(\log \#\mathbb{L})$ query complexity and $O(\theta \cdot \log^2 \#\mathbb{L})$ add and delete complexity, where θ is a public parameter.

1.2 Related Work

Structured encryption was introduced by Chase and Kamara [15] as a generalization of index-based searchable symmetric encryption (SSE) [42, 16]. The most common and important type of STE schemes are multi-map encryption schemes which are a basic building block in the design of optimal SSE schemes [16, 34, 11], expressive SSE schemes [12, 29, 30, 38, 24] and encrypted databases [28, 13]. STE and encrypted multi-maps have been studied along several dimensions including dynamism [34, 33, 11, 38, 26] and I/O efficiency [11, 14, 4, 37, 20, 5, 19].

The notion of forward privacy was introduced by Stefanov, Papamanthou and Shi [43] and formally defined by Bost [8], who also proposed the first forward-private construction that does not leverage oblivious RAM techniques. Kamara and Moataz pointed out in [30] that the definition of [8] does not necessarily capture the intuitive security guarantee of forward-privacy and suggested that it be formalized as requiring that updates be leakage-free. Backward privacy was introduced by Bost, Minaud and Ohrimenko [9]. Several follow up works showed how to improve on the constructions of [9], sometimes achieving both forward and backward privacy [36, 23, 25, 1, 17, 41].

2 Preliminaries

Notation. The set of all binary strings of length n is denoted as $\{0,1\}^n$, and the set of all finite binary strings as $\{0,1\}^*$. $[n]$ is the set of integers $\{1,\ldots,n\}$, and $2^{[n]}$ is the corresponding power set. We write $x \leftarrow \chi$ to represent an element x being sampled from a distribution χ, and $x \xleftarrow{\$} X$ to represent an element x being sampled uniformly at random from a set X. The output x of an algorithm $\mathcal{A}$ is denoted by $x \leftarrow \mathcal{A}$. Given a sequence O of n elements, we refer to its ith element as O_i or $\mathsf{O}[i]$. If T is a set then $\#T$ refers to its cardinality. Given strings x and y, we refer to their concatenation as either $\langle x, y\rangle$ or $x\|y$.

Multi-maps. A static multi-map MM with capacity n is a collection of n label/tuple pairs $\{(\ell_i, \mathbf{v}_i)_i\}_{i\leq n}$ that supports Get operations. We denote the label space of a multi-map by $\mathbb{L}$ and the set of labels stored in a multi-map MM by $\mathbb{L}_{\mathsf{MM}}$. We write $\mathbf{v}_i = \mathsf{MM}[\ell_i]$ to denote getting the tuple associated with label ℓ_i. A multi-map is semi-dynamic if it also supports an insertion operation and it is fully-dynamic if it supports both insertions and deletions. Note that one can define various kinds of insertions and deletions. In this work we focus on additions, appends, erasures and deletions. An addition operation adds a label/tuple pair $(\ell, \mathbf{v})$ to the multi-map and is denoted as $\mathsf{MM}[\ell] := \mathbf{v}$. An append operation appends a tuple to the pre-existing tuple of a label. For example, if the label/tuple pair $(\ell, \mathbf{v})$ is already in the multi-map, then appending $\mathbf{v}'$ to ℓ results in ℓ being associated with the tuple $\mathbf{v}\|\mathbf{v}'$. We sometimes write this as $\mathsf{MM}[\ell]\|\mathbf{v}'$. An erase operation removes a set of values $\mathbf{v}'$ from the tuple $\mathbf{v}$ of a given label ℓ. We sometimes write this $\mathsf{MM}[\ell] - \mathbf{v}'$. A delete operation removes the entire label/tuple pair of a given label.

Document collections. A document collection is a set of documents $\mathsf{DC} = (D_1, \ldots, D_n)$, each document consisting of a set of keywords from some universe $\mathbb{W}$. We assume the universe of

keywords is totally ordered (e.g., using lexicographic order) and denote by $\mathbb{W}[i]$ the ith keyword in $\mathbb{W}$. We assume every document has an identifier that is independent of its contents and denote it $\mathsf{id}(D_i)$. We assume the existence of an efficient indexing algorithm that takes as input a data collection DC and outputs a multi-map that maps every keyword w in $\mathbb{W}$ to the identifiers of the documents that contain w. In previous work, this multi-map is referred to as an inverted index or as a database. For consistency, we refer to any multi-map derived in this way from a document collection as a database and denote it DC. Given a keyword w, we denote by $\mathsf{DC}(w) \subseteq \mathsf{DC}$, the set of all documents that contains w. We refer to the word-length of a document $|D|$ as its volume. We denote by $\mathsf{co}_{\mathsf{DC}}(w) \subseteq \mathbb{W}$ the set of keywords in $\mathbb{W}$ that co-occur with w; that is, the keywords that are contained in documents that contain w. When DC is clear from the context we omit DC and write only $\mathsf{co}(w)$.

Basic cryptographic primitives. A private-key encryption scheme is a set of three polynomial-time algorithms $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ such that Gen is a probabilistic algorithm that takes a security parameter k and returns a secret key K; Enc is a probabilistic algorithm takes a key K and a message m and returns a ciphertext c; Dec is a deterministic algorithm that takes a key K and a ciphertext c and returns m if K was the key under which c was produced. Informally, a private-key encryption scheme is secure against chosen-plaintext attacks (CPA) if the ciphertexts it outputs do not reveal any partial information about the plaintext even to an adversary that can adaptively query an encryption oracle. We say a scheme is random-ciphertext-secure against chosen-plaintext attacks (RCPA) if the ciphertexts it outputs are computationally indistinguishable from random even to an adversary that can adaptively query an encryption oracle.[7] In addition to encryption schemes, we also make use of pseudo-random functions (PRF) and permutations (PRP), which are polynomial-time computable functions that cannot be distinguished from random functions by any probabilistic polynomial-time adversary. We refer the reader to [35] for notation and security definitions for these objects.

2.1 Structured Encryption

Structured encryption (STE) schemes encrypt data structures in such a way that they can be privately queried. The first definitions of structured encryption were presented by Chase and Kamara [15]. STE schemes can be interactive or non-interactive. Interactive schemes produce encrypted structures that are queried or updated through an interactive two-party protocol between a client and a server, whereas non-interactive schemes produce structures that can be queried or updated by sending a single token. These schemes can also be response-hiding or response-revealing where the former reveal the response to queries to the server whereas the latter do not. We recall here the syntax of an interactive response-hiding dynamic structured encryption scheme.

Definition 2.1 (Interactive response-hiding dynamic structured encryption)**.** *An interactive response-hiding dynamic structured encryption scheme* $\Sigma_{\mathsf{DS}} = (\mathsf{Setup}, \mathsf{Query}, \mathsf{Insert}, \mathsf{Delete}, \mathsf{Res})$ *for data type* DS *consists of the following polynomial time algorithms:*

1. $(K, st) \leftarrow \mathsf{Setup}_{\mathbf{C}}(1^k, \mathsf{DS})$ *is an algorithm that takes as input the security parameter* k *and a data structure* DS *and outputs a secret key* K *and an (optional) state* st*.*

[7] RCPA-secure encryption can be instantiated practically using either the standard PRF-based private-key encryption scheme or, e.g., AES in counter mode.

2. $((r, st'), \mathsf{EDS}') \leftarrow \mathsf{Query}_{\mathbf{C},\mathbf{S}}((K, q, st), \mathsf{EDS})$ *is an interactive protocol executed between a client* $\mathbf{C}$ *and server* $\mathbf{S}$*.* $\mathbf{C}$ *inputs the secret key* K*, a query* q *and state* st*.* $\mathbf{S}$ *inputs the encrypted data structure* EDS*. The protocol outputs a response* r *and an updated state* st' *to the client and an updated encrypted structure* EDS' *to the server.*

3. $(st', \mathsf{EDS}') \leftarrow \mathsf{Insert}_{\mathbf{C},\mathbf{S}}((K, st, a), \mathsf{EDS})$ *is an interactive protocol executed between a client* $\mathbf{C}$ *and server* $\mathbf{S}$*. The client inputs a secret key* K*, a state* st *and an add operation* a*.The server inputs an encrypted structure* EDS*. The protocol outputs an updated state* st' *to the client and an updated encrypted structure* EDS' *to the server.*

4. $(st', \mathsf{EDS}') \leftarrow \mathsf{Delete}_{\mathbf{C},\mathbf{S}}((K, st, d), \mathsf{EDS})$ *is an interactive protocol executed between a client* $\mathbf{C}$ *and server* $\mathbf{S}$*. The client inputs a secret key* K*, a state* st *and a delete operation* d*. The server inputs an encrypted structure* EDS*. The protocol outputs an updated state* st' *to the client and an updated encrypted structure* EDS' *to the server.*

5. $r \leftarrow \mathsf{Res}_{\mathbf{C}}(K, \mathrm{ct})$ *is a deterministic algorithm that takes as input a secret key* K *and an encrypted query result* ct*. It outputs a response* r*.*

The syntax of a response-revealing scheme can be recovered by having Query output the response directly and omitting the Res algorithm.

Adaptive security. The standard notion of security for STE guarantees that: (1) an encrypted structure reveals no information about its underlying structure beyond the setup leakage $\mathcal{L}_{\mathsf{S}}$; and (2) the various operations that are supported (e.g., query, add, delete) reveal no information about the structure and the operations beyond some stateful operation leakage $\mathcal{L}_{\mathsf{O}}$. If this holds for non-adaptively chosen operations then the scheme is said to be non-adaptively secure. If, on the other hand, the operations can be chosen adaptively, the scheme is said to be adaptively-secure [16, 15]. Note that the operation leakage is usually broken down into separate leakage functions—one for each supported operation—but here we consider a single leakage function $\mathcal{L}_{\mathsf{O}}$ for all operations. The advantage of this formulation is that it allows us to more easily capture leakage that is a function of different operations.

Definition 2.2 (Adaptive Security of dynamic interactive STE)**.** *Let* $\Sigma = (\mathsf{Setup}, \mathsf{Query}, \mathsf{Insert}, \mathsf{Delete}, \mathsf{Res})$ *be an interactive dynamic STE scheme and consider the following probabilistic experiments where* $\mathcal{A}$ *is a stateful adversary,* $\mathcal{S}$ *is a stateful simulator,* $\Lambda = (\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{O}})$ *is a leakage profile and* $z \in \{0,1\}^*$*:*

$\mathbf{Real}_{\Sigma,\mathcal{A}}(k)$: *given* z*,* $\mathcal{A}$ *chooses a data structure* DS *and receives an encrypted structure* EDS *from the challenger, where* $(K, st; \mathsf{EDS}) \leftarrow \mathsf{Setup}(1^k, \mathsf{DS})$*.* $\mathcal{A}$ *then adaptively chooses a polynomial number of operations* $\mathsf{op}_1, \ldots, \mathsf{op}_m$ *and, for each one, the adversary and challenger execute the appropriate protocol with* $\mathcal{A}$ *playing the role of the server and the challenger playing the role of the client. Finally, the adversary outputs a bit* $b \in \{0,1\}$*.*

$\mathbf{Ideal}_{\Sigma,\mathcal{A},\mathbb{S}}(k)$: *given* z*,* $\mathcal{A}$ *chooses a data structure* DS*. Given* z *and* $\mathcal{L}_{\mathsf{S}}(\mathsf{DS})$ *the simulator* $\mathcal{S}$ *sends an encrypted data structure* EDS *to* $\mathcal{A}$*. The adversary adaptively picks a polynomial number of operations* $\mathsf{op}_1, \ldots, \mathsf{op}_m$ *and, for each one,* $\mathcal{A}$ *and* $\mathcal{S}$ *execute the appropriate protocol with* $\mathcal{A}$ *playing the role of the server and* $\mathcal{S}$ *playing the role of the client. During these executions, the simulator* $\mathcal{S}$ *only receives* $\mathcal{L}_{\mathsf{O}}(\mathsf{DS}, \mathsf{op}_i)$*. Finally, the adversary outputs a bit* $b \in \{0,1\}$*.*

We say that Σ is adaptively $(\mathcal{L}_\mathsf{S}, \mathcal{L}_\mathsf{O})$-secure if there exists a PPT simulator $\mathcal{S}$ such that for all PPT adversaries $\mathcal{A}$, for all $z \in \{0,1\}^*$:

$$|\Pr[\mathbf{Real}_{\Sigma,\mathcal{A}}(k) = 1] - \Pr[\mathbf{Ideal}_{\Sigma,\mathcal{A},\mathbb{S}}(k) = 1]| \leq \mathsf{negl}(k).$$

Modeling leakage. The leakage of an STE scheme Σ is characterized by a leakage profile $\Lambda_\Sigma = (\mathcal{L}_\mathsf{S}, \mathcal{L}_\mathsf{O})$ composed of a setup leakage $\mathcal{L}_\mathsf{S}$ and an operation leakage $\mathcal{L}_\mathsf{O}$. Each of these leakage functions can themselves be functions of various leakage patterns. In this work, all leakage functions and leakage patterns are *stateful*. We recall some leakage patterns that will appear throughout this work:

- the *query equality* qeq takes as input a data structure and a query and reveals if and when the query was repeated
- the *operation equality* oeq takes as input a data structure and an operation and reveals if and when the query associated with the operation appeared in the past.
- the *response length* rlen takes as input a data structure and an query and reveals the length of the query's response.

3 Defining Correlation Security

In this work we focus on security against correlation attacks which are a generalization of injection attacks. We describe our framework in the context of EMMs for concreteness but note that it can be applied to any encrypted data structure. A correlation attack is a query-recovery attack that works in two phases. First, the adversary learns the labels of a subset of operations, e.g., by injecting files or executing an inference attack. Then, it uses leakage to link unknown operations to known operations. We describe a framework that formally captures correlations revealed by various leakage patterns and allows us to formalize security against correlation attacks.

Equality patterns & correlation graphs. An equality pattern reveals if and when two values are the same. Examples include the query equality pattern which reveals if and when two queries are the same and the operation equality pattern which reveals if and when two operations are for the same label. Equality patterns can be defined in different ways (e.g., as binary vectors or binary matrices) but, effectively, they can all be represented as graphs with operations as vertices and edges between two operations if they are for the same label. We call such graphs *correlation graphs*. For example, the query equality and the operation equality of the sequence

$$\mathbf{op} = (\mathsf{op}_1, \mathsf{op}_2, \mathsf{op}_3, \mathsf{op}_4) = ((\mathtt{qry}, \ell_5), (\mathtt{qry}, \ell_2), (\mathtt{app}, \ell_5, \mathbf{v}), (\mathtt{qry}, \ell_5))$$

can be represented with the graphs $\mathcal{G}_\mathsf{qeq}$ and $\mathcal{G}_\mathsf{oeq}$ illustrated below:

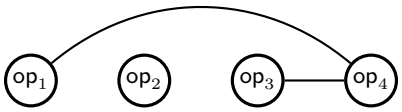
Leakage patterns reveal information incrementally, per operation, so it will be useful for us to have notation to describe this. Given a leakage profile $\Lambda = (\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{O}})$ and a sequence of operation $(\mathsf{op}_1, \ldots, \mathsf{op}_i)$ we write $\mathcal{G}_i = \mathcal{G}_{i-1} + \mathcal{L}_{\mathsf{O}}(\mathsf{DS}, \mathsf{op}_i)$ to refer to the correlation graph that results from adding $\mathcal{L}_{\mathsf{O}}(\mathsf{DS}, \mathsf{op}_i)$ to $\mathcal{G}_{i-1}$.

Composition of equality patterns. Encrypted search schemes often reveal more than a single pattern so when evaluating their security one needs to consider the composition of the patterns. For equality patterns this can be done by taking the union of their correlation graphs. For example, consider a scheme with leakage profile

$$\Lambda = (\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{O}}) = (\star, (\mathsf{qeq}, \mathsf{patt}))$$

where patt reveals if a query was on the same label of a previous append. The correlation graphs of qeq and patt on the sequence $\mathbf{op}$ above are $\mathcal{G}_{\mathsf{qeq}}$ and $\mathcal{G}_{\mathsf{patt}}$ illustrated below,

and the correlation graph of $\mathcal{L}_{\mathsf{O}}$ is



Correlation security. Intuitively, our notion of correlation security guarantees that a leakage profile hides certain correlations between operations. We formalize this using a game-based definition that guarantees that an adversary cannot distinguish between operations even when given the correlation graph of adversarially-chosen operations. [8] By setting constraints on how exactly the adversary is allowed to choose its operations, one can define security against specific correlation attacks. More formally, let $\mathcal{L}_{\mathsf{O}}$ be an operation leakage and let π_1, π_2, π_3 be predicates over sequences of operations. Consider the following probabilistic experiment between a stateful adversary $\mathcal{A}$ and a challenger:

$\mathbf{Corr}^{\pi_1,\pi_2,\pi_3}_{\mathcal{L}_{\mathsf{O}},\mathcal{A}}(k)$:

1. $\mathcal{A}$ chooses a data structure DS and receives $\perp$ from the challenger $\mathcal{C}$;
2. Let $\mathcal{G}_0$ be an empty graph;
3. $\mathcal{A}$ adaptively chooses polynomially-many operations $\mathbf{op} = (\mathsf{op}_1, \ldots, \mathsf{op}_m)$ as follows. For all $1 \leq i \leq m$,
   (a) $\mathcal{A}$ chooses and sends an operation op_i such that $\pi_1(\mathsf{op}_1, \ldots, \mathsf{op}_i) = 1$ to the challenger;
   (b) the challenger returns $\mathcal{G}_i = \mathcal{G}_{i-1} + \mathcal{L}_{\mathsf{O}}(\mathsf{DS}, \mathsf{op}_i)$ to $\mathcal{A}$;

[8] Note that correlation-security is a security notion that is defined for leakage profiles not for STE schemes.

4. $\mathcal{A}$ chooses two sequences of operations $\mathbf{op}_0^\star$ and $\mathbf{op}_1^\star$ of polynomial length λ such that $\pi_2(\mathbf{op}, \mathbf{op}_0^\star, \mathbf{op}_1^\star) = 1$ and sends them to the challenger;
5. the challenger samples $b \xleftarrow{\$} \{0,1\}$;
6. for all $1 \leq i \leq \lambda$, let $\mathcal{G}_{m+i} = \mathcal{G}_{m+i-1} + \mathcal{L}_{\mathsf{O}}(\mathsf{DS}, \mathsf{op}_{b,i}^\star)$ to $\mathcal{A}$;
7. $\mathcal{A}$ adaptively chooses polynomially-many operations $\mathbf{op}' = (\mathsf{op}'_1, \ldots, \mathsf{op}'_{m'})$ operations as follows. For all $1 \leq i \leq m'$,
   (a) $\mathcal{A}$ chooses and sends an operation op'_i such that $\pi_3(\mathbf{op}, \mathbf{op}_0^\star, \mathbf{op}_1^\star, \mathsf{op}'_1, \ldots, \mathsf{op}'_i) = 1$ to the challenger;
   (b) the challenger returns $\mathcal{G}_{m+\lambda+i} = \mathcal{G}_{m+\lambda+i-1} + \mathcal{L}_{\mathsf{O}}(\mathsf{DS}, \mathsf{op}'_i)$ to $\mathcal{A}$;
8. $\mathcal{A}$ outputs a bit b';
9. The experiment outputs 1 if $b' = b$ and 0 otherwise.

Definition 3.1 (Correlation security)**.** *We say that a leakage profile $\Lambda = (\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{O}})$ is (π_1, π_2, π_3)-correlation secure if for all* PPT *adversaries $\mathcal{A}$,*

$$\Pr\left[\mathbf{Corr}^{\pi_1,\pi_2,\pi_3}_{\mathcal{L}_{\mathsf{O}},\mathcal{A}}(k) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(k).$$

Injection security. As discussed in Section 1, injection attacks work by causing updates with known labels and then using leakage to correlate the updates to queries with unknown labels. And while forward privacy prevents direct collections between updates and pre-injection queries it provides no guarantees for post-injection queries and does not prevent indirect correlations between updates and pre-injection queries. In fact, all practical forward-private constructions [43, 8, 9, 22, 1] have these limitations. To address these limitations, a leakage profile must prevent correlations between updates and all queries not just pre-injection queries. This can be captured using our definitional framework by imposing the following conditions on the adversary's choice of operations: in step 3 the adversary is free to choose any operation; in step 4 the challenge operations have to be queries and must be different than any query chosen in step 3; and in step 7 the adversary can choose any update but only queries that are different than the challenge queries. In the definition below, we formalize this intuition.

Definition 3.2 (Security against injection attacks)**.** *Let $\mathsf{inj} = (\pi_1, \pi_2, \pi_3)$, where*

- $\pi_1(\mathbf{op}) = 1$ *for all poly-size sequences of operations* $\mathbf{op}$*;*
- $\pi_2(\mathbf{op}, \mathbf{op}_0^\star, \mathbf{op}_1^\star)$ *outputs* 1 *if* $\mathbf{op}_0^\star$ *and* $\mathbf{op}_1^\star$ *are query-only sequences with the same query equality leakage and if none of the queries in* $\mathbf{op}_0^\star$ *and* $\mathbf{op}_1^\star$ *are in* $\mathbf{op}$*; otherwise it outputs* 0*;*
- $\pi_3(\mathbf{op}, \mathbf{op}_0^\star, \mathbf{op}_1^\star, \mathbf{op}')$ *outputs* 1 *if none of the queries in* $\mathbf{op}_0^\star$ *and* $\mathbf{op}_1^\star$ *are in* $\mathbf{op}'$*; otherwise it outputs* 0*.*

We say that a leakage profile $\Lambda = (\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{O}})$ is secure against file injections if for all PPT *adversaries $\mathcal{A}$,*

$$\Pr\left[\mathbf{Corr}^{\mathsf{inj}}_{\mathcal{L}_{\mathsf{O}},\mathcal{A}}(k) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(k).$$

Notice that Definition 3.2 allows a leakage profile to reveal correlations between queries but not between updates and queries. This is because the definition: (1) allows the challenge queries to have the same labels as previous and future updates; but (2) prohibits the challenge queries from having the same label as previous or future queries; and (3) prohibits the sequences of challenge queries from having different query equality. In the following Theorem, we show that this is enough to guarantee security against injection attacks in the sense that correlations to both pre- and post-injection queries are prevented.

Theorem 3.3. *The leakage profile* $\Lambda = (\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{O}}) = (\star, \mathsf{qeq})$ *is injection-secure.*

Proof. Consider the correlation graph $\mathcal{G}$ produced in a $\mathbf{Corr}^{\mathsf{inj}}_{\mathcal{L}_{\mathsf{O}},\mathcal{A}}$ experiment. By definition of (π_1, π_2, π_3), the queries in the challenge sequence $\mathbf{op}^{\star}_b = (q_{b,1}, \ldots, q_{b,\lambda})$ are not connected to any other query nodes. Furthermore, since there is no add or delete leakage by the definition of Λ, there are no edges incident to the add and delete nodes. It follows then that there is no path in $\mathcal{G}$ from any adversarially-chosen operation to the challenge queries. Therefore, the best the adversary can do is to guess b, from which the Theorem follows.

■

4 An Injection-Secure Encrypted Multi-Map

Our construction $\mathsf{FIX} = (\mathsf{Setup}, \mathsf{Get}, \mathsf{Append}, \mathsf{Erase}, \mathsf{Res})$ makes black-box use of a static volume-hiding multi-map encryption scheme $\Sigma = (\mathsf{Setup}, \mathsf{Get}, \mathsf{Res})$ and a dynamic volume-hiding multi-map encryption scheme $\Delta = (\mathsf{Setup}, \mathsf{Get}, \mathsf{Insert}, \mathsf{Delete}, \mathsf{Res})$. The details of the scheme are in Figures (1) and (2). At a high-level, it works as follows.

Setup. The setup algorithm takes as input a security parameter k and a multi-map MM. It first picks a permutation π from $\{0, \cdots, \#\mathbb{L}_{\mathsf{MM}} - 1\}$ to $\mathbb{L}_{\mathsf{MM}}$ uniformly at random. It then instantiates two multi-maps: (1) a new multi-map MM_n with label space $\mathbb{L}_{\mathsf{MM}}$ and sets, for all $\ell \in \mathbb{L}_{\mathsf{MM}}$, $\mathsf{MM}_n[\ell] := \perp$ (an empty tuple); and (2) a multi-map $\mathsf{MM}_{\mathsf{stash}}$. It also initializes a counter count to 0. The multi-map $\mathsf{MM}_{\mathsf{stash}}$ will be used as a queue to purposely delay updates, while the counter will be used to determine which labels to update at a given time. The setup algorithm then encrypts MM by executing $(K_o, st_o, \mathsf{EMM}_o) \leftarrow \Sigma.\mathsf{Setup}(1^k, \mathsf{MM})$ and MM_n by computing $(K_n, st_n, \mathsf{EMM}_n) \leftarrow \Delta.\mathsf{Setup}(1^k, \mathsf{MM}_n)$. The old encrypted multi-map EMM_o will be queried but never updated whereas the new encrypted multi-map EMM_n will be both queried and updated. Finally, the algorithm outputs a key $K = (K_o, K_n)$, a state $st = (st_o, st_n, \mathsf{MM}_{\mathsf{stash}}, \pi, \mathsf{count})$, and an encrypted multi-map $\mathsf{EMM} = (\mathsf{EMM}_o, \mathsf{EMM}_n)$.

Append & erase. The append and erase operations of FIX are based on the same core operation we call Push. Given an append operation $\mathsf{op} = (\mathtt{app}, \ell, \mathbf{v})$, the client updates its stash by appending $v \| \mathtt{add}$ to $\mathsf{MM}_{\mathsf{stash}}[\ell]$, for all $v \in \mathbf{v}$. It then "pushes" the changes as explained below. We sometimes refer to $\mathbf{v}$ as ℓ's update tuple.

Given an erase operation $\mathsf{op} = (\mathtt{del}, \ell, \mathbf{v})$ the client does the following. For all values $v \in \mathbf{v}$, if $v \| \mathtt{add}$ already exists in $\mathsf{MM}_{\mathsf{stash}}[\ell]$, it removes it from the stash. Otherwise it adds $v \| \mathtt{del}$ to $\mathsf{MM}_{\mathsf{stash}}[\ell]$. It then "pushes" the changes as described below.

To push its updates, the client retrieves and updates μ label/tuple pairs chosen using the random permutation π. More precisely, for all $0 \leq i \leq \mu - 1$, the client queries EMM_o and EMM_n on $\ell_{\pi(j)}$, where $j = \mathsf{count} + i$, resulting in responses $\mathbf{r}_o$ and $\mathbf{r}_n$, respectively. It then uses $\mathbf{r}_o$, $\mathbf{r}_n$ and its stash to generate a new tuple $\mathbf{v}_n$. $\mathbf{v}_n$ will contain the contents of $\mathbf{r}_n$ and the values in $\mathsf{MM}_{\mathsf{stash}}[\ell_{\pi(j)}]$ of the form $v \| \mathtt{add}$. If any value v in $\mathbf{r}_n$ is also in $\mathsf{MM}_{\mathsf{stash}}[\ell_{\pi(j)}]$ in the form $v \| \mathtt{del}$, it is removed from $\mathbf{r}_n$. Because EMM_o is a static structure, all the values from $\mathbf{r}_o$ that need to be erased are added to $\mathbf{v}_n$ using a *lazy delete* approach. Specifically, for every value v in $\mathbf{r}_o$ that is in $\mathsf{MM}[\ell_{\pi(j)}]$ in the form $v \| \mathtt{del}$, the client adds $v \| \mathtt{del} \| o$ to $\mathbf{v}_n$. The client then uses Δ's delete operation to remove the label/tuple pair for $\ell_{\pi(j)}$ from EMM_n, and inserts the new pair $(\ell_{\pi(j)}, \mathbf{v}_n)$. It also removes $\ell_{\pi(j)}$ from the stash. Finally, it increments the counter by μ and outputs the updated state while the server outputs the updated EMM_n.

Get. This is a two-party protocol between the client and the server. Given a label ℓ, the client queries both EMM_o and EMM_n on ℓ using $\Sigma.\mathsf{Get}$ and $\Delta.\mathsf{Get}$, respectively. It then computes the union of the results, and removes all the values in $\mathsf{MM}_{\mathsf{stash}}[\ell]$ tagged with $\mathtt{del}$. It also removes all the values in the result set from EMM_o, if they are tagged with $\mathtt{del} \| o$ in the result set from EMM_n. The client then outputs the final result along with the updated state, while the server outputs nothing.

Efficiency. The communication complexity of $\mathsf{FIX.Get}$ is

$$O(\mathsf{comm}^{\Sigma}_{\mathsf{Get}}(\ell) + \mathsf{comm}^{\Delta}_{\mathsf{Get}}(\ell))$$

where $\mathsf{comm}^{\Sigma}_{\mathsf{Get}}, \mathsf{comm}^{\Delta}_{\mathsf{Get}}$ are the communication complexities of $\Sigma.\mathsf{Get}$ and $\Delta.\mathsf{Get}$ respectively. The query complexity of FIX is

$$O(\mathsf{time}^{\Sigma}_{\mathsf{Get}}(\ell) + \mathsf{time}^{\Delta}_{\mathsf{Get}}(\ell) + \mathsf{dels}_o(\ell) + \#\mathsf{MM}_{\mathsf{stash}}[\ell])$$

where $\mathsf{time}^{\Sigma}_{\mathsf{Get}}, \mathsf{time}^{\Delta}_{\mathsf{Get}}$ are the computational complexities of $\Sigma.\mathsf{Get}$ and $\Delta.\mathsf{Get}$ respectively, $\mathsf{dels}_o(\ell)$ is the number of erasure operations issued for values of ℓ in Σ, and $\#\mathsf{MM}_{\mathsf{stash}}[\ell]$ is equal to the number of values in the last $\#\mathbb{L}_{\mathsf{MM}}/\mu - 1$ updates associated with ℓ.

Storage complexity is the sum of the storage complexities induced by Σ and Δ. The communication complexities of $\mathsf{FIX.Append}$ and $\mathsf{FIX.Erase}$ are

$$O(\mu \cdot (\mathsf{comm}^{\mathsf{FIX}}_{\mathsf{Get}}(\ell) + \mathsf{comm}^{\Delta}_{\mathsf{Delete}}(\ell) + \mathsf{comm}^{\Delta}_{\mathsf{Insert}}(\ell)))$$

where $\mathsf{comm}^{\mathsf{FIX}}_{\mathsf{Get}}$, $\mathsf{comm}^{\Delta}_{\mathsf{Insert}}$ and $\mathsf{comm}^{\Delta}_{\mathsf{Delete}}$ are the communication complexities of $\mathsf{FIX.Get}$, $\Delta.\mathsf{Insert}$ and $\Delta.\mathsf{Delete}$, respectively.

The update complexity of FIX is

$$O(\#\mathbf{v} + \mu \cdot (\mathsf{time}^{\mathsf{FIX}}_{\mathsf{Get}}(\ell) + \mathsf{time}^{\Delta}_{\mathsf{Delete}}(\ell) + \mathsf{time}^{\Delta}_{\mathsf{Insert}}(\ell)))$$

where $\mathsf{time}^{\mathsf{FIX}}_{\mathsf{Get}}$ is the get complexity of FIX and $\mathsf{time}^{\Delta}_{\mathsf{Delete}}(\ell)$, $\mathsf{time}^{\Delta}_{\mathsf{Insert}}(\ell)$ are the computational complexities of $\Delta.\mathsf{Delete}$, $\Delta.\mathsf{Insert}$ respectively.

The size of the client state is analyzed in section 4.1. Note that it is independent of the underlying schemes Δ and Σ.

Let $\Sigma = (\mathsf{Setup}, \mathsf{Get}_{\mathbf{C},\mathbf{S}})$ be a static response-hiding multi-map encryption scheme, $\Delta = (\mathsf{Setup}, \mathsf{Get}_{\mathbf{C},\mathbf{S}}, \mathsf{Insert}_{\mathbf{C},\mathbf{S}}, \mathsf{Delete}_{\mathbf{C},\mathbf{S}})$ be a dynamic response-hiding multi-map encryption scheme. Consider the scheme $\mathsf{FIX} = (\mathsf{Setup}, \mathsf{Get}_{\mathbf{C},\mathbf{S}}, \mathsf{Append}_{\mathbf{C},\mathbf{S}}, \mathsf{Erase}_{\mathbf{C},\mathbf{S}})$ defined as follows:

- $\mathsf{Setup}\big(1^k, \mathsf{MM}\big)$:
  1. sample a permutation $\pi \xleftarrow{\$} \{\{0, \cdots, \#\mathbb{L}_{\mathsf{MM}} - 1\} \to \mathbb{L}_{\mathsf{MM}}\}$;
  2. initialize a multi-map MM_n with label space $\mathbb{L}_{\mathsf{MM}}$, an empty multi-map $\mathsf{MM}_{\mathsf{stash}}$, and a counter count initialized to 0;
  3. compute
  $$(K_o, st_o, \mathsf{EMM}_o) \leftarrow \Sigma.\mathsf{Setup}(1^k, \mathsf{MM});$$
  4. compute
  $$(K_n, st_n, \mathsf{EMM}_n) \leftarrow \Delta.\mathsf{Setup}(1^k, \mathsf{MM}_n);$$
  5. output (K, st, EMM) where $K := (K_o, K_n)$, $st := (st_o, st_n, \mathsf{MM}_{\mathsf{stash}}, \pi, \mathsf{count})$ and $\mathsf{EMM} := (\mathsf{EMM}_o, \mathsf{EMM}_n)$.
- $\mathsf{Get}_{\mathbf{C},\mathbf{S}}\big((K, st, \ell), \mathsf{EMM}\big)$:
  1. $\mathbf{C}$ parses K as (K_o, K_n), st as $(st_o, st_n, \mathsf{MM}_{\mathsf{stash}}, \pi, \mathsf{count})$;
  2. $\mathbf{S}$ parses EMM as $(\mathsf{EMM}_o, \mathsf{EMM}_n)$;
  3. $\mathbf{C}$ initializes an empty set $\mathbf{v}$;
  4. $\mathbf{C}$ and $\mathbf{S}$ execute
  $$(\mathbf{r}_o, \perp) \leftarrow \Sigma.\mathsf{Get}_{\mathbf{C},\mathbf{S}}\big((K_o, st_o, \ell), \mathsf{EMM}_o\big);$$
  and,
  $$(\mathbf{r}_n, \perp) \leftarrow \Delta.\mathsf{Get}_{\mathbf{C},\mathbf{S}}\big((K_n, st_n, \ell), \mathsf{EMM}_n\big);$$
  5. $\mathbf{C}$ computes $\mathbf{r}^+_{\mathsf{local}}$ and $\mathbf{r}^-_{\mathsf{local}}$ from $\mathsf{MM}_{\mathsf{stash}}[\ell]$ such that
  $$\mathbf{r}^+_{\mathsf{local}} = \{v : v \| \mathtt{add} \in \mathsf{MM}_{\mathsf{stash}}[\ell]\} \quad \text{and} \quad \mathbf{r}^-_{\mathsf{local}} = \{v : v \| \mathtt{del} \in \mathsf{MM}_{\mathsf{stash}}[\ell]\};$$
  6. $\mathbf{C}$ sets $\mathbf{r}^-_o := \{v : v \| \mathtt{del} \| \mathtt{old} \in \mathbf{r}_n\}$;
  7. $\mathbf{C}$ sets $\mathbf{v} := \big(\mathbf{r}_o \cup \mathbf{r}_n \cup \mathbf{r}^+_{\mathsf{local}}\big) \setminus \big(\mathbf{r}^-_{\mathsf{local}} \cup \mathbf{r}^-_o\big)$;
  8. $\mathbf{C}$ outputs $\mathbf{v}$, while $\mathbf{S}$ outputs $\perp$.

Figure 1: FIX (Part 1).

- $\mathsf{Append}_{\mathbf{C},\mathbf{S}}\big((K,\mu,st,(\ell,\mathbf{v})),\mathsf{EMM}\big)$:
  1. for all $v \in \mathbf{v}$, $\mathbf{C}$ sets $\mathsf{MM}_{\mathsf{stash}}[\ell] := \mathsf{MM}_{\mathsf{stash}}[\ell] \cup \{v \| \mathtt{add}\}$;
  2. $\mathbf{C}$ and $\mathbf{S}$ execute $\mathsf{Push}_{\mathbf{C},\mathbf{S}}\big((K,\mu,st),\mathsf{EMM}\big)$;
- $\mathsf{Erase}_{\mathbf{C},\mathbf{S}}\big((K,\mu,st,(\ell,\mathbf{v})),\mathsf{EMM}\big)$:
  1. for all $v \in \mathbf{v}$,
     (a) if $v\|\mathtt{add} \notin \mathsf{MM}_{\mathsf{stash}}[\ell]$, $\mathbf{C}$ sets $\mathsf{MM}_{\mathsf{stash}}[\ell] := \mathsf{MM}_{\mathsf{stash}}[\ell] \cup \{v\|\mathtt{del}\}$
     (b) else it sets $\mathsf{MM}_{\mathsf{stash}}[\ell] := \mathsf{MM}_{\mathsf{stash}}[\ell] \setminus \{v\|\mathtt{add}\}$
  2. $\mathbf{C}$ and $\mathbf{S}$ execute $\mathsf{Push}_{\mathbf{C},\mathbf{S}}\big((K,\mu,st),\mathsf{EMM}\big)$;
- $\mathsf{Push}_{\mathbf{C},\mathbf{S}}\big((K,\mu,st),\mathsf{EMM}\big)$:
  1. $\mathbf{C}$ parses K as (K_o,K_n) and st as $(st_o,st_n,\mathsf{MM}_{\mathsf{stash}},\pi,\mathsf{count})$
  2. $\mathbf{S}$ parses EMM as $(\mathsf{EMM}_o,\mathsf{EMM}_n)$;
  3. for all $0 \le i \le \mu-1$,
     (a) set $j := \mathsf{count} + i \mod \#\mathbb{L}_{\mathsf{MM}}$;
     (b) $\mathbf{C}$ and $\mathbf{S}$ execute
     $$(\mathbf{r}_o,\perp) \leftarrow \Sigma.\mathsf{Get}_{\mathbf{C},\mathbf{S}}\big((K_o,st_o,\ell_\pi(j)),\mathsf{EMM}_o\big);$$
     and
     $$(\mathbf{r}_n,\perp) \leftarrow \Delta.\mathsf{Get}_{\mathbf{C},\mathbf{S}}\big((K_n,st_n,\ell_\pi(j)),\mathsf{EMM}_n\big);$$
     (c) for all $v\|\mathtt{x} \in \mathsf{MM}_{\mathsf{stash}}[\pi(j)]$,
        i. if $\mathtt{x} = \mathtt{add}$, $\mathbf{C}$ sets $\mathbf{r}^+_{\mathsf{local}} := \mathbf{r}^+_{\mathsf{local}} \cup \{v\}$;
        ii. if $\mathtt{x} = \mathtt{del}$,
           A. if $v \notin \mathbf{r}_o$, $\mathbf{C}$ sets $\mathbf{r}^-_{\mathsf{local}} := \mathbf{r}^-_{\mathsf{local}} \cup \{v\}$
           B. else $\mathbf{C}$ sets $\mathbf{r}^+_{\mathsf{local}} := \mathbf{r}^+_{\mathsf{local}} \cup \{v\|\mathtt{del}\|\mathtt{old}\}$;
     (d) $\mathbf{C}$ removes $\ell_{\pi(j)}$'s tuple from $\mathsf{MM}_{\mathsf{stash}}[\ell_{\pi(j)}]$;
     (e) $\mathbf{C}$ computes $\mathbf{v}_n = (\mathbf{r}_n \cup \mathbf{r}^+_{\mathsf{local}}) \setminus \mathbf{r}^-_{\mathsf{local}}$;
     (f) $\mathbf{C}$ and $\mathbf{S}$ execute
     $$(\mathbf{r},\perp) \leftarrow \Delta.\mathsf{Delete}_{\mathbf{C},\mathbf{S}}\Big(\Big(K_n,st_n,\ell_{\pi(j)}\Big),\mathsf{EMM}_n\Big);$$
     (g) $\mathbf{C}$ and $\mathbf{S}$ execute
     $$(st'_n,\mathsf{EMM}'_n) \leftarrow \Delta.\mathsf{Insert}_{\mathbf{C},\mathbf{S}}\Big(\Big(K_n,st_n,\big(\ell_{\pi(j)},\mathbf{v}_n\big)\Big),\mathsf{EMM}_n\Big);$$
  4. $\mathbf{C}$ increments the counter $\mathsf{count} := \mathsf{count} + \mu$;
  5. $\mathbf{C}$ outputs an updated state $st' = (st_o,st'_n,\mathsf{MM}_{\mathsf{stash}},\pi,\mathsf{count})$, while $\mathbf{S}$ outputs an updated structure $\mathsf{EMM}' = (\mathsf{EMM}_o,\mathsf{EMM}'_n)$.

Figure 2: FIX (Part 2).

Concrete efficiency. We now provide a concrete efficiency analysis assuming Σ is instantiated with the static volume-hiding scheme VLH [31][9] and that Δ is instantiated with our DVLH construction detailed in Section 5. We assume that both VLH and DVLH are instantiated with an optimal-time multi-map encryption scheme. The communication complexity of FIX.Get is then equal to

$$O(\lambda_o + 2^{s_o} + \lambda_n + 2^{s_n})$$

where λ_o (resp., λ_n) is the parameter of the dynamic pseudo-random transform used in Σ (resp., Δ) and 2^{s_o} (resp., 2^{s_n}) is the largest value outputted by the PRF used by the transform in Σ (resp., Δ).

The query complexity is

$$O(\lambda_o + 2^{s_o} + \lambda_n + 2^{s_n} + \mathsf{dels}_o(\ell) + \#\mathsf{MM}_{\mathsf{stash}}[\ell]),$$

the add and erase communication complexities are

$$O(\mu \cdot (\lambda_o + 2^{s_o} + \lambda_n + 2^{s_n}))$$

and the add and erase computational complexities are

$$O(\#\mathbf{v} + \mu \cdot (\lambda_o + 2^{s_o} + \lambda_n + 2^{s_n} + \mathsf{dels}_o(\ell) + \#\mathsf{MM}_{\mathsf{stash}}[\ell])).$$

To get a better sense of the concrete efficiency of our scheme we set the parameters as follows. If μ is set to $O(\log \#\mathbb{L})$, λ_n is set to $O(\lambda_o) = O(\log \#\mathbb{L})$. and s_n to $s_o + \log\log \#\mathbb{L}$ then the query communication and computational complexities are

$$O(\log(\#\mathbb{L}) \cdot 2^{s_o}),$$

and the append and erase communication and computational complexities are

$$O(\log^2(\#\mathbb{L}) \cdot 2^{s_o}),$$

where $s_o \approx 10$ for a multi-map with $50,000$ labels, 2^{22} total values and with response-lengths that are Zipf-distributed.

Correctness. Notice that FIX, when instantiated with VLH and DVLH, will result in a lossy multi-map encryption scheme. As shown in [31], one way to measure lossyness is to calculate the number of truncated labels in the encrypted multi-map. In particular, one can observe that for FIX the number of truncated labels is at most equal to the sum of the number of truncated labels in both VLH and DVLH. We refer the reader to [31] and Section 5.2 for more details on the number of truncations in VLH and DVLH, respectively.

4.1 Client Stash Size

We now analyze the size of the client stash. Since append and erase operations both affect the stash, we use the term *update* to refer to both operations. We consider two settings: (1) the *worst-case* setting where we analyze the stash size independently of the update and tuple length distributions; and (2) an *average-case* setting where updates are chosen uniformly at random and where the update tuple lengths are Zipf distributed.

[9] If truncations in the static encrypted multi-map are not desirable, Σ can be instantiated with AVLH from [31]

Worst case. In this setting, we assume that each update tuple has maximum tuple length s_{max}, i.e., for all updates $\mathsf{op} = (\omega, \ell, \mathbf{v})$, where $\omega \in \{\mathtt{app}, \mathtt{del}\}$, $\#\mathbf{v} = s_{max}$. Recall that FIX has $\#\mathbb{L}_{\mathsf{MM}}/\mu$ update cycles where for each cycle the tuples of μ labels get pushed to the encrypted structure. Note that the mapping π that ties a label to its cycle is fixed at setup and never changes. Note that the worst case occurs if every update is mapped to the update cycle that precedes the current one because, in this case, the updates remain in the stash for $\#\mathbb{L}/\mu - 1$ cycles before being evicted and this is the maximum number of cycles possible. Therefore, the size of the stash is at most $\#\mathbb{L}_{\mathsf{MM}} \cdot s_{max}/\mu$, which means it is $O(\#\mathbb{L}^2_{\mathsf{MM}})$ if s_{max} is $O(\#\mathbb{L}_{\mathsf{MM}})$. Interestingly, if we set μ to be $O(s_{max})$, the stash size is $O(\#\mathbb{L}_{\mathsf{MM}})$ which is the size of the state for most state-of-the-art dynamic EMMs. For the rest of this subsection we set $m \stackrel{\circ}{=} \#\mathbb{L}_{\mathsf{MM}}$.

Average case. The above bound on the stash size helps us understand the limitations of FIX in the worst case but it does not tell us much about how the stash size behaves in practice since clients are very unlikely to generate updates as described above and that all have size s_{max}. This motivates us to study a setting where the update lengths are sampled from some known distribution. More precisely, we assume that the labels to be updated are selected uniformly at random and that the size of their updates is picked uniformly at random from the set

$$U = \left\{ \frac{s_0}{1^s \cdot H_{m,s}}, \frac{s_0}{2^s \cdot H_{m,s}}, \ldots, \frac{s_0}{m^s \cdot H_{m,s}} \right\},$$

where $s_0 = s_{\max} \cdot H_{m,s}$. In the Theorem below, we analyze the expected stash size in this average case setting.

Theorem 4.1. *If the update labels are sampled uniformly at random and if their update lengths are sampled uniformly at random from U, then the expected stash size of FIX is at most $H_m \cdot s_{\max}/2\mu$.*

Proof. We model the stash as a set of $t \stackrel{\circ}{=} m/\mu$ bins $\mathbf{B}_1, \ldots, \mathbf{B}_t$, where each bin is associated to μ unique labels according to the random permutation π. We then consider the qth update experiment where an "update label" $\ell_{(q)}$ is sampled uniformly at random, the size of its tuple $\mathbf{v}_{(q)}$ is sampled uniformly at random from U and $\mathbf{v}_{(q)}$ is placed in $\ell_{(q)}$'s associated bin $\mathbf{B}_{\pi(\ell_{(q)})}$. Finally, bin $\mathbf{B}_q$ is emptied.[10]

Let $L_{(q)}$ and $S_{(q)}$ be the random variables that output the number of labels in the stash and the size of the stash after the qth update, respectively. We say that $\ell_{(i)}$, for $1 \leq i \leq q$, *survives* until the qth update experiment if it lands in a bin that was not emptied in the update experiments i through q. First, notice that none of the labels $\ell_{(i)}$ for $1 \leq i \leq q-t+1$ survive until the qth update experiment. Furthermore, note that $\ell_{(q)}$ survives if it lands in any of the bins in $\{\mathbf{B}_1, \ldots, \mathbf{B}_t\} \setminus \mathbf{B}_{y_0}$, where $1 \leq y_0 \leq t$ is such that $y_0 \equiv q \pmod t$. Similarly, $\ell_{(q-1)}$ survives if it lands in any of the bins in $\{\mathbf{B}_1, \ldots, \mathbf{B}_t\} \setminus \{\mathbf{B}_{y_0}, \mathbf{B}_{y_1}\}$, where $1 \leq y_1 \leq t$ is such that $y_1 \equiv q-1 \pmod t$. More generally, for all $0 \leq i \leq t-2$, $\ell_{(q-i)}$ survives if it lands in any of the bins in $\{\mathbf{B}_1, \ldots, \mathbf{B}_t\} \setminus \{\mathbf{B}_{y_z}\}_{0 \leq z \leq i}$, where $1 \leq y_z \leq t$ is such that $y_z \equiv q - z \pmod t$. Let X_i be the random variable that outputs 1 if $\ell_{(q-i)}$ survives and outputs 0 otherwise. It follows that $L_{(q)} = \sum_{i=0}^{t-2} X_i$ and that

$$\mathbb{E}[L_{(q)}] = \sum_{i=0}^{t-2} \mathbb{E}[X_i] = \sum_{i=0}^{t-2} \Pr[X_i = 1] = \sum_{i=0}^{t-2} (t - (1+i)) \cdot \frac{1}{t} = \sum_{i=1}^{t-1} \frac{t-i}{t} = \sum_{i=1}^{t-1} \frac{i}{t} = \frac{t-1}{2}. \quad (1)$$

[10] Note that for the proof, we assume that the number of updates, q, is larger than t. The result will hold when $q < t$ simply because fewer elements were inserted.

Let $T_{i,j}$ be the random variable that outputs the tuple size of the jth label in $\mathbf{B}_i$ after the qth update experiment. We then have,

$$S_{(q)} = \sum_{i=1}^{t} \sum_{j=1}^{L_i} T_{i,j},$$

where L_i is the random variable that outputs the number of labels in $\mathbf{B}_i$ after the qth update experiment. It follows that,

$$\begin{aligned}
\mathbb{E}[S_{(q)}] &= \sum_{i=1}^{t} \mathbb{E}\left[\sum_{j=1}^{L_i} T_{i,j}\right] \\
&= \sum_{i=1}^{t} \mathbb{E}[L_i] \cdot \mathbb{E}[T_{i,1}] && (2) \\
&= \mathbb{E}[T_{1,1}] \cdot \sum_{i=1}^{t} \mathbb{E}[L_i] && (3) \\
&= \mathbb{E}[T_{1,1}] \cdot \mathbb{E}\left[\sum_{i=1}^{t} L_i\right] \\
&= \mathbb{E}[T_{1,1}] \cdot \mathbb{E}\left[L_{(q)}\right] && (4)
\end{aligned}$$

where Equation (2) follows from Wald's identity (since the L_i's are independent of the $T_{i,j}$'s) and from the fact that the $T_{i,j}$'s are i.i.d., where Equation (3) is also due to the $T_{i,j}$'s being i.i.d., and where Equation (4) follows from $L_{(q)} = \sum_{i=1}^{t} L_i$.

Now recall that $T_{1,1}$ outputs the length of the 1st tuple in $\mathbf{B}_1$ and that it is sampled uniformly at random from U. We therefore have

$$\mathbb{E}[T_{1,1}] = \sum_{i=1}^{m} \frac{1}{m} \cdot \frac{s_{\max} \cdot H_{m,1}}{i \cdot H_{m,1}} = \frac{s_{\max}}{m} \cdot \sum_{i=1}^{m} \frac{1}{i} = \frac{H_m \cdot s_{\max}}{m} \tag{5}$$

where the third equality follows from the fact $\sum_{i=1}^{m} 1/i = H_m$. Plugging Equations (5) and (1) into Equation (4), we have

$$\mathbb{E}[S_{(q)}] = \frac{H_m \cdot s_{\max}}{m} \cdot \frac{t-1}{2} = \frac{H_m \cdot s_{\max} \cdot (m-\mu)}{2\mu \cdot m} \leq \frac{H_m \cdot s_{\max}}{2\mu},$$

where the second equality follows from $t = m/\mu$.

■

The key takeaway from this Theorem is that if we set μ to be $O(\log m)$, the expected stash size is in the order of $s_{\max}$ (which is at least equal to m) since H_m is upper bounded by $\log m + 1$.

Concentration bound. For a more robust analysis, we give a high-probability bound on the stash in the following Theorem.

Theorem 4.2. *If the update labels sampled uniformly at random and if their tuple lengths are sampled from a $\mathcal{Z}_{m,1}$ distribution, then, with probability greater than $1-\epsilon$ the stash size X is at most*

$$\frac{H_m \cdot s_{\max}}{2\mu} + s_{\max} \cdot \sqrt{\frac{(m-2\mu) \cdot \ln(1/\epsilon)}{2\mu}}.$$

Proof. We model the stash similarly to Theorem 4.1 and use the same notation. The main difference is that we define the random variable $S_{(q)}$ (size of the stash) as:

$$S_{(q)} = \sum_{i=0}^{t-2} X_i \cdot T_i,$$

where X_i is the random variable that outputs 1 if $\ell_{(q-i)}$ survives and outputs 0 otherwise, and T_i is the random variable that outputs the tuple size of the $(q-i)$th update. It is easy to see that $S_{(q)}$ will only account for the tuple lengths of the labels that survive which occurs when X_i is equal to 1.

From Theorem 4.1, we also know that

$$\mathbb{E}[T_i] = \frac{H_m \cdot s_{\max}}{m} \quad \text{and} \quad \sum_{i=0}^{t-2} \mathbb{E}[X_i] = \frac{t-1}{2}$$

Now observe that the X_i's and T_i's are independent given that the tuple length is sampled independently of whether the label "survives" or not. We can bound the expected value of $S_{(q)}$ as

$$\mathbb{E}[S_{(q)}] = \sum_{i=0}^{t-2} \mathbb{E}[X_i \cdot T_i] = \sum_{i=0}^{t-2} \mathbb{E}[X_i] \cdot \mathbb{E}[T_i] = \frac{H_m \cdot s_{\max}}{m} \cdot \frac{t-1}{2} \leq \frac{H_m \cdot s_{\max}}{2\mu}. \tag{6}$$

We set $Y_i \stackrel{\circ}{=} X_i \cdot T_i$, for all $i \in \{0, \cdots, t-2\}$ and note that the Y_i's are independent given that the X_i's and T_i's are independent. Note also that $0 \leq Y_i \leq s_{\max}$. Using Hoeffding's inequality, we then have, for all $\delta > 0$,

$$\begin{aligned}
\Pr\left[S_{(q)} \leq \frac{H_m \cdot s_{\max}}{2\mu} + \delta \right] &\geq \Pr\left[S_{(q)} \leq \mathbb{E}[S_{(q)}] + \delta \right] \\
&\geq 1 - \exp\left(\frac{-2\delta^2}{\sum_{i=0}^{t-2} (s_{\max} - 0)^2} \right) \\
&= 1 - \exp\left(\frac{-2\delta^2}{(t-2) \cdot s_{\max}^2} \right) \\
&= 1 - \exp\left(\frac{-2\delta^2 \cdot \mu}{(m-2\mu) \cdot s_{\max}^2} \right).
\end{aligned}$$

Setting

$$\delta = s_{\max} \cdot \sqrt{\frac{(m-2\mu) \cdot \ln(1/\epsilon)}{2\mu}}$$

ends finalizes the proof.

■

4.2 Security

We describe and prove the leakage profile of FIX. We first give a black-box description based on the leakage profiles of Σ and Δ and then describe a concrete profile when Σ and Δ are instantiated with state-of-the-art constructions.

Black-box leakage. Let Σ be a static multi-map encryption scheme with leakage profile $\Lambda_\Sigma = (\mathcal{L}^\Sigma_{\mathsf{S}}, \mathcal{L}^\Sigma_{\mathsf{Q}})$ and Δ be a dynamic multi-map encryption scheme with leakage profile $\Lambda_\Delta = (\mathcal{L}^\Delta_{\mathsf{S}}, \mathcal{L}^\Delta_{\mathsf{O}})$. The setup leakage of FIX is

$$\mathcal{L}_{\mathsf{S}}(\mathsf{MM}) = \Big(\mathcal{L}^\Sigma_{\mathsf{S}}(\mathsf{MM}_o), \mathcal{L}^\Delta_{\mathsf{S}}(\mathsf{MM}_\perp)\Big),$$

where MM_o is the original multi-map and $\mathsf{MM}_\perp$ is the multi-map with label space $\mathbb{L}_{\mathsf{MM}}$ and $\mathsf{MM}_\perp[\ell] = \perp$ for all $\ell \in \mathbb{L}_{\mathsf{MM}}$. The operation leakage of FIX is

- $\mathcal{L}_{\mathsf{O}}(\mathsf{MM}, \mathsf{op})$
  1. if op is a query, parse op as ℓ and output $\mathcal{L}^\Sigma_{\mathsf{Q}}(\mathsf{MM}_o, \ell)$ and $\mathcal{L}^\Delta_{\mathsf{O}}(\mathsf{MM}_n, \ell)$;
  2. if op is an append or delete operation then, for all $0 \leq i \leq \mu - 1$,
     (a) let $j = \mathsf{count} + i \mod m$
     (b) output
        i. $\mathcal{L}^\Sigma_{\mathsf{Q}}(\mathsf{MM}_o, \ell_{\pi(j)})$,
        ii. $\mathcal{L}^\Delta_{\mathsf{O}}(\mathsf{MM}_n, (\mathtt{qry}, \ell_{\pi(j)}))$,
        iii. $\mathcal{L}^\Delta_{\mathsf{O}}(\mathsf{MM}_n, (\mathtt{del}, \ell_{\pi(j)}))$,
        iv. $\mathcal{L}^\Delta_{\mathsf{O}}(\mathsf{MM}_n, (\mathtt{add}, \ell_{\pi(j)}, \mathbf{v}_n))$
     (c) set $\mathsf{count} = \mathsf{count} + \mu$;

where count is a stateful variable initialized to 1.

Theorem 4.3. *Let $\Lambda_{\mathsf{FIX}} = (\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{O}})$ where $\mathcal{L}_{\mathsf{S}}$ and $\mathcal{L}_{\mathsf{O}}$ are as above. If Σ is Λ_Σ-secure and Δ is Λ_Δ-secure, then FIX is Λ_{FIX}-secure.*

Proof sketch: Let $\mathcal{S}_\Sigma$ and $\mathcal{S}_\Delta$ be the simulators guaranteed to exist from the adaptive security of Σ and Δ and consider the simulator $\mathcal{S}$ that works as follows.

Simulating setup: given $\mathcal{L}^\Sigma_{\mathsf{S}}(\mathsf{MM}_o)$ and $\mathcal{L}^\Delta_{\mathsf{S}}(\mathsf{MM}_\perp)$, the simulator outputs $\mathsf{EMM}_o \leftarrow \mathcal{S}_\Sigma(\mathcal{L}^\Sigma_{\mathsf{S}}(\mathsf{MM}_o))$ and $\mathsf{EMM}_n \leftarrow \mathcal{S}_\Delta(\mathcal{L}^\Delta_{\mathsf{S}}(\mathsf{MM}_\perp))$.

Simulating operations: if op is a get the simulator receives $\mathcal{L}^\Sigma_{\mathsf{Q}}(\mathsf{MM}_o, \ell)$ and $\mathcal{L}^\Delta_{\mathsf{O}}(\mathsf{MM}_n, \ell)$ and uses

1. $\mathcal{S}_\Sigma(\mathcal{L}^\Sigma_{\mathsf{Q}}(\mathsf{MM}_o, \ell))$ to simulate the query protocol of Σ, where $\mathcal{S}_\Sigma$ plays the role of the client;
2. $\mathcal{S}_\Delta(\mathcal{L}^\Delta_{\mathsf{O}}(\mathsf{MM}_n, \ell))$ to simulate the query protocol of Δ, where $\mathcal{S}_\Delta$ plays the role of the client.

If op is an append or erase operation,

- for all $0 \leq i \leq \mu - 1$ and $j = \mathsf{count} + i \mod m$ the simulator,
  1. receives $\mathcal{L}^\Sigma_{\mathsf{Q}}(\mathsf{MM}_o, \ell_{\pi(j)})$ and uses $\mathcal{S}_\Sigma\left(\mathcal{L}^\Sigma_{\mathsf{Q}}\left(\mathsf{MM}, \ell_{\pi(j)}\right)\right)$ to simulate the query protocol of Σ, where $\mathcal{S}_\Sigma$ plays the role of the client;

2. receives $\mathcal{L}^{\Delta}_{\mathsf{O}}(\mathsf{MM}_n, (\mathtt{qry}, \ell_{\pi(j)}))$ and uses $\mathcal{S}_{\Delta}\left(\mathcal{L}^{\Delta}_{\mathsf{O}}\left(\mathsf{MM}_n, (\mathtt{qry}, \ell_{\pi(j))})\right)\right)$ to simulate the query protocol of Δ, where $\mathcal{S}_{\Delta}$ plays the role of the client;
3. receives $\mathcal{L}^{\Delta}_{\mathsf{O}}(\mathsf{MM}_n, (\mathtt{del}, \ell_{\pi(j)}))$ and uses $\mathcal{S}_{\Delta}\left(\mathcal{L}^{\Delta}_{\mathsf{O}}\left(\mathsf{MM}_n, (\mathtt{del}, \ell_{\pi(j)})\right)\right)$ to simulate the delete protocol of Δ, where $\mathcal{S}_{\Delta}$ plays the role of the client;
4. receives $\mathcal{L}^{\Delta}_{\mathsf{O}}(\mathsf{MM}_n, (\mathtt{add}, \ell_{\pi(j)}, \mathbf{v}_n))$ and uses $\mathcal{S}_{\Delta}\left(\mathcal{L}^{\Delta}_{\mathsf{O}}\left(\mathsf{MM}_n, (\mathtt{add}, \ell_{\pi(j)}, \mathbf{v}_n)\right)\right)$ to simulate the add protocol of Δ, where $\mathcal{S}_{\Delta}$ plays the role of the client,

It remains to show that for all probabilistic polynomial-time adversaries $\mathcal{A}$, the probability that $\mathbf{Real}(k)$ outputs 1 is negligibly close to the probability that $\mathbf{Ideal}(k)$ outputs 1. This can be shown with the following sequence of games:

1. Game_0: is the same as $\mathbf{Real}_{\mathsf{FIX},\mathcal{A}}(k)$.
2. Game_1: is the same as Game_0 except that: (1) EMM_o is replaced by the output of $\mathcal{S}_{\Sigma}(\mathcal{L}^{\Sigma}_{\mathsf{S}}(\mathsf{MM}_o))$; and (2) all executions of $\Sigma.\mathsf{Get}$ on some label ℓ is replaced by a simulated execution between $\mathcal{S}_{\Sigma}(\mathcal{L}^{\Sigma}_{\mathsf{Q}}(\mathsf{MM}_o, \ell))$ and the adversary.
3. Game_2: is the same as Game_1 except that: (1) EMM_n is replaced by the output of $\mathcal{S}_{\Delta}(\mathcal{L}^{\Delta}_{\mathsf{S}}(\mathsf{MM}_{\perp}))$; (2) all executions of $\Delta.\mathsf{Get}$ on a label ℓ is replaced by a simulated execution between $\mathcal{S}_{\Delta}(\mathcal{L}^{\Delta}_{\mathsf{O}}(\mathsf{EMM}_n, \ell))$ and the adversary; (3) all executions of $\Delta.\mathsf{Insert}$ on an add operation $\mathsf{op} = (\mathtt{add}, \ell, \mathbf{v})$ is replaced by a simulated execution between $\mathcal{S}_{\Delta}(\mathcal{L}^{\Delta}_{\mathsf{O}}(\mathsf{EMM}_n, \mathsf{op}))$ and the adversary; and (4) all executions of $\Delta.\mathsf{Delete}$ on a delete operation $\mathsf{op} = (\mathtt{del}, \ell)$ is replaced by a simulated execution between $\mathcal{S}_{\Delta}(\mathcal{L}^{\Delta}_{\mathsf{O}}(\mathsf{EMM}_n, \mathsf{op}))$.

Note that Game_2 is the $\mathbf{Ideal}_{\mathcal{L},\mathcal{A},\mathcal{S}_{\mathsf{FIX}}}(k)$ experiment and that Game_0 and Game_1 are indistinguishable by the Λ_{Σ}-security of Σ and that Game_1 and Game_2 are indistinguishable by the Λ_{Δ}-security of Δ.

■

Concrete leakage. We now analyze the leakage profile of FIX when its underlying EMMs are instantiated with concrete schemes. Specifically, we assume Σ is a static volume-hiding multi-map encryption scheme with leakage profile

$$\Lambda_{\Sigma} = (\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{Q}}) = (\mathsf{dsize}(\mathsf{MM}_o), \mathsf{qeq}),$$

where dsize is the data size pattern $\mathsf{dsize}(\mathsf{MM}) = m_{\mathsf{MM}}$. This leakage is acheived by the static volume-hiding scheme VLH [31]. We also assume that Δ has leakage profile

$$\Lambda_{\Delta} = (\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{O}}) = (\mathsf{dsize}(\mathsf{MM}_n), \mathsf{oeq}).$$

Note that the above leakage profile can be achieved when Δ is instantiated with our DVLH construction from Section 5.1.

Theorem 4.4. *If Σ is $(\mathsf{dsize}, \mathsf{qeq})$-secure and if Δ is $(\mathsf{dsize}, \mathsf{oeq})$-secure, then FIX is injection-secure.*

Proof. It follows from Theorem 4.3 that, when Σ and Δ have the above leakage profiles, FIX has setup leakage

$$\mathcal{L}_{\mathsf{S}} = (\mathsf{dsize}(\mathsf{MM}_o), \mathsf{dsize}(\mathsf{MM}_n)) = m_{\mathsf{MM}},$$

and operation leakage

- $\mathcal{L}_{\mathsf{O}}(\mathsf{MM}, \mathsf{op})$:
  1. if op is a query, parse op as ℓ and output $\mathsf{qeq}(\mathsf{MM}_o, \ell)$ and $\mathsf{oeq}(\mathsf{MM}_n, \ell)$;
  2. if op is an append or erase operation then, for all $0 \leq i \leq \mu - 1$,
     (a) let $j = \mathsf{count} + i \mod m$
     (b) output
        i. $\mathsf{qeq}(\mathsf{MM}_o, \ell_{\pi(j)})$,
        ii. $\mathsf{oeq}(\mathsf{MM}_n, (\mathtt{qry}, \ell_{\pi(j)}))$,
        iii. $\mathsf{oeq}(\mathsf{MM}_n, (\mathtt{del}, \ell_{\pi(j)}))$,
        iv. $\mathsf{oeq}(\mathsf{MM}_n, (\mathtt{add}, \ell_{\pi(j)}, \mathbf{v}_n))$
  3. set $\mathsf{count} = \mathsf{count} + \mu$;

where count is a stateful variable initialized to 1. We now show that $\mathcal{L}_{\mathsf{O}}$ can be simulated from the following leakage function

- $\mathcal{L}'_{\mathsf{O}}(\mathsf{MM}, \mathsf{op})$:
  1. if op is a query, parse op as ℓ and output $\mathsf{qeq}(\mathsf{MM}_o, \ell)$ and $\mathsf{qeq}(\mathsf{MM}_n, \ell)$[11];
  2. if op is an append or erase operation, output $\perp$.

More precisely, we will show that for any sequence of operations $\mathbf{op} = (\mathsf{op}_1, \ldots, \mathsf{op}_n)$, one can simulate the correlation graph induced by $\mathcal{L}_{\mathsf{O}}$ given only the correlation graph induced by $\mathcal{L}'_{\mathsf{O}}$. In the following we use the term update to refer to either an append or erase operation. Fist, note that the sequence of operations $\mathbf{op}$ made by the client results in FIX making another sequence of operations $\boldsymbol{\omega} = (\omega_1, \ldots, \omega_m)$ on EMM_o and EMM_n, where $m > n$. We refer to $\boldsymbol{\omega}$ as the sequence induced by $\mathbf{op}$. More precisely, every query q in $\mathbf{op}$ induces two queries in $\boldsymbol{\omega}$: one on EMM_o and one on EMM_n. Similarly, every update u in $\mathbf{op}$ induces μ sets of 4 operations in $\boldsymbol{\omega}$: one query on MM_o, one query on MM_n and two updates on MM_n. For notational convenience, we define the function $\varphi : \mathbf{op} \to 2^{\boldsymbol{\omega}}$ to map the set of operations in $\boldsymbol{\omega}$ that correspond to an operation op in $\mathbf{op}$.

Note also that the correlation graph of $\mathcal{L}_{\mathsf{O}}$ on $\mathbf{op}$ is really a correlation graph on $\boldsymbol{\omega}$; i.e., the vertices are operations in $\boldsymbol{\omega}$ and the edges are correlations created by qeq and oeq as described in $\mathcal{L}_{\mathsf{O}}$. In particular, this graph has three kinds of edges: (1) u-u edges which correlate update nodes; (2) q-u edges which correlate queries and updates; and (3) q-q edges which correlate query nodes.

The u-u edges are fixed and independent of $\mathbf{op}$ so they can be simulated without any knowledge. More precisely, we know that there always exist a u-u edge between the ith update in $\boldsymbol{\omega}$ and the jth update in $\boldsymbol{\omega}$ if $j < i$ and $j = i \mod m_{\mathsf{MM}}$. [12] This is because FIX only executes updates on a fixed schedule; specifically, a particular label is only updated every m_{MM} update on MM_n. The q-u edges can also be simulated without any knowledge as follows. For a query q in $\mathbf{op}$, let i be its position in $\boldsymbol{\omega}$. Choose an unused index j' uniformly at random in the set $\{1, \ldots, m_{\mathsf{MM}}\}$ and let j be the index of the j'th update in $\boldsymbol{\omega}$. If $j < i$ then add a q-u edge between the second query in $\varphi(q)$ and u_j, where u_j is the j'th update in $\boldsymbol{\omega}$. If $j > i$, then do not add an edge but remember the correlation and add the edge when the j'th induced update occurs.

[11] As the labels are updated in a specific order defined by the random permutation π, and the leakage of an actual update and a dummy update is indistinguishable, the only leakage during query on MM_n is qeq and not oeq.

[12] We stress that we are referring to the ith and jth updates in $\boldsymbol{\omega}$ and not the ith and jth operations in $\boldsymbol{\omega}$.

Now, add edges between the second query in $\varphi(q)$ and any update in $\boldsymbol{\omega}$ that has a u-u edge with u_j.

Summarizing, FIX has leakage profile

$$\Lambda_{\mathsf{FIX}} = (\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{O}}) = (m_{\mathsf{MM}}, \mathsf{qeq}),$$

which, by Theorem 3.3, implies that it is secure against file injections.

■

5 A Dynamic Volume-Hiding Multi-Map Encryption Scheme

In this section, we introduce our new fully-dynamic volume-hiding multi-map encryption scheme we call DVLH. Our scheme is built on top of a dynamic variant of the pseudo-random transform PRT [31]. We first describe the dynamic version of the PRT, called DPRT, in Figure 3 and provide a high level description in the following.

Overview. DPRT is a data structure transformation that takes as input a security parameter k, a public parameter λ and a multi-map MM, and outputs a transformed multi-map MM'. Both the multi-map transformation and Get algorithms of DPRT are the same as the ones of PRT. The only difference is that DPRT now has an Insert, Append and an Erase algorithm that can add label/values tuple to the multi-map or append/erase values to/from a tuple in the multi-map. More precisely, all these algorithms take as input a security parameter k, a public parameter λ, a label/values tuple $(\ell, \mathbf{v})$ and a multi-map MM. The Insert algorithm executes DPRT on a multi-map composed of a single label/tuple pair $(\ell, \mathbf{v})$, and then adds the transformed tuple to the input multi-map. Both Append and Erase algorithms initialize an empty multi-map $\mathsf{MM}^\star$. Then, Append (resp., Erase) removes the dummy entries (if any) from $\mathsf{MM}[\ell]$ and adds the tuple $(\ell, \mathsf{MM}[\ell]||\mathbf{v})$ (resp., $(\ell, \mathsf{MM}[\ell] \setminus \mathbf{v})$) to $\mathsf{MM}^\star$. It then executes $\mathsf{MM}' \leftarrow \mathsf{DPRT}(1^k, \lambda, \mathsf{MM}^\star)$ and replaces $\mathsf{MM}[\ell]$ with the transformed tuple $\mathsf{MM}'[\ell]$.[13]

5.1 Our Construction

Our dynamic volume-hiding scheme $\mathsf{DVLH} = (\mathsf{Setup}, \mathsf{Get}_{\mathbf{C},\mathbf{S}}, \mathsf{Insert}_{\mathbf{C},\mathbf{S}}, \mathsf{Delete}_{\mathbf{C},\mathbf{S}}, \mathsf{Append}_{\mathbf{C},\mathbf{S}}, \mathsf{Erase}_{\mathbf{C},\mathbf{S}})$ makes use of DPRT—the dynamic version of the pseudo-random transform PRT and black-box use of a dynamic response-hiding multi-map encryption scheme $\Sigma^{\mathsf{rh}}_{\mathsf{MM}}$ that supports both Put and Delete operations, where Delete takes as input a label ℓ and removes its tuple from the multi-map. For our purposes, we would also like $\Sigma^{\mathsf{rh}}_{\mathsf{MM}}$ to be optimal-time and to have update leakage at most $\mathcal{L}_{\mathsf{U}} = \#\mathbf{v}$. These last two properties are achieved by the constructions given in [1, 8] but these constructions do not support Delete; instead, they only support edit^- which takes as input a label ℓ and a value $1 \leq i \leq \#\mathsf{MM}[\ell]$ and removes the ith element of ℓ's tuple. Note, however, that these schemes also support edit^+ which takes as input a label ℓ and a value v and adds v to ℓ's tuple and that this is enough to support Delete with leakage $\mathcal{L}_{\mathsf{O}}(\mathsf{Delete}, \ell) = \perp$ as follows. Given a label ℓ, it suffices to call the underlying scheme' s edit^+ operation on the pair $(\ell, \star)$, where $\star$ is a special symbol outside

[13] In this description, we made the implicit assumption that the client knows whether a label ℓ already exists in the multi-map when it performs an Insert or an Append. In general, this is not the case and there would be a need for the client to keep a local state that stores such information.

Let $F : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^s$ be a pseudo-random function, $r$ank: $\mathbb{R}^n \to \mathbb{R}^n$ be ranking function and $\lambda \in \mathbb{N}$ be a public parameter. Consider the dynamic transform DPRT defined as follows:

- $\mathsf{DPRT}(1^k, \lambda, \mathsf{MM})$:
  1. sample a key $K \xleftarrow{\$} \{0,1\}^k$;
  2. instantiate an empty multi-map MM';
  3. for all $\ell \in \mathbb{L}_{\mathsf{MM}}$,
     (a) let $r := \mathsf{MM}[\ell]$ and $n_\ell = \#r$;
     (b) compute $r' := r\mathrm{rank}(r)$;
     (c) let $n'_\ell = \lambda + F_K(\ell || n_\ell)$;
     (d) if $n'_\ell > n_\ell$, set $\mathsf{MM}'[\ell] := (r', \bot_1, \ldots, \bot_{n'_\ell - n_\ell})$;
     (e) otherwise, set $\mathsf{MM}'[\ell] := (r'_1, \ldots, r'_{n'_\ell})$;
  4. output MM'.
- $\mathsf{Get}(\ell, \mathsf{MM})$: output $\mathsf{MM}[\ell]$.
- $\mathsf{Insert}(1^k, \lambda, (\ell, \mathbf{v}), \mathsf{MM})$:
  1. execute $\mathsf{MM}' \leftarrow \mathsf{DPRT}(1^k, \lambda, \{(\ell, \mathbf{v})\})$;
  2. set $\mathsf{MM}[\ell] := \mathsf{MM}'[\ell]$ and output the updated multi-map MM.
- $\mathsf{Append}(1^k, \lambda, (\ell, \mathbf{v}), \mathsf{MM})$:
  1. instantiate an empty multi-map $\mathsf{MM}^\star$;
  2. remove all dummy entries from $\mathsf{MM}[\ell]$;
  3. set $\mathsf{MM}^\star[\ell] := \mathsf{MM}[\ell] \cup \mathbf{v}$;
  4. execute $\mathsf{MM}' \leftarrow \mathsf{DPRT}(1^k, \lambda, \mathsf{MM}^\star)$;
  5. set $\mathsf{MM}[\ell] := \mathsf{MM}'[\ell]$ and output the updated multi-map MM.
- $\mathsf{Erase}(1^k, \lambda, (\ell, \mathbf{v}), \mathsf{MM})$:
  1. execute $\mathsf{DPRT.Append}(1^k, \lambda, (\ell, \mathbf{v}), \mathsf{MM})$ but replace step 3 with the following:
     set $\mathsf{MM}^\star[\ell] := \mathsf{MM}[\ell] \setminus \mathbf{v}$;

Figure 3: DPRT.

Let DPRT be the dynamic pseudo-random transform described in Figure 3 and let $\Sigma^{\mathsf{rh}}_{\mathsf{MM}} = (\mathsf{Setup}, \mathsf{Get}_{\mathbf{C},\mathbf{S}}, \mathsf{Put}_{\mathbf{C},\mathbf{S}}, \mathsf{Delete}_{\mathbf{C},\mathbf{S}})$ be a dynamic response-hiding multi-map encryption scheme. Consider the scheme $\mathsf{DVLH} = (\mathsf{Setup}, \mathsf{Get}_{\mathbf{C},\mathbf{S}}, \mathsf{Insert}_{\mathbf{C},\mathbf{S}}, \mathsf{Append}_{\mathbf{C},\mathbf{S}}, \mathsf{Delete}_{\mathbf{C},\mathbf{S}}, \mathsf{Erase}_{\mathbf{C},\mathbf{S}})$ defined as follows:

- $\mathsf{Setup}\big(1^k, \lambda, \mathsf{MM}\big)$:
  1. compute $\mathsf{MM}' \leftarrow \mathsf{DPRT}(1^k, \lambda, \mathsf{MM})$ and execute
  $$(K, st, \mathsf{EMM}) \leftarrow \Sigma^{\mathsf{rh}}_{\mathsf{MM}}.\mathsf{Setup}(1^k, \mathsf{MM}');$$
  2. output (K, st, EMM).
- $\mathsf{Get}_{\mathbf{C},\mathbf{S}}\big((K, st, \ell), \mathsf{EMM}\big)$:
  1. $\mathbf{C}$ and $\mathbf{S}$ execute
  $$(r, \perp) \leftarrow \Sigma^{\mathsf{rh}}_{\mathsf{MM}}.\mathsf{Get}_{\mathbf{C},\mathbf{S}}\big((K, st, \ell), \mathsf{EMM}\big);$$
  2. $\mathbf{C}$ outputs r and $\mathbf{S}$ outputs $\perp$.
- $\mathsf{Insert}_{\mathbf{C},\mathbf{S}}\big((K, \lambda, st, (\ell, \mathbf{v})), \mathsf{EMM}\big)$:
  1. $\mathbf{C}$ executes $\mathsf{MM}' \leftarrow \mathsf{DPRT}.\mathsf{Insert}\big(1^k, \lambda, (\ell, \mathbf{v}), \mathsf{MM}\big)$ where MM is an empty multi-map;
  2. $\mathbf{C}$ and $\mathbf{S}$ execute
  $$(st', \mathsf{EMM}') \leftarrow \Sigma^{\mathsf{rh}}_{\mathsf{MM}}.\mathsf{Put}_{\mathbf{C},\mathbf{S}}\Big(\big(K, st, (\ell, \mathsf{MM}'[\ell])\big), \mathsf{EMM}\Big);$$
  3. $\mathbf{C}$ outputs an updated state st' and $\mathbf{S}$ outputs EMM'.
- $\mathsf{Delete}_{\mathbf{C},\mathbf{S}}\big((K, \lambda, st, \ell), \mathsf{EMM}\big)$:
  1. $\mathbf{C}$ and $\mathbf{S}$ execute
  $$(st', \mathsf{EMM}') \leftarrow \Sigma^{\mathsf{rh}}_{\mathsf{MM}}.\mathsf{Delete}_{\mathbf{C},\mathbf{S}}\big((K, st, \ell), \mathsf{EMM}\big);$$
  2. $\mathbf{C}$ outputs an updated state st' and $\mathbf{S}$ outputs EMM'.
- $\mathsf{Append}_{\mathbf{C},\mathbf{S}}\big((K, \lambda, st, (\ell, \mathbf{v})), \mathsf{EMM}\big)$:
  1. $\mathbf{C}$ and $\mathbf{S}$ execute
  $$(r, \perp) \leftarrow \Sigma^{\mathsf{rh}}_{\mathsf{MM}}.\mathsf{Get}_{\mathbf{C},\mathbf{S}}\big((K, st, \ell), \mathsf{EMM}\big)$$
  and,
  $$(st^\star, \mathsf{EMM}^\star) \leftarrow \Sigma^{\mathsf{rh}}_{\mathsf{MM}}.\mathsf{Delete}_{\mathbf{C},\mathbf{S}}\big((K, st, \ell), \mathsf{EMM}\big);$$
  2. $\mathbf{C}$ executes $\mathsf{MM}' \leftarrow \mathsf{DPRT}.\mathsf{Append}\big(1^k, \lambda, (\ell, \mathbf{v}), \{(\ell, r)\}\big)$;
  3. $\mathbf{C}$ and $\mathbf{S}$ execute
  $$(st', \mathsf{EMM}') \leftarrow \Sigma^{\mathsf{rh}}_{\mathsf{MM}}.\mathsf{Put}_{\mathbf{C},\mathbf{S}}\Big(\big(K, st^\star, (\ell, \mathsf{MM}'[\ell])\big), \mathsf{EMM}^\star\Big);$$
  4. $\mathbf{C}$ outputs an updated state st' and $\mathbf{S}$ outputs EMM'.
- $\mathsf{Erase}_{\mathbf{C},\mathbf{S}}\big((K, \lambda, st, (\ell, \mathbf{v})), \mathsf{EMM}\big)$:
  1. execute $\mathsf{DVLH}.\mathsf{Append}(1^k, \lambda, st, (\ell, \mathbf{v}), \mathsf{EMM})$ but replace step 2 with the following:
  $\mathbf{C}$ executes $\mathsf{MM}' \leftarrow \mathsf{DPRT}.\mathsf{Erase}\big(1^k, \lambda, (\ell, \mathbf{v}), \{(\ell, r)\}\big)$;

Figure 4: DVLH.

of the label space. Then, during a get operation, if the client retrieves a tuple that includes $\star$, it outputs $\perp$. The details of DVLH are provided in Figure 4. At a high level it works as follows.

Overview. Unlike $\mathsf{VLH}^{\mathsf{d}}$ in [31], DVLH is fully dynamic in the sense that it handles editing existing tuples. Recall that the dynamism of $\mathsf{VLH}^{\mathsf{d}}$ is restricted to removing a label/tuple pair, adding a new label/tuple pair, and editing an existing tuple while not modifying its size. The Setup and Get protocols are the same as the ones of $\mathsf{VLH}^{\mathsf{d}}$. So we only describe how all the Update protocols work. The Delete algorithm removes the corresponding $(\ell, \mathbf{v})$ pair from the encrypted multi-map. The Append and Erase algorithms execute $r \leftarrow \Sigma^{\mathsf{rh}}_{\mathsf{MM}}.\mathsf{Get}$ and then remove ℓ from the encrypted multi-map. The Append (resp., Erase) algorithm then executes $\mathsf{DPRT.Append}$ (resp., $\mathsf{DPRT.Erase}$) on $(\ell, \mathbf{v})$ and a multi-map consisting of just one tuple $\{(\ell, r)\}$. This outputs a transformed multi-map MM' containing just the updated tuple; ℓ and its corresponding values. Finally, the client and the server execute the $\Sigma^{\mathsf{rh}}_{\mathsf{MM}}.\mathsf{Put}$ protocol to put the new pair $(\ell, \mathsf{MM}'[\ell])$ into the encrypted multi-map. Lastly, the Insert algorithm first executes $\mathsf{DPRT.Insert}$ on $(\ell, \mathbf{v})$ and an empty multi-map. This outputs a transformed multi-map MM' containing the transformed tuple; ℓ and its corresponding values. The client and the server execute the $\Sigma^{\mathsf{rh}}_{\mathsf{MM}}.\mathsf{Put}$ protocol to put the new pair $(\ell, \mathsf{MM}'[\ell])$ into the encrypted multi-map.

Efficiency. If $\Sigma^{\mathsf{rh}}_{\mathsf{MM}}$ has optimal query complexity, the best and worst case communication complexity for $\mathsf{DVLH.Get}$ is the same as $\mathsf{VLH.Get}$ which is $O(\lambda)$ and $O(\lambda + \nu)$, where $\nu = 2^s$ is the largest value that can be taken by the pseudo-random function used by the transform. The average case communication complexity is $O(\lambda + 2^{s-1})$. As for storage, it is the same as VLH. A label/tuple deletion has asymptotically the same cost as the $\Sigma^{\mathsf{rh}}_{\mathsf{MM}}.\mathsf{Delete}$ protocol. As for Insert, it incurs the same communication cost as $\Sigma^{\mathsf{rh}}_{\mathsf{MM}}.\mathsf{Put}$ operation asymptotically, as it executes a Put operation within it.[14] The communication cost of the Append and Erase operations is equal to the sum of the communication costs of the $\Sigma^{\mathsf{rh}}_{\mathsf{MM}}.\mathsf{Get}$, $\Sigma^{\mathsf{rh}}_{\mathsf{MM}}.\mathsf{Delete}$ and $\Sigma^{\mathsf{rh}}_{\mathsf{MM}}.\mathsf{Put}$ operations asymptotically, as it executes a Get, Delete and Put operations within it.

Correctness. Similarly to $\mathsf{VLH}^{\mathsf{d}}$, the correctness of DVLH is also affected by possible truncations. The difference is that tuples can now be subjected to many truncations. A bound on the number of labels effected by truncations is derived in Section 5.2.

Security. We now describe the leakage profile of DVLH assuming $\Sigma^{\mathsf{rh}}_{\mathsf{MM}}$ is instantiated with one of the standard dynamic multi-map encryption schemes [1, 8] all of which have leakage profile $\Lambda_{\mathsf{MM}} = (\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{Q}}, \mathcal{L}_{\mathsf{U}}) = (\mathsf{trlen}, (\mathsf{qeq}, \mathsf{bopeq}, \mathsf{rlen}), \mathsf{ulen})$ where trlen is the total response length ($\mathsf{trlen}(\mathsf{MM}) = \sum_{\ell \in \mathbb{L}} \#\mathsf{MM}[\ell]$), qeq is the query equality pattern that reveals if and when the label has been queried in the past, bopeq is the backward operation equality pattern that reveals whether a query and another operation are over the same label only in the case when the operation occurred before the query, rlen is the response length of a query, and ulen is the number of values in an update. In the following theorem, dsize is the data size pattern which is defined as $\mathsf{dsize}(\mathsf{MM}) = \#\mathbb{L}_{\mathsf{MM}}$ and oeq is the operation equality pattern that captures which operations were over the same label.

[14] Note however that, under the assumption of $\Sigma^{\mathsf{rh}}_{\mathsf{MM}}$ having an optimal query and put complexity, the cost of the Put operation is proportional to the length of the tuple which is in this case equal to $O(\lambda + \nu)$ in the worst case.

Theorem 5.1. *If* $\Sigma^{\mathsf{rh}}_{\mathsf{MM}}$ *is a* $(\mathsf{trlen}, (\mathsf{qeq}, \mathsf{bopeq}, \mathsf{rlen}), \mathsf{ulen})$*-secure dynamic multi-map encryption scheme and* F *is a pseudo-random function, then* DVLH *is a* $(\mathsf{dsize}, \mathsf{oeq}, \mathsf{oeq})$*-secure dynamic multi-map encryption scheme.*

Proof. Let $\mathcal{S}_{\mathsf{MM}}$ be the simulator guaranteed to exist from the adaptive security of $\Sigma^{\mathsf{rh}}_{\mathsf{MM}}$. Consider the simulator $\mathcal{S}_{\mathsf{DVLH}}$, that works as follows:

Simulating setup: given $\mathsf{dsize}(\mathsf{MM}) = \#\mathbb{L}_{\mathsf{MM}}$, it computes $\theta = \sum_{i=1}^{\#\mathbb{L}_{\mathsf{MM}}}(\lambda + r_i)$, where $r_1, \ldots, r_{\#\mathbb{L}_{\mathsf{MM}}} \xleftarrow{\$} \{0,1\}^s$ and outputs $\mathsf{EMM} \leftarrow \mathcal{S}_{\mathsf{MM}}(\theta)$. It then stores $r_1, \ldots, r_{\#\mathbb{L}_{\mathsf{MM}}}$ in a set R.

Simulating operations: if op is the ith Get, $\mathcal{S}_{\mathsf{DVLH}}$ receives $\mathsf{oeq}(\mathsf{MM}, \ell_1, \ldots, \ell_i)$ from which it extracts $\gamma = (\mathsf{qeq}(\mathsf{MM}, \ell_1, \ldots, \ell_i), \mathsf{bopeq}(\mathsf{MM}, \ell_1, \ldots, \ell_i))$. Using oeq, it also determines if the label ℓ_i that is being queried, has been queried or updated before. If this is the first time ℓ_i is being queried for, it removes a random element from the set R and assigns it to ℓ_i. Let us denote this element as r_{ℓ_i}. Otherwise, if the label had been queried in the past, then r_{ℓ_i} had been already assigned . $\mathcal{S}_{\mathsf{DVLH}}$ now uses $\mathcal{S}_{\mathsf{MM}}(\mathcal{L}_{\mathsf{Q}}(\mathsf{MM}, \ell_i))$ where $\mathcal{L}_{\mathsf{Q}}(\mathsf{MM}, \ell_i) = (\mathsf{qeq}, \mathsf{bopeq}, r_{\ell_i} + \lambda)$, to simulate a Get on EMM (recall that λ is public). $\Sigma^{\mathsf{rh}}_{\mathsf{MM}}.\mathsf{Get}$ is an interactive protocol so here $\mathcal{S}_{\mathsf{DVLH}}$ uses $\mathcal{S}_{\mathsf{MM}}$ to play the role of the client.

If op is the ith Insert, Delete, Append or Erase operation, $\mathcal{S}_{\mathsf{DVLH}}$ receives $\mathsf{oeq}(\mathsf{MM}, (\mathsf{op}_1, \ell_1), \ldots, (\mathsf{op}_i, \ell_i))$ from which it extracts $\gamma = (\mathsf{qeq}(\mathsf{MM}, \ell_1, \ldots, \ell_i), \mathsf{bopeq}(\mathsf{MM}, \ell_1, \ldots, \ell_i))$. It also identifies whether the label ℓ_i on which the operation is performed, has been operated on in the past.

1. if op is an Insert operation, $\mathcal{S}_{\mathsf{DVLH}}$ sets r_{ℓ_i} to r where $r \xleftarrow{\$} \{0,1\}^s$. It then uses $\mathcal{S}_{\mathsf{MM}}(\mathcal{L}_{\mathsf{U}}(\mathsf{MM}, (\mathsf{add}, \ell_i, \cdot)))$ where $\mathcal{L}_{\mathsf{U}}(\mathsf{MM}, (\mathsf{add}, \ell_i, \cdot)) = r_{\ell_i} + \lambda$ to simulate a Put on the EMM structure.
2. if op is an Delete operation, $\mathcal{S}_{\mathsf{DVLH}}$ discards r_ℓ if it exists. Then it uses $\mathcal{S}_{\mathsf{MM}}(\perp)$ to simulate a Delete on the EMM.
3. otherwise, there are two sub-cases: if this is the first time ℓ_i is being operated on, it removes a random element from R and assigns it to ℓ_i. Let us denote this element as r_{ℓ_i}. Otherwise, if the label had been operated on before, r_{ℓ_i} already exists. $\mathcal{S}_{\mathsf{DVLH}}$ uses $\mathcal{S}_{\mathsf{MM}}(\mathcal{L}_{\mathsf{Q}}(\mathsf{MM}, \ell_i))$ where $\mathcal{L}_{\mathsf{Q}}(\mathsf{MM}, \ell_i) = (\mathsf{qeq}, \mathsf{bopeq}, r_{\ell_i} + \lambda)$, to simulate a Get on EMM. Then it uses $\mathcal{S}_{\mathsf{MM}}(\perp)$ to simulate a Delete on EMM. $\mathcal{S}_{\mathsf{DVLH}}$ then updates the value of r_{ℓ_i} such that $r_{\ell_i} := r$ where $r \xleftarrow{\$} \{0,1\}^s$. It then uses $\mathcal{S}_{\mathsf{MM}}(\mathcal{L}_{\mathsf{U}}(\mathsf{MM}, (\mathsf{add}, \ell_i, \cdot)))$ such that $\mathcal{L}_{\mathsf{U}}(\mathsf{MM}, (\mathsf{add}, \ell_i, \cdot)) = r_{\ell_i} + \lambda$ to simulate a Put on EMM.

It remains to show that for all probabilistic polynomial-time adversaries $\mathcal{A}$, the probability that $\mathbf{Real}(k)$ outputs 1 is negligibly close to the probability that $\mathbf{Ideal}(k)$ outputs 1. This can be shown with a standard sequence of games:

1. Game_0: is the same as $\mathbf{Real}_{\mathsf{DVLH},\mathcal{A}}(k)$.

2. Game_1: is the same as Game_0 except that the pseudo-random function F is replaced with a random function. Game_1 and Game_0 are indistinguishable, otherwise the pseudo-randomness of F would be violated.

3. Game_2: is the same as Game_1 except that

   - the encrypted multi-map EMM is replaced by the output of $\mathcal{S}_{\mathsf{MM}}(\theta)$;
   - the execution of $\Sigma^{\mathsf{rh}}_{\mathsf{MM}}.\mathsf{Get}_{\mathbf{C},\mathbf{S}}((K, st, \ell), \mathsf{EMM})$ is replaced by a simulated execution between $\mathcal{S}_{\mathsf{MM}}(\mathsf{qeq}, \mathsf{bopeq}, r_\ell + \lambda)$ and the adversary;
   - the execution of $\Sigma^{\mathsf{rh}}_{\mathsf{MM}}.\mathsf{Delete}_{\mathbf{C},\mathbf{S}}((K, st, \ell), \mathsf{EMM})$ is replaced by a simulated execution between $\mathcal{S}_{\mathsf{MM}}(\perp)$ and the adversary, and
   - $\Sigma^{\mathsf{rh}}_{\mathsf{MM}}.\mathsf{Put}_{\mathbf{C},\mathbf{S}}((K, st, (\ell, \mathbf{v})), \mathsf{EMM})$ is replaced by a simulated execution between $\mathcal{S}_{\mathsf{MM}}(r_\ell + \lambda)$ and the adversary.

   Game_2 and Game_1 are indistinguishable, otherwise the $(\mathsf{trlen}, (\mathsf{qeq}, \mathsf{bopeq}, \mathsf{rlen}), \mathsf{ulen})$-security of $\Sigma^{\mathsf{rh}}_{\mathsf{MM}}$ would be violated.

4. Note that Game_3 is the $\mathbf{Ideal}_{\mathcal{L},\mathcal{A},\mathcal{S}_{\mathsf{DVLH}}}(k)$ experiment.

■

5.2 Correctness

To analyze correctness, we need an upper bound on the number of truncations after $t \in \mathbb{N}$ append operations. Here, we use the term "append" interchangeably with "update" since it represents the main non-trivial update. We assume that the updated labels are picked uniformly at random and that only a single value is appended to their tuple.

We focus on multi-maps with Zipf-distributed tuple lengths. More precisely, for a given multi-map size N (i.e., the sum of the tuple lengths) and label space $\mathbb{L}$, we consider multi-maps with tuple lenghts in

$$L = \left\{ \frac{N}{H_{m,1}}, \frac{N}{2 \cdot H_{m,1}}, \ldots, \frac{N}{m \cdot H_{m,1}} \right\},$$

where $m \stackrel{\circ}{=} \#\mathbb{L}$. Before stating our theorem we recall some useful Lemmas related to *negative association.*

Lemma 5.2 (Independence implies negative association [21])**.** *If $X_1, \ldots, X_n$ are independent random variables, then they are negatively associated.*

Lemma 5.3 (Zero-one principle [21])**.** *If $X_1, \ldots, X_n$ are zero-one random variables such that $\sum_{i=1}^{n} X_i \leq 1$, then they are negatively associated.*

Lemma 5.4 (Closure [21])**.** *Negatively associated random variables satisfy the following closure properties:*

- *if $X_1, \ldots, X_n$ are negatively associated, if $Y_1, \ldots, Y_m$ are negatively associated, and if $\{X_i\}_i$ and $\{Y_j\}_j$ are mutually independent, then $X_1, \ldots, X_n, Y_1, \ldots, Y_m$ are negatively associated.*

- *let $X_1, \ldots, X_n$ be negatively associated random variables and let $I_1, \ldots, I_k \subseteq [n]$ be disjoint index sets for $k \geq 1$. Furthermore, let $\{h_j : \mathbb{R}^{\#I_j} \to \mathbb{R}\}_{j \in [k]}$ be a set of functions that are all either non-decreasing or all non-increasing and define $Y_j := h_j(X_i, i \in I_j)$. Then $Y_1, \cdots, Y_k$ are negatively associated. In other words, non-decreasing or non-increasing functions of disjoint subsets of negatively associated variables are also negatively associated.*

Theorem 5.5. *If the tuple lengths of a multi-map are assigned to labels uniformly at random from L (without replacement) and if the update labels are sampled independently and uniformly at random from $\mathbb{L}$, then after t updates the number of truncations is at most*

$$\mu + \sqrt{\frac{m \cdot (t+1) \ln(1/\epsilon)}{2}}$$

with probability at least $1 - \epsilon$, where

$$\mu \leq \frac{1}{\nu}\left(\frac{N \cdot H_{\rho,1}}{H_{m,1}} - \lambda \cdot \rho + \frac{\lambda \cdot N \cdot H_{\rho,1}}{m \cdot H_{m,1}} + \frac{(t-\lambda)}{m}\big(N + (t-\lambda)m\big)\right)$$

for all $m, \lambda \in \mathbb{N}$, $t \geq \lambda$ and $\rho = \lfloor N/(\lambda \cdot H_{m,1}) \rfloor$.

Proof. Any label in the multi-map can get truncated either at setup or at one of the t updates. We denote by $X_i^{(j)}$ the indicator random variable that is equal to 1 if the ith label gets truncated at the jth update if $j > 0$ or at setup when $j = 0$. We are interested in calculating the overall number of truncations cross all labels. More formally we are interested in the concentration around the following quantity:

$$S := \sum_{i=1}^{m} \sum_{j=0}^{t} X_i^{(j)},$$

where m is the size of the label space and t is the number of updates. In the rest of the , we proceed as follows:

- in Claim 1 we bound the probability that a label is truncated at setup time,
- in Claim 2 we bound the probability that a label is truncated at the jth update, when $j \geq 1$,
- in Claim 3 we bound the expected number of truncations,
- in Claim 4 we show that $\{X_i^{(j)}\}_{i,j}$ are negatively associated for all $i \in \{1, \cdots, m\}$ and $j \in \{0, \cdots, t\}$.

We start by introducing some notation. For the ith label and jth update when $j > 0$, or setup when $j = 0$, we denote by $R_i^{(j)}$ the random variable equal to the output of the pseudo-random function (PRF). For simplicity, we model the PRF as a random function with an output space equal to $\{0, \cdots, \nu - 1\}$. That is, $\Pr\left[R_i^{(j)} = k\right] = 1/\nu$, for any $k \in \{0, \cdots, \nu - 1\}$. Similarly, we denote by $Y_i^{(j)}$ the length of the tuple of the ith label at either setup when $j = 0$ or at the jth update when $j > 0$. In the following, we simplify our setting by assuming that the length of the tuple increases with every update. That is, $Y_i^{(j)} = Y_i^{(0)} + j$.[15]

[15]The tuple lengths linearly increase as a function of the number of updates, and as a result, the probability of any tuple to get truncated will increase. This is a worst-case setting given that tuple lengths may decrease as a result of a prior truncations, or simply because the fact that not all labels are not updated all the times (which our equality entails). While this assumption leads to a looser bound, it makes the overall proof simpler.

Claim 1. *We show that*

$$\Pr\left[X_i^{(0)} = 1\right] = \frac{1}{\nu \cdot m}\left(\frac{N \cdot H_{\rho,1}}{H_{m,1}} - \lambda \cdot \rho\right)$$

where $\rho = \lfloor N/(\lambda \cdot H_{m,1}) \rfloor$*, and* $\lambda, \nu \in \mathbb{N}$*.*

Proof. First notice that a truncation occurs when the output of the random function is smaller than the size of the tuple, plus, the constant λ. In particular, a truncation occurs when $R_i^{(j)} < Y_i^{(j)} - \lambda$. So at setup, i.e., when $j = 0$, we have:

$$\begin{aligned}
\Pr\left[X_i^{(0)} = 1\right] &= \Pr\left[R_i^{(0)} < Y_i^{(0)} - \lambda\right] \\
&= \sum_{y \in L} \Pr\left[R_i^{(0)} < Y_i^{(0)} - \lambda \mid Y_i^{(0)} = y\right] \cdot \Pr\left[Y_i^{(0)} = y\right] \\
&= \frac{1}{m} \sum_{y \in L} \Pr\left[R_i^{(0)} < y - \lambda\right] \\
&= \frac{1}{m} \sum_{y \in L, y > \lambda} \sum_{k=0}^{y-\lambda-1} \Pr\left[R_i^{(0)} = k\right] \qquad (7) \\
&= \frac{1}{\nu \cdot m} \sum_{y \in L, y > \lambda} (y - \lambda) \qquad (8)
\end{aligned}$$

Equality 7 holds since the random variable $R_i^{(0)}$ is not defined to output negative values, and therefore we restrict the support to be all elements $y \in L$ such that $y > \lambda$. Given Equation 8, we compute the non-negative integer $\rho \in [m]$ such that

$$\frac{N}{(\rho+1) \cdot H_{m,1}} < \lambda \leq \frac{N}{\rho \cdot H_{m,1}}$$

and reformulate the summation in Equation 8 such that

$$\begin{aligned}
\Pr\left[X_i^{(0)} = 1\right] &= \frac{1}{\nu \cdot m} \sum_{k=1}^{\rho} \left(\frac{N}{k \cdot H_{m,1}} - \lambda\right) \\
&= \frac{1}{\nu \cdot m}\left(\frac{N \cdot H_{\rho,1}}{H_{m,1}} - \lambda \cdot \rho\right)
\end{aligned}$$

■

Claim 2. *If* $j \leq \lambda$*, we show that*

$$\Pr\left[X_i^{(j)} = 1\right] = \frac{1}{\nu \cdot m^2}\left(\frac{N \cdot H_{\rho,1}}{H_{m,1}} - (\lambda - j) \cdot \rho_j\right)$$

Otherwise if $j > \lambda$*, we show that*

$$\Pr\left[X_i^{(j)} = 1\right] = \frac{1}{\nu \cdot m^2}\left(N - (\lambda - j)m\right)$$

where $\rho_j = \lfloor N/((\lambda - j) \cdot H_{m,1}) \rfloor$*, and* $\lambda, \nu \in \mathbb{N}$*.*

Proof. The second case is similar to the one above except that we now have to account for increasing lengths of the tuple, but more importantly, that a label can get truncated only if picked to be updated. We call the event that the ith label gets updated at the jth step $\mathsf{Sel}_{i,j}$. Given our assumption that the labels get updated uniformly at random, we have $\Pr[\,\mathsf{Sel}_{i,j}\,] = 1/m$. Given all of this we obtain the following

$$\begin{aligned}
\Pr\left[X_i^{(j)} = 1\right] &= \Pr\left[X_i^{(j)} = 1 \mid \mathsf{Sel}_{i,j}\right] \cdot \Pr[\,\mathsf{Sel}_{i,j}\,] + \Pr\left[X_i^{(j)} = 1 \mid \overline{\mathsf{Sel}}_{i,j}\right] \cdot \Pr\left[\overline{\mathsf{Sel}}_{i,j}\right] \\
&= \frac{1}{m} \cdot \Pr\left[X_i^{(j)} = 1 \mid \mathsf{Sel}_{i,j}\right] && (9) \\
&= \frac{1}{m} \cdot \Pr\left[R_i^{(j)} < Y_i^{(j)} - \lambda\right] && (10) \\
&= \frac{1}{m} \cdot \sum_{y \in L} \Pr\left[R_i^{(0)} < Y_i^{(0)} + j - \lambda \mid Y_i^{(0)} = y\right] \cdot \Pr\left[Y_i^{(0)} = y\right] \\
&= \frac{1}{\nu \cdot m^2} \sum_{y \in L, y > \lambda - j} (y + j - \lambda) && (11)
\end{aligned}$$

Equality 9 holds since the event of truncating a label when the label itself was not selected to be updated, $\{X_i^{(j)} = 1 \mid \overline{\mathsf{Sel}}_{i,j}\}$, can never occur. Equality 10 holds as a label that gets updated can be truncated iff the output of the random function is smaller than its tuple length minus the constant λ. Equality 11 holds as the support is only defined for values $y \in L$ such that $y > \lambda - j$.

We are going to consider two cases: (*case 1*) when $j \leq \lambda$, and (*case 2*) when $\lambda < j$. For the first case, we identify ρ_j such that

$$\frac{N}{(\rho_j + 1) \cdot H_{m,1}} < \lambda - j \leq \frac{N}{\rho_j \cdot H_{m,1}}$$

And similar to the calculation done in Claim 1, we can show that

$$\Pr\left[X_i^{(j)} = 1\right] = \frac{1}{\nu \cdot m^2}\left(\frac{N \cdot H_{\rho,1}}{H_{m,1}} - (\lambda - j) \cdot \rho_j\right)$$

In the second case, given that $\lambda - j < 0$ all the values $y \in L$ are possible, and therefore Equation 11 is equivalent to

$$\begin{aligned}
\Pr\left[X_i^{(j)} = 1\right] &= \frac{1}{\nu \cdot m^2} \sum_{y \in L} (y + j - \lambda) \\
&= \frac{1}{\nu \cdot m^2} \sum_{k=1}^{m} \left(\frac{N}{k \cdot H_{m,1}} + j - \lambda\right) \\
&= \frac{1}{\nu \cdot m^2}(N - (\lambda - j)m)
\end{aligned}$$

■

Claim 3. *We show that the expected value of S is at most*

$$\mathbb{E}[S] \leq \frac{1}{\nu}\left(\frac{N \cdot H_{\rho,1}}{H_{m,1}} - \lambda \cdot \rho + \frac{\lambda \cdot N \cdot H_{\rho,1}}{m \cdot H_{m,1}} + \frac{(t - \lambda)}{m}(N + (t - \lambda)m)\right)$$

where $\rho = \lfloor N/((\lambda - j) \cdot H_{m,1})\rfloor$, and $\lambda, \nu \in \mathbb{N}$.

Proof. The expected value of S is equal to:

$$\begin{aligned}
\mathbb{E}[S] &= \mathbb{E}[\sum_{i=1}^{m}\sum_{j=0}^{t} X_i^{(j)}] \\
&= \mathbb{E}[\sum_{i=1}^{m} X_i^{(0)}] + \mathbb{E}[\sum_{i=1}^{m}\sum_{j=1}^{t} X_i^{(j)}] \\
&= \mathbb{E}[\sum_{i=1}^{m} X_i^{(0)}] + \mathbb{E}[\sum_{i=1}^{m}\sum_{j=1}^{\lambda} X_i^{(j)}] + \mathbb{E}[\sum_{i=1}^{m}\sum_{j=\lambda+1}^{t} X_i^{(j)}] \\
&= \sum_{i=1}^{m}\mathbb{E}[X_i^{(0)}] + \sum_{i=1}^{m}\sum_{j=1}^{\lambda}\mathbb{E}[X_i^{(j)}] + \sum_{i=1}^{m}\sum_{j=\lambda+1}^{t}\mathbb{E}[X_i^{(j)}] \\
&= \sum_{i=1}^{m}\frac{1}{\nu\cdot m}\left(\frac{N\cdot H_{\rho,1}}{H_{m,1}} - \lambda\cdot\rho\right) + \sum_{i=1}^{m}\sum_{j=1}^{\lambda}\frac{1}{\nu\cdot m^2}\left(\frac{N\cdot H_{\rho,1}}{H_{m,1}} - (\lambda-j)\cdot\rho_j\right) + \sum_{i=1}^{m}\sum_{j=\lambda+1}^{t}\frac{1}{\nu\cdot m^2}(N-(\lambda-j)m) \qquad (12) \\
&\leq \sum_{i=1}^{m}\frac{1}{\nu\cdot m}\left(\frac{N\cdot H_{\rho,1}}{H_{m,1}} - \lambda\cdot\rho\right) + \sum_{i=1}^{m}\sum_{j=1}^{\lambda}\frac{1}{\nu\cdot m^2}\left(\frac{N\cdot H_{\rho,1}}{H_{m,1}}\right) + \sum_{i=1}^{m}\sum_{j=\lambda+1}^{t}\frac{1}{\nu\cdot m^2}(N+(t-\lambda)m) \\
&= \frac{1}{\nu}\left(\frac{N\cdot H_{\rho,1}}{H_{m,1}} - \lambda\cdot\rho + \frac{\lambda\cdot N\cdot H_{\rho,1}}{m\cdot H_{m,1}} + \frac{(t-\lambda)}{m}(N+(t-\lambda)m)\right) \qquad (13)
\end{aligned}$$

Equality 12 holds since $X_{i,j}$ is a Bernoulli random variable and the parameters are from Claims 1 and 2, for all $i \in [m]$ and $j \in \{0,\cdots,t\}$.

■

Claim 4. *We show that the random variables $\{X_{i,j}\}_{i,j}$ are negatively associated, for all $i \in [m]$ and $j \in \{0,\cdots,t\}$.*

Proof. First recall that $\{X_i^{(j)}\}_i$, for $j \in \{1,\cdots,t\}$, are zero-one random variables such that $\sum_{i=1}^{m} X_i^{(j)} \leq 1$. Given Lemma 5.3, we now know that $\{X_i^{(j)}\}_i$ are negatively associated. Moreover, observe that $\{X_i^{(j)}\}_i$ and $\{X_i^{(j')}\}_i$ for distinct j and j' are mutually independent. This follows from the fact that at any update step, the label that gets truncated is independent from the labels that get truncated in the past or in the future. Using the first statement of Lemma 5.4, we now have that $\{X_i^{(j)}\}_{i,j}$ are NA for $i \in [m]$ and $j \in \{1,\cdots,t\}$. Also notice that $\{X_i^{(0)}\}_i$ are independent random variables and using Lemma 5.2, $\{X_i^{(0)}\}_i$ are NA. Again using the first statement of Lemma 5.4, we obtain that $\{X_i^{(j)}\}_{i,j}$ are NA for $i \in [m]$ and $j \in \{0,\cdots,t\}$.

■

Given the result of of Claim 4, we can use Chernoff-Hoeffding bounds to bound S. This gives

us, for $\delta \geq 0$,

$$\Pr\left[\, S > \mathbb{E}[S] + \delta \,\right] \leq \exp\left(-\frac{2\delta^2}{\sum_{i=1}^{m}\sum_{j=0}^{t}(1-0)^2}\right)$$
$$\Pr\left[\, S \leq \mathbb{E}[S] + \delta \,\right] > 1 - \exp\left(-\frac{2\delta^2}{m \cdot (t+1)}\right)$$

Setting $\varepsilon = \exp\left(-2\delta^2/(m \cdot (t+1))\right)$, we have

$$\Pr\left[S \leq \mathbb{E}[S] + \sqrt{\frac{m \cdot (t+1)\ln(1/\epsilon)}{2}} \right] > 1 - \epsilon,$$

where $\mathbb{E}[S]$ is upper bounded in Claim 3.

■

6 An Injection-Secure SSE Scheme

We now present the first searchable symmetric encryption scheme that is secure against injection attacks; i.e., that protects pre- and post-injection queries. The construction can be viewed as a variant of the OPQ scheme of [6] with the underlying multi-map encryption scheme instantiated with FIX and with a simple but important modification. Before describing this change, however, it is useful to consider what happens if we were to use OPQ as-is.

The OPQ scheme. At a high level, the OPQ searchable symmetric encryption scheme is based on an encrypted multi-map but uses it differently than the traditional index-based approach to designing SSE. In other words, instead of encrypting the documents in the collection with a standard encryption scheme and using an EMM to map keywords to document identifiers, OPQ maps the keywords to tuples that consist of the *documents* that contain the keyword. This of course comes at a price in storage but as observed in [6] it suppresses co-occurrence leakage. So one (flawed) attempt at constructing an SSE scheme secure against injection attacks would be to use the OPQ scheme with FIX as its underlying EMM. The problem with this approach is that the insert and delete operations of OPQ reveal the size of the added/deleted document *independently* of the underlying EMM's add/delete leakage profile. This, in turn, means that using OPQ even with FIX as its underlying EMM would not result in an SSE scheme with the appropriate leakage profile.

Overview. To address the limitation discussed above, we modify OPQ so that whenever a document is added or deleted, FIX's insert and delete operations are only invoked a fixed number of times θ which is a public parameter. The scheme, $\mathsf{FixSSE} = (\mathsf{Setup}, \mathsf{Search}, \mathsf{Insert}, \mathsf{Delete})$, is detailed in Figure 5 and works as follows.

Setup. The Setup algorithm takes as input a security parameter 1^k and a document collection $\mathsf{DC} = (D_1, \ldots, D_n)$, where each document D_i is composed of keywords from some space $\mathbb{W}$. It first builds a multi-map MM that maps each keyword $w \in \mathbb{W}$ to a tuple $\mathbf{t} = (t_1, \ldots, t_m)$ constructed as follows. First, it finds the set of documents that contain the keyword w which we denote $\mathsf{DC}(w) = \{D \in \mathsf{DC} : w \in D\}$. It then concatenates all the documents in $\mathsf{DC}(w) = (D_{z_1}, \ldots, D_{z_m})$ resulting in a document $D^\star = D_{z_1} \| \cdots \| D_{z_m}$. The tuple element t_i of $\mathbf{t}$ is then the ith B-sized block of $D^\star$, where $B \in \mathbb{N}_{\geq 1}$. For simplicity we assume that, for all $D \in \mathsf{DC}$, $|D|$ is a multiple of B but if this isn't the case we just pad the last block to be of size B. It then runs FIX.Setup on MM and outputs a key K, a state st and an encrypted multi-map EMM. Finally, the client outputs the key K, the state st and the encrypted document collection $\mathsf{EDC} = \mathsf{EMM}$.

Search. This is a two-party protocol between the client and the server. The client and server execute the FIX.Get protocol where the client's input consists of the key K, the state st and a keyword $w \in \mathbb{W}$, whereas the server's input consists of the encrypted document collection EDC. The client outputs the subset of documents $\mathsf{DC}(w)$ and the server outputs $\perp$.[16]

Insert. This is a two-party protocol between the client and the server. Given a document D, the client builds a tuple $\mathbf{t} = (t_1, \ldots, t_m)$ composed of B-sized blocks, where $m = |D|/B$. The client extracts all keywords $\mathbb{W}_D$ in document D. Then, for each $w \in \mathbb{W}_D$, the client and server execute FIX.Append where the client's input is K, st and an update $u = (\mathsf{app}, w, \mathbf{t})$ and the server's input is the encrypted document collection EDC. Then, for all $i \in [\theta - \#\mathbb{W}_D]$, the client and the server execute FIX.Append where the client's input is K, st and an update $u = (\mathsf{app}, \perp, \{\perp\})$ and the server's input is the encrypted document collection EDC. Finally, the client outputs an updated state st' whereas the server outputs an updated encrypted document collection EDC′.

Delete. The Delete protocol is the same as the Insert protocol except that instead of executing FIX.Append, the client and server execute FIX.Erase.

Efficiency. The communication and computational complexities of FixSSE.Search are the same as the communication and computational complexities of FIX.Get since the former just executes the latter. Once executing the setup protocol and before any update operation, the expected storage complexity is equal to $O((\lambda + \nu) \cdot \#\mathbb{W} \cdot B)$ where B is the block size and ν is the output size of the pseudo-random function in both VLH and DVLH. The size of the stash in the worst-case is at most $\#\mathbb{W}/\mu \cdot |D^\star|$ where $D^\star$ is the largest document stored by FixSSE.[17] The communication and computational complexities of FixSSE.Insert and FixSSE.Delete are θ times the complexities of FIX.Append and FIX.Erase, respectively. Using the same parameters we used to analyze the concrete efficiency of FIX and under the same assumptions, the complexity of FixSSE.Insert and FixSSE.Delete is $O(\theta \cdot \log^2(\#\mathbb{L}))$.

[16] Note that FIX is a black-box construction that, itself, makes use of static and dynamic multi-map encryption schemes. Depending on the correctness guarantees provided by the underlying concrete instantiations, the final output of FIX.Get will vary. For simplicity, we assume here that FIX is instantiated with perfectly correct EMMs.

[17] As shown in Section 4, a more realistic bound can also be derived under the assumption that the documents are sampled from some specific distribution such as the Zipf distribution.

Security. The leakage profile of FixSSE is the same as FIX since all of the former's operations just execute the latter's operations. The only difference is that during Insert and Delete operations θ Append and Erase operations occur but since θ is a public parameter there is no additional leakage. It follows then by Theorem 3.3 that FixSSE is secure against file injection attacks.

References

[1] G. Amjad, S. Kamara, and T. Moataz. Breach-resistant structured encryption. *Proceedings on Privacy Enhancing Technologies*, 2019:245–265, 01 2019.

[2] G. Amjad, S. Patel, G. Persiano, K. Yeo, and M. Yung. Dynamic volume-hiding encrypted multi-maps with applications to searchable encryption. *IACR Cryptol. ePrint Arch.*, page 765, 2021.

[3] G. Asharov, M. Naor, G. Segev, and I. Shahaf. Searchable symmetric encryption: Optimal locality in linear space via two-dimensional balanced allocations. In *ACM Symposium on Theory of Computing (STOC '16)*, STOC '16, pages 1101–1114, New York, NY, USA, 2016. ACM.

[4] G. Asharov, M. Naor, G. Segev, and I. Shahaf. Searchable symmetric encryption: Optimal locality in linear space via two-dimensional balanced allocations. In *ACM on Symposium on Theory of Computing (STOC '16)*, 2016.

[5] G. Asharov, G. Segev, and I. Shahaf. *Tight Tradeoffs in Searchable Symmetric Encryption: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part I*, pages 407–436. 01 2018.

[6] L. Blackstone, S. Kamara, and T. Moataz. Revisiting leakage abuse attacks. *IACR Cryptol. ePrint Arch.*, 2019:1175, 2019.

[7] L. Blackstone, S. Kamara, and T. Moataz. Revisiting leakage abuse attacks. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society, 2020.

[8] R. Bost. Sophos - forward secure searchable encryption. In *ACM Conference on Computer and Communications Security (CCS '16)*, 20016.

[9] R. Bost, B. Minaud, and O. Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. In *ACM Conference on Computer and Communications Security (CCS '17)*, 2017.

[10] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart. Leakage-abuse attacks against searchable encryption. In *ACM Conference on Communications and Computer Security (CCS '15)*, pages 668–679. ACM, 2015.

[11] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *Network and Distributed System Security Symposium (NDSS '14)*, 2014.

Let $\theta, B \in \mathbb{N}$ and $\mathsf{FIX} = (\mathsf{Setup}, \mathsf{Get}_{\mathbf{C},\mathbf{S}}, \mathsf{Append}_{\mathbf{C},\mathbf{S}}, \mathsf{Erase}_{\mathbf{C},\mathbf{S}})$ be the dynamic multi-map encryption scheme as defined in Figures (1) and (2). Consider a searchable symmetric encryption scheme $\mathsf{FixSSE} = (\mathsf{Setup}, \mathsf{Search}_{\mathbf{C},\mathbf{S}}, \mathsf{Insert}_{\mathbf{C},\mathbf{S}}, \mathsf{Delete}_{\mathbf{C},\mathbf{S}})$ as follows:

- $\mathsf{Setup}\big(1^k, \mathsf{DC}\big)$:
  1. initialize an empty multi-map MM;
  2. For each $w \in \mathbb{W}$,
     (a) compute $(D_1, \ldots, D_n) := \mathsf{DC}(w)$;
     (b) set $D^\star := D_1 \| D_2 \| \ldots \| D_n$;
     (c) set $\mathbf{t} := (t_1, \ldots, t_m)$, where $m = |D^\star|/B$ and t_i is the ith B-sized block of $D^\star$;
     (d) set $\mathsf{MM}[w] := \mathbf{t}$;
  3. compute $(K, st, \mathsf{EMM}) \leftarrow \mathsf{FIX.Setup}(1^k, \mathsf{MM})$;
  4. output K, st, and $\mathsf{EDC} := \mathsf{EMM}$.
- $\mathsf{Search}_{\mathbf{C},\mathbf{S}}\big((K, st, w), \mathsf{EDC}\big)$:
  1. $\mathbf{C}$ and $\mathbf{S}$ execute
     $$(\mathbf{v}, \perp) \leftarrow \mathsf{FIX.Get}_{\mathbf{C},\mathbf{S}}\big((K, st, w), \mathsf{EDC}\big);$$
  2. $\mathbf{C}$ outputs $\mathbf{v}$ and $\mathbf{S}$ outputs $\perp$.
- $\mathsf{Insert}_{\mathbf{C},\mathbf{S}}\big((K, st, D), \mathsf{EDC}\big)$:
  1. set $\mathbf{t} := (t_1, \ldots, t_m)$, where $m = |D|/B$ and t_i is the ith B-sized block of D;
  2. for each $w \in \mathbb{W}_D$, $\mathbf{C}$ and $\mathbf{S}$ execute
     $$(st', \mathsf{EDC}') \leftarrow \mathsf{FIX.Append}_{\mathbf{C},\mathbf{S}}\big((K, st, (w, \mathbf{t})), \mathsf{EDC}\big);$$
  3. for all $i \in [\theta - \#\mathbb{W}_D]$, $\mathbf{C}$ and $\mathbf{S}$ execute
     $$(st", \mathsf{EDC}") \leftarrow \mathsf{FIX.Append}_{\mathbf{C},\mathbf{S}}\big((K, st, (\perp, \{\perp\})), \mathsf{EDC}'\big);$$
  4. $\mathbf{C}$ outputs $st = st''$ whereas $\mathbf{S}$ outputs $\mathsf{EDC} = \mathsf{EDC}''$.
- $\mathsf{Delete}_{\mathbf{C},\mathbf{S}}\big((K, st, D), \mathsf{EDC}\big)$:
  1. set $\mathbf{t} := (t_1, \ldots, t_m)$, where $m = |D|/B$ and t_i is the ith B-sized block of D;
  2. for each $w \in D$, $\mathbf{C}$ and $\mathbf{S}$ execute
     $$(st', \mathsf{EDC}') \leftarrow \mathsf{FIX.Erase}_{\mathbf{C},\mathbf{S}}\big((K, st, (w, t)), \mathsf{EDC}\big);$$
  3. for all $i \in [\theta - \#\mathbb{W}_D]$, $\mathbf{C}$ and $\mathbf{S}$ execute
     $$(st', \mathsf{EDC}") \leftarrow \mathsf{FIX.Erase}_{\mathbf{C},\mathbf{S}}\big((K, st, (\perp, \{\perp\})), \mathsf{EDC}'\big);$$
  4. $\mathbf{C}$ outputs $st = st''$ and $\mathbf{S}$ outputs $\mathsf{EDC} = \mathsf{EDC}''$.

Figure 5: The FixSSE searchable symmetric encryption scheme.

[12] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology - CRYPTO '13*. Springer, 2013.

[13] D. Cash, R. Ng, and A. Rivkin. Improved structured encryption for SQL databases via hybrid indexing. In K. Sako and N. O. Tippenhauer, editors, *Applied Cryptography and Network Security - 19th International Conference, ACNS 2021, Kamakura, Japan, June 21-24, 2021, Proceedings, Part II*, volume 12727 of *Lecture Notes in Computer Science*, pages 480–510. Springer, 2021.

[14] D. Cash and S. Tessaro. The locality of searchable symmetric encryption. In *Advances in Cryptology - EUROCRYPT 2014*, 2014.

[15] M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *Advances in Cryptology - ASIACRYPT '10*, volume 6477 of *Lecture Notes in Computer Science*, pages 577–594. Springer, 2010.

[16] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In *ACM Conference on Computer and Communications Security (CCS '06)*, pages 79–88. ACM, 2006.

[17] I. Demertzis, J. G. Chamani, D. Papadopoulos, and C. Papamanthou. Dynamic searchable encryption with small client storage. *IACR Cryptol. ePrint Arch.*, 2019:1227, 2019.

[18] I. Demertzis, D. Papadopoulos, and C. Papamanthou. Searchable encryption with optimal locality: Achieving sublogarithmic read efficiency. *IACR Cryptology ePrint Archive*, 2017:749, 2017.

[19] I. Demertzis, D. Papadopoulos, and C. Papamanthou. Searchable encryption with optimal locality: Achieving sublogarithmic read efficiency. In *Annual International Cryptology Conference*, pages 371–406. Springer, 2018.

[20] I. Demertzis and C. Papamanthou. Fast searchable encryption with tunable locality. In *ACM International Conference on Management of Data (SIGMOD '17)*, SIGMOD '17, pages 1053–1067, New York, NY, USA, 2017. ACM.

[21] D. P. Dubhashi and D. Ranjan. Balls and bins: A study in negative dependence. *BRICS Report Series*, 3(25), 1996.

[22] M. Etemad, A. Küpçü, C. Papamanthou, and D. Evans. Efficient dynamic searchable encryption with forward privacy. *PoPETs '18, Issue 1*, 2018.

[23] M. Etemad, A. Küpçü, C. Papamanthou, and D. Evans. Efficient dynamic searchable encryption with forward privacy. *Proceedings on Privacy Enhancing Technologies*, 2018(1):5–20, 2018.

[24] S. Faber, S. Jarecki, H. Krawczyk, Q. Nguyen, M. Rosu, and M. Steiner. Rich queries on encrypted data: Beyond exact matches. In *European Symposium on Research in Computer Security (ESORICS '15). Lecture Notes in Computer Science*, volume 9327, pages 123–145, 2015.

[25] J. Ghareh Chamani, D. Papadopoulos, C. Papamanthou, and R. Jalili. New constructions for forward and backward private symmetric searchable encryption. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 1038–1055, New York, NY, USA, 2018. ACM.

[26] F. Hahn and F. Kerschbaum. Searchable encryption with secure and efficient updates. In *ACM Conference on Computer and Communications Security (CCS '14)*, CCS '14, pages 310–320, New York, NY, USA, 2014. ACM.

[27] M. S. Islam, M. Kuzu, and M. Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *Network and Distributed System Security Symposium (NDSS '12)*, 2012.

[28] S. Kamara and T. Moataz. SQL on Structurally-Encrypted Data. Technical Report 2016/453, IACR ePrint Cryptography Archive, 2016.

[29] S. Kamara and T. Moataz. SQL on structurally-encrypted databases. *IACR Cryptology ePrint Archive*, 2016:453, 2016.

[30] S. Kamara and T. Moataz. Boolean searchable symmetric encryption with worst-case sublinear complexity. In *Advances in Cryptology - EUROCRYPT '17*, 2017.

[31] S. Kamara and T. Moataz. Computationally volume-hiding structured encryption. In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II*, pages 183–213, 2019.

[32] S. Kamara, T. Moataz, and O. Ohrimenko. Structured encryption and leakage suppression. In H. Shacham and A. Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018*, pages 339–370, Cham, 2018. Springer International Publishing.

[33] S. Kamara and C. Papamanthou. Parallel and dynamic searchable symmetric encryption. In *Financial Cryptography and Data Security (FC '13)*, 2013.

[34] S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *ACM Conference on Computer and Communications Security (CCS '12)*. ACM Press, 2012.

[35] J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. Chapman & Hall/CRC, 2008.

[36] K. S. Kim, M. Kim, D. Lee, J. H. Park, and W.-H. Kim. Forward secure dynamic searchable symmetric encryption with efficient updates. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1449–1463, 2017.

[37] I. Miers and P. Mohassel. Io-dsse: Scaling dynamic searchable encryption to millions of indexes by improving locality. Cryptology ePrint Archive, Report 2016/830, 2016. http://eprint.iacr.org/2016/830.

[38] V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S.-G. Choi, W. George, A. Keromytis, and S. Bellovin. Blind seer: A scalable private dbms. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 359–374. IEEE, 2014.

[39] S. Patel, G. Persiano, and K. Yeo. Leakage cell probe model: Lower bounds for key-equality mitigation in encrypted multi-maps. Cryptology ePrint Archive, Report 2019/1132, 2019. https://eprint.iacr.org/2019/1132.

[40] 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 *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, pages 79–93, New York, NY, USA, 2019. Association for Computing Machinery.

[41] S. Patranabis and D. Mukhopadhyay. Forward and backward private conjunctive searchable symmetric encryption. In *NDSS Symposium 2021 (virtual)*, 2021.

[42] D. Song, D. Wagner, and A. Perrig. Practical techniques for searching on encrypted data. In *IEEE Symposium on Research in Security and Privacy*, pages 44–55. IEEE Computer Society, 2000.

[43] E. Stefanov, C. Papamanthou, and E. Shi. Practical dynamic searchable encryption with small leakage. In *Network and Distributed System Security Symposium (NDSS '14)*, 2014.

[44] 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*, 2016.