Mitigating Leakage in Secure Cloud-Hosted Data Structures: Volume-Hiding for Multi-Maps via Hashing

Sarvar Patel[*] Giuseppe Persiano[†] Kevin Yeo[‡] Moti Yung[§]

Abstract

Volume leakage has recently been identified as a major threat to the security of cryptographic cloud-based data structures by Kellaris *et al.* [CCS'16] (see also the attacks in Grubbs *et al.* [CCS'18] and Lacharité *et al.* [S&P'18]). In this work, we focus on volume-hiding implementations of *encrypted multi-maps* as first considered by Kamara and Moataz [Eurocrypt'19]. Encrypted multi-maps consist of outsourcing the storage of a multi-map to an untrusted server, such as a cloud storage system, while maintaining the ability to perform private queries. Volume-hiding encrypted multi-maps ensure that the number of responses (volume) for any query remains hidden from the adversarial server. As a result, volume-hiding schemes can prevent leakage attacks that leverage the adversary's knowledge of the number of query responses to compromise privacy.

We present both conceptual and algorithmic contributions towards volume-hiding encrypted multi-maps. We introduce the first formal definition of *volume-hiding leakage functions*. In terms of design, we present the first *volume-hiding encrypted multi-map* dprfMM whose storage and query complexity are both asymptotically optimal. Furthermore, we experimentally show that our construction is practically efficient. Our server storage is smaller than the best previous construction while we improve query complexity by a factor of 10-16x.

In addition, we introduce the notion of *differentially private volume-hiding leakage functions* which strikes a better, tunable balance between privacy and efficiency. To accompany our new notion, we present a *differentially private volume-hiding encrypted multi-map* dpMM whose query complexity is the volume of the queried key plus an additional logarithmic factor. This is a significant improvement compared to all previous volume-hiding schemes whose query overhead was the maximum volume of any key. In natural settings, our construction improves the average query overhead by a factor of 150-240x over the previous best volume-hiding construction even when considering small privacy budget of $\epsilon = 0.2$.

1 Introduction

In this paper, we study *structured encryption* (STE), first introduced by Chase and Kamara [CK10], which is a cryptographic primitive used to study the security of cloud-hosted data structures. Structured encryption schemes enable the owner of a data structure to encrypt the data structure and outsource the storage of encrypted data structure to a potentially untrusted third-party such as a cloud storage system. Additionally, STE schemes allow the data owner to perform data structure operations on the outsourced encrypted data structure without revealing any information to the server beyond some well-defined and "sensible" leakage function.

An important example of a STE scheme is the *encrypted multi-map* (EMM) [CGKO11, KM19] primitive which enables the storage of keys associated to a sequence of (possibly) multiple values. Furthermore, multi-maps allows its owner to query for a key and receive all values associated with the key. EMM's form the basis of many important applications. Two such applications are searching over a corpus of encrypted documents

[*]`sarvar@google.com`. Google LLC.
[†]`giuper@gmail.com`. Università di Salerno.
[‡]`kwlyeo@google.com`. Google LLC.
[§]`moti@google.com`. Google LLC and Columbia University.

and performing queries over encrypted relational databases. As a result, the construction of both efficient and private encrypted multi-maps is a very important problem. We now explain these applications in more detail.

Searchable encryption is a primitive first introduced by Song *et al.* [SWP00] and has continued to be well-studied in the past decade. One can see some of the following as examples of works in searchable encryption [Goh03, BDCOP04, BBO07, CGKO11, KPR12, CJJ$^+$13, CJJ$^+$14, CT14, NPG14, SPS14, Bos16, ANSS16, MM16, BMO17, PPY17, DP17, KM17, ASS18, AKM18, DPP18]. Searchable encryption enables a data owner with a corpus of documents to encrypt and store the corpus to an untrusted third-party while maintaining the ability to privately search for documents containing specified keywords. In many cases, searchable encryption uses EMMs as the main underlying primitive to map keywords to documents that contain keywords. Although, EMMs have also been used in many other ways.

Encrypted databases are another important application of EMMs. The goal is to encrypt databases whose storage will be outsourced to an untrusted third party while enabling the data owner to privately perform database operations. We note that earlier attempts [PRZB11] at encrypted databases used property-preserving encryption schemes such as deterministic [BBO07] and order-preserving [BCLO09, BCO11] encryption schemes. Work by Naveed *et al.* [NKW15] show that encrypted databases built from property-preserving encryption have security vulnerabilities. Recently, Kamara and Moataz [KM18] present an encrypted SQL database scheme that foregoes the use of property-preserving encryption. Instead, they rely on EMMs to perform SQL operations privately.

As a result of the above important applications, it is clear that constructing EMMs that are both efficient and private is important. While efficiency is clear to evaluate, assessing the privacy of an EMM (and STE schemes in general) is a challenging problem. So far, our only measure of privacy is a "sensible" or "reasonable" leakage function, which is both a vague and subjective qualifier. There has been a lot of work that attempt to understand various leakage profiles and determine whether certain leakage profiles may be abused to compromise privacy. The first leakage-abuse attacks was presented by Islam *et al.* [IKK12]. Many follow up works [CGPR15, NKW15, GMN$^+$16, KKNO16, PW16, ZKP16, GSB$^+$17, LMP18, GLMP] consider either different leakage profiles and/or weaker assumptions. These attacks significantly further our understanding of the dangers of various types of leakage profiles. Furthermore, these attacks present guidance on the necessary requirements of private STE schemes. Therefore, an important line of research is to construct STE schemes with smaller leakage that protect against these attacks.

In an attempt to mitigate the risk of leakage profiles, the seminal work of Kamara and Moataz [KM19] introduce the notion of volume-hiding EMM schemes. These schemes ensure that the number of values (volume) associated with a single key is never leaked to the adversary. Several of the above leakage-abuse attacks rely on the knowledge of volume to compromise privacy. As a result, volume-hiding schemes can foil such attacks. In this work, we continue the study of volume-hiding constructions by presenting schemes with better query and storage efficiency.

1.1 Our Contributions

In this work, we make both conceptual and algorithmic contributions in the area of volume-hiding EMMs. In particular, we present formal definitions for volume-hiding EMMs and introduce the notion of differentially private volume-hiding EMMs. Furthermore, we present efficient constructions for both types of EMMs.

Throughout this section, we will phrase the efficiency improvements of our constructions using multipliers. As an example, a 2x improvement in communications means that our new construction uses half the communication of the previous best construction.

Volume-Hiding EMMs. We start by describing our contributions to volume-hiding encryption schemes for multi-maps. To our knowledge, we present the first formal security definition for volume-hiding leakage functions. Our security definition is built on top of the typical simulation-based security definitions for STE schemes allowing the proof techniques from STE schemes to also be used for volume-hiding STE schemes. Accompanying our conceptual definitional contributions, we present the following efficient, volume-hiding STE scheme for general multi-maps:

Theorem 1 (Informal)**.** *Consider any multi-map* MM *with* n *total values and let* ℓ *be the maximum number of values associated with a key of* MM*. Then there exists a volume-hiding encrypted multi-map with communication complexity of* $O(\ell)$*, server storage of* $O(n)$ *and client storage of size* $f(\lambda)$*, for every function* $f(\lambda) = \omega(1)$*. The leakage of the encrypted multi-map consists of only the query equality pattern and the values* ℓ *and* n*.*

Our construction is lossless (that is, it always returns all the values associated with a key) and is asymptotically optimal in terms of storage and communication complexity. Indeed, in a losseless construction each of the n values must be stored at least once and our construction results in server storage of about $2n$. More precisely, it can be instantiated to use $(2 + \alpha)n$ storage, for every constant $\alpha > 0$. Moreover, due to volume-hiding, it must be that each query returns at least ℓ records and our construction will always return exactly 2ℓ records. Finally, the client storage typically must include at least a private key. Our construction requires storing a private key in addition to just $\omega(1)$ values. We show experimentally that our volume-hiding scheme is concretely efficient and improves on previous best volume-hiding scheme. In particular, our scheme uses less server storage and improves query overhead by a factor of 10-16x over [KM19] when encrypting multi-maps of size 1-67 MB in the plaintext and consisting of 2^{16}-2^{22} total values.

Differentially Private, Volume-Hiding EMMs. As another contribution, we directly address a statement appearing in [KM19]: "*it is hard to imagine any non-lossy construction being able to hide response length of a query and have query complexity* $o(t)$*, where* t *is the maximum response length.*" We introduce the notion of a *differentially private volume-hiding* leakage function as a weakening of volume-hiding. Kellaris et al. [KKNO17] studied the related notion of differentially private volume-hiding for ORAM.

We present the following differentially private, volume-hiding encryption scheme whose query complexity for many keys are much smaller than the maximum volume:

Theorem 2 (Informal)**.** *Consider any multi-map* MM *with* n *total values and let, for every* $\texttt{key}$*,* $\ell(\texttt{key})$ *denote the number of values, the* volume*, associated with* $\texttt{key}$*. Then there exists an encrypted multi-map with communication complexity* $O(\ell(\texttt{key}) + f(\lambda))$*, for any function* $f(\lambda) = \omega(\log \lambda)$*, with server storage of* $O(n)$*, and client storage of size* $g(\lambda)$*, for any function* $g(\lambda) = \omega(1)$*. The scheme is lossless except with probability negligible in* λ*. The leakage of the encrypted multi-map consists of only query equality pattern, the number of total values* n *as well as differentially private leakage of volumes of the queried keys.*

We note that, unlike volume-hiding schemes, our differentially private scheme is able to achieve query complexity that is only dependent on the volume of the queried key. In particular, the query complexity will be at most the volume of the query key plus an additional logarithmic factor. In typical scenarios, most keys have smaller volume than the maximum volume. Using experiments, we show that our schemes can reduce query overhead by a factor of 150-240x over the previous best volume-hiding scheme [KM19] when encrypting multi-maps consisting of 2^{16}-2^{22} values and occupy 1-67 MB in the plaintext.

By defining our new security notion of differential privacy volume-hiding, we attempt to strike a tunable balance between efficiency and privacy. The previous notion of volume-hiding considered strong privacy that resulted in large query complexity. When defending future, feasible attacks, volume-hiding might be too strong resulting in unnecessary query overhead. We thus look at the differential privacy framework to obtain a more efficient protocol at the cost of weakening the security notion in a meaningful way. We observe that all previous volume attacks on STE schemes [IKK12, NKW15, KKNO16, GLMP18, LMP18] are generic and do not exploit weaknesses of specific constructions. Rather, they are attacks to the ideal world functionality. Therefore, our generic framework will provide protection against all of these, as well as future, attacks to ideal world functionalities.

1.2 Comparison to Previous Works

In this section, we compare our new constructions with those that were presented in the work on volume-hiding EMMs by Kamara and Moataz [KM19] as well as some naive approaches. To do this, we briefly review their EMM constructions and compare them to our work. For a more detailed exposition on these constructions, we refer the reader to [KM19]. We summarize our comparisons in Table 1. All shown

	Query Complexity	Server Storage for General MMs	Lossy with Non-Negligible Probability	Server Storage for Concentrated MMs	Computational Assumption
Naive Padding	$\Theta(\ell)$	$\Theta(m \cdot \ell)$		$\Theta(m \cdot \ell)$	One-Way Functions
Pseudorandom Transform [KM19]	$\Theta(\ell)$	$\Theta(m \cdot \ell)$	✓	$\Theta(m \cdot \ell)$	One-Way Functions
Dense Subgraph Transform [KM19]	$\Theta(\ell \cdot \log n)$	$\Theta(n)$		$\Theta(n)$	One-Way Functions
Dense Subgraph and Planting Transform [KM19]	$\Theta\left(\ell \cdot \frac{n}{\texttt{polylog}(m)}\right)$	$\Theta(n)$		$\Theta(n - \sqrt{m} \cdot \texttt{polylog}(m))$	Planted Densest Subgraph Problem
Ours (dprfMM)	2ℓ	$(2+\alpha)n$		$(2+\alpha)n$	One-Way Functions
Ours (dpMM)	$2\ell(k) + \omega(\log \lambda)$	$(2+\alpha)(m+n)$		$(2+\alpha)(m+n)$	One-Way Functions

Table 1: This table compares previous volume-hiding encryption schemes for multi-maps from [KM19] with the constructions of this paper. For notation, ℓ represent the volume of the largest key of the input multi-map, m denotes the number of unique keys and n denotes the total number of values over all keys. For any queried key k, we denote by $\ell(k)$ as the number of values associated with the queried key k. The value α may be any positive constant. Finally, we refer to $2^{-\lambda}$ as the probability of losing data in our differentially private solution.

efficiencies are taken from the suggested concrete parameters sections of [KM19]. We denote the number of keys by m, the total number of values by n, and the maximum number of values associated with a key by ℓ.

Naive Padding. The simplest approach to volume-hiding encryption schemes is naive padding. Given any multi-map, a new multi-map can be constructed where each key has ℓ values. For keys with less than ℓ values, additional dummy values are padded until there are ℓ values. We note that the resulting storage is $m \cdot \ell$. In many cases, $(m \cdot \ell) \gg n$ where n is the number of total values. For example, when $m = n/2$ and $\ell = n/2$, the storage of naive padding is $m \cdot \ell = \Theta(n^2)$ which is quadratically larger than the number of total values. Additionally, naive padding results in larger overhead if the multi-map follows naturally appearing distributions such as Zipf's distribution. In practice, plaintext multi-maps typically use storage on the order of the number total values n. Therefore, naive padding uses too much storage in almost all cases and our storage goal for volume-hiding EMMs should be close to n.

ORAMs. Another naive approach is to use oblivious RAMs (ORAMs), which were introduced by Goldreich and Ostrovsky [GO96]. ORAMs are a powerful primitive that enable access to storage hosted by a potentially untrusted server such as a cloud storage provider without leaking any information beyond the number of accesses. As a result, there has been a lot of work in ORAM [GM11, KLO12, SvDS$^+$13, PPRY18, AKL$^+$18] leading to logarithmic overhead constructions. Furthermore, ORAM lower bounds [LN18, PY19] have shown that these are the best possible constructions. A simple approach is to take any STE scheme and replace each access using an ORAM access to suppress leakage. Furthermore, one can make an STE scheme volume-hiding by making fake ORAM accesses until ℓ records have been retrieved. The resulting scheme would have $\Theta(\ell \cdot \log n)$ overhead using the best theoretical constructions and $\Theta(\ell \cdot \log^2 n)$ using the more concretely efficient ORAM constructions. In either case, the overhead of ORAM is too large for practical use cases.

Pseudorandom Transform. From a high level, the pseudorandom transform [KM19] takes an input multi-map and generates a new multi-map such that the number of values associated with each key is generated using a pseudorandom function. As a result, there may exist keys whose volume in the new multi-map is smaller compared to the input multi-map. Therefore, the construction is lossy as truncation occurs and several values are removed from the new multi-map. The authors show that the number of truncated keys is small when the input multi-map is Zipf's distributed. Furthermore, the authors of [KM19] show that the storage overhead is $(m \cdot \ell)/2$ which is better than naive padding. Unfortunately, this transform is not practical as there are no guarantees for data loss (truncation) on general multi-maps and the storage overhead is closer to naive padding of $m \cdot \ell$ than our goal of n.

Densest Subgraph Transform for General Multi-Maps. This transform considers a bipartite graph where each of the m keys are in one part and there are b empty bins in the other part. Each of the m keys are assigned ℓ bins chosen uniformly at random. Each of the, at most ℓ, values associated with a key is placed into one of the key's assigned bin such that at most one value appears in each bin. After this is done for all keys, the bins are padded with dummy values to the size of the maximum bin. By balls-and-bins analysis, the size of the bins must be $\Omega(\log n)$ size. As a result, the query overhead becomes at least $\Theta(\ell \cdot \log n)$. On the other hand, the storage overhead becomes $\Theta(n)$. We note that this construction is strictly worse in query overhead than our construction as it requires an extra $\log n$ overhead while achieving the same asymptotic storage overhead of $\Theta(n)$. Furthermore, we experimentally show that our constructions achieve 10-16x query overhead improvements.

Densest Subgraph Transform for Concentrated Multi-Maps. The last construction of [KM19] considered concentrated multi-maps. That is, multi-maps where a large number of keys share many same values. In this case, they modify the previous construction such that this concentrated set of values will appear only once in a single set of bins. As a result, the storage overhead becomes $\Theta(n - \sqrt{m} \cdot \texttt{polylog}(m))$. Unfortunately, this construction requires assuming that the planted densest subgraph problem is hard, which has been not heavily studied. Due to the lack of cryptanalysis, it is very difficult to find concrete parameters that can be used in practice where security can be trusted. Furthermore, the resulting query complexity is very large. As a result, this construction does not seem to be usable in practice at the moment.

Final Comparison. To make a final comparison, it is clear that our volume-hiding construction is more concretely efficient and practical compared to all previous works. In particular, it achieves the best storage overhead of just $(2 + \alpha)$ times the number of total values. Additionally, the query overhead is only 2ℓ. In contrast, the schemes from [KM19] either might lose a lot of information for general multi-maps and have large storage overhead (pseudorandom transform), require computational assumptions that have not been well studied (densest subgraph for concentrated multi-maps) or require a much larger query overhead of $\Theta(\ell \cdot \log n)$ (densest subgraph for general multi-maps) than our construction. Furthermore, our differentially private volume-hiding construction uses slightly more server storage while enabling significantly smaller query overhead.

1.3 Our Techniques

In this section, we outline our techniques that enable us to construct better volume-hiding, encryption schemes for multi-maps. We start by considering a simple construction. Consider any key k and its associated values $v_1, \ldots, v_{\ell(k)}$ where $\ell(k)$ is the number of values associated with key k. The server stores a dictionary in the following manner. The value v_i will appear at location $F_K(k \mid\mid i)$ in the dictionary where F is a pseudorandom function (PRF), K is the key of the PRF and $k \mid\mid i$ is the concatenation of the key k to the index i. Assuming the output range of the PRF F is large (like $\Omega(n^2)$ where n is the total number of values), there would be no collisions in the dictionary. To query for a key k, the client would simply send the values $F_K(k \mid\mid 1), \ldots, F_K(k \mid\mid \ell(k))$ and the server would return the associated encrypted values from the dictionary.

Going towards the problem of volume-hiding, it is not clear how to modify this EMM construction to hide the volumes of all the keys. The first naive approach would be to send ℓ PRF values for any key where ℓ is the maximum value of any key. Unfortunately, this does not work. Consider a key k where $\ell(k) < \ell$. If the client sends the values $F_K(k \mid\mid 1), \ldots, F_K(k \mid\mid \ell)$, the server will see that the values $F_K(k \mid\mid \ell(k)+1), \ldots, F_K(k \mid\mid \ell)$ do not exist in the server-held dictionary. As a result, the server can quickly determine the volume of key k is exactly $\ell(k)$. One way to mitigate this leakage is to simply populate the dictionary with the missing values for all keys k. However, this is similar to the naive padding approach which would result in $m \cdot \ell$ storage which is much larger than our goal of n (the number of total values) in most cases.

Therefore, we seem to have contrasting problems now. We wish to make sure that all PRF values for those with no value, $F_K(k \mid\mid \ell(k) + 1), \ldots, F_K(k \mid\mid \ell)$, point to some non-empty entry in the dictionary. On the other hand, we do not want to simply insert dummy values in for each of the PRF values and increase server storage. The main idea to overcome these problems is to reduce the output space of the PRF F. For example, we could attempt to reduce the output range of the PRF F to be $\Theta(n)$. Now, the problem is to place each of the n values into a location that is determined by F. The remaining empty locations can be simply padded with dummy values without, hopefully, incurring a huge storage overhead like before. This is the exact problem that is considered in *hashing* where the goal is to place n items into a space of $\Theta(n)$ locations such that each item can be easily retrieved by looking into only a small number of locations specified by a PRF F. One approach to this problem would be to simply place a value into its location specified by the PRF F. Each location will be padded to the size of the largest bin. The classical balls-and-bins analysis shows the size of the largest will contain $\Omega(\log n)$ items. To query, a key will be download its ℓ associated bins which would incur an $\Omega(\ell \cdot \log n)$ query complexity. We note an approach similar to this idea is used by the densest subgraph transform in [KM19].

To avoid the additional logarithmic overhead, we will use *cuckoo hashing* introduced by Pagh and Rodler [PR04] for our constructions. In particular, we will use a variant presented by Kirsch *et al.* [KMW09] where an additional stash exists to exponentially decrease the probability of failure. From a high level, cuckoo hashing consists of two tables of size $(1+\alpha)n$ for a small constant $0 < \alpha < 1$. For value v_i associated with key k, the value v_i will be placed into one of the three locations: $F_K(k \mid\mid i \mid\mid 0)$ in table one or $F_K(k \mid\mid i \mid\mid 1)$ in table two or in the stash. The stash will be stored by the client. The remaining empty locations in each table will be filled by dummy values and all values in both tables will be encrypted. Let us now reconsider a query for a key k where $\ell(k) < k$. The client can send the values $\{F_K(k \mid\mid i \mid\mid 0), F_K(k \mid\mid i \mid\mid 1)\}_{i\in[\ell]}$ and the server will return all 2ℓ encrypted values in the tables. We prove that this construction is volume-hiding as the server will be unaware whether a location is filled with a real or dummy value.

Additionally, we present a concrete optimization that further improves the communication costs of our volume-hiding encryption scheme for multi-maps. In the above scheme, the client must send the output of 2ℓ PRF evaluations. However, the PRF evaluations may be easily arranged to correspond to a consecutive set of inputs. As a result, we are able to use delegatable PRFs [GGM86, KPTZ13] where the 2ℓ PRF evaluations can be sent using smaller communication. In our specific application, the client will always be sending the prefix of exactly 2ℓ evaluations. We are able to convey these 2ℓ PRF evaluations using exactly one PRF evaluation in such a way that the server can securely expand the single PRF evaluation to the required 2ℓ PRF evaluations. This optimization reduces the bandwidth from the client to the server from 2ℓ PRF outputs to just one PRF output.

Finally, we describe our modifications to construct a differentially private volume-hiding EMM. A first attempt at differentially private volume-hiding would be to send $\ell(k) + X$ PRF evaluations for any key k with $\ell(k)$ values and X is drawn from the Laplacian distribution. The problem is that $\mathsf{Lap}(1/\epsilon)$ may be negative resulting in a lossy construction. Instead, we will pick some public parameter $f(\lambda)$ such that the probability that X drawn from the Laplacian is smaller than $f(\lambda)$ occurs with probability at most $2^{-\lambda}$. By picking large enough λ and $f(\lambda) = \omega(\log \lambda)$, we can guarantee that our construction is non-lossy except with small probability that is not observable in practice. Note, the query complexity of our differentially private volume-hiding EMM is $\ell(k) + \omega(\log \lambda)$ which is, on average, significantly smaller than the largest volume ℓ. The only caveat is that we now need to know the volume of any queried key. To do this, we will store a count table which, for each key k with positive volume, stores the volume of the key $\ell(k)$. The count table will be stored on the server using cuckoo hashing.

2 Definitions

Structured encryption schemes considers the problem of encrypting a data structure. Additionally, the encrypted data structure should enable the original data owner to be able to perform queries in a private manner when the encrypted data structure is held by a potentially untrusted third party server. The first definitions of structured encryption were presented by Chase and Kamara [CK10]. While we can consider generic definitions for encrypting any data structure, we focus our definitions on encrypting multi-maps exclusively.

2.1 Encrypted Multi-Maps

We start by considering the multi-map (MM) primitive, which maintains a set of m key to value vector pairs $\mathsf{MM} = \{(\mathtt{key}_i, \vec{\mathsf{v}}_i)\}_{i \in [m]}$ where each $\mathtt{key}_i$ is from the *key universe* $\mathcal{K}$ and $\vec{\mathsf{v}}_i$ is a tuples (or vector) of values from the *value universe* $\mathcal{V}$. For convenience, we suppose that all keys are unique. That is, $\mathtt{key}_i \neq \mathtt{key}_j$ when $i \neq j$. Note, this assumption is without loss of generality as if there are two equal keys $\mathtt{key}_i = \mathtt{key}_j$, then the two tuples of values can be combined into a single tuple of values. For any $\mathtt{key}_i$, we denote by the number of values associated with $\mathtt{key}_i$ by $\ell(\mathtt{key}_i)$ (that is, $\ell(\mathtt{key}_i) := |\vec{\mathsf{v}}_i|$). Note, the number of values associated with different keys can be different. We denote the maximum number of values associated by ℓ (that is, $\ell := \max_{i \in [m]} \ell(\mathtt{key}_i) = \max_{i \in [m]} |\vec{\mathsf{v}}_i|$). Note, we will refer to ℓ as the *width* of the multi-map. We denote the total number of values by $n := \sum_{i \in [m]} \ell(\mathtt{key}_i) = \sum_{i \in [m]} |\vec{\mathsf{v}}_i|$.

In terms of functionality, a multi-map provides a query operation which takes as input $\mathtt{key}$ from the key universe $\mathcal{K}$. The output of the query operation with input $\mathtt{key}$ for a multi-map MM will be the tuple of values associated with the $\mathtt{key}$ stored in MM. For example, if $\mathtt{key} = \mathtt{key}_i$ for some $i \in [m]$, then the query operation will return the tuples $\vec{\mathsf{v}}_i$. On the other hand, if $\mathtt{key} \neq \mathtt{key}_i$ for all $i \in [m]$, then the output of the query operation will be $\perp$.

Moving towards encrypted multi-maps, we now present a formal definition of structured encryption for multi-maps. Our STE definition will consider query algorithms with r rounds of interaction where the data structure must correctly answer to queries using r rounds of communication between the client and the server. However, we will only consider non-interactive ($r = 1$) or two-rounds of interaction ($r = 2$) in this paper. We focus on schemes with few rounds of interaction due to their practicality and efficiency as they only use a small number of roundtrips of communication between the client and server per query.

Definition 1 (r-Interactive, Structured Encryption for MMs)**.** *A r-interactive structured encryption scheme for multi-maps $\Sigma_{\mathsf{MM}} = (\mathsf{Setup}, \mathsf{Query}, \mathsf{Reply}, \mathsf{Result})$ where $r \geq 1$ consists of the $r + 2$ following polynomial time algorithms:*

1. $(K, \mathsf{EMM}) \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{MM})$ *is an algorithm executed by the client $\mathbb{C}$ that takes as input the security parameter λ as well as the input multi-map* MM *. The output of* Setup *is a private key K as well as the encrypted multi-map* EMM*. The private key K will be held by the client $\mathbb{C}$ while the encrypted multi-map* EMM *is sent to the server $\mathbb{S}$.*

2. *For $i \in \{0, \dots, r-1\}$, we define the query algorithm as* $\mathsf{Request}_i \leftarrow \mathsf{Query}^i(K, \mathtt{key}, \mathsf{Response}_{i-1})$ *that is executed by the client $\mathbb{C}$ that takes as input the private key K, the* $\mathtt{key}$ *from the key universe $\mathcal{K}$ as well as the server's response,* $\mathsf{Response}_{i-1}$ *from the previous query. For the first query ($i = 0$),* $\mathsf{Response}_{-1}$ *will be $\perp$. The output is a search request* $\mathsf{Request}_i$ *which will be sent to the server $\mathbb{S}$.*

3. *For $i \in \{0, \dots, r-1\}$, we define the reply algorithm as* $\mathsf{Response}_i \leftarrow \mathsf{Reply}^i(\mathsf{Request}_i, \mathsf{EMM})$ *that is executed by the server $\mathbb{S}$ that takes as input* $\mathsf{Request}_i$ *and the encrypted multi-map* EMM*. The output of* Reply *will be* $\mathsf{Response}_i$ *which is the server's response for the i-th query.*

4. $\vec{\mathsf{v}} \leftarrow \mathsf{Result}(K, \mathsf{Response}_{r-1})$ *is an algorithm executed by the client $\mathbb{C}$ that takes as input* $\mathsf{Response}_{r-1}$ *as well as the private key K. The output will be the tuple of values $\vec{\mathsf{v}}$ associated with* $\mathtt{key}$ *used to generate* $\mathsf{Request}_0, \dots, \mathsf{Request}_{r-1}$ *by the client $\mathbb{C}$.*

For non-interactive encryption schemes with $r = 1$, we will simply drop all subscripts and superscripts related to Query and Reply as there will only be one algorithm for each.

2.2 Adaptive Security and Leakage Functions

The security notion of structured encryption for multi-maps is parameterized by two leakage functions, $\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Query}}$. Leakage function $\mathcal{L}_{\mathsf{Setup}}$ gives an upper bound on the information leaked by the encrypted multi-map EMM generated by Setup. Leakage function $\mathcal{L}_{\mathsf{Query}}$ instead bounds the information leaked by the request generated by the client using Query. Note that of the algorithms used to perform a query (that is,

$\mathsf{Query}, \mathsf{Reply}$ and Result) Query is the only one that can potentially leak information to the server $\mathbb{S}$ through its output $\mathsf{Request}$. We next formally present the security notion using the simulation-based game definitions. In particular, we will consider the adaptive variant where the query operations may be chosen adaptively by the adversary after seeing the leakage from previous queries.

Let $\Sigma = (\mathsf{Setup}, \{\mathsf{Query}^i, \mathsf{Reply}^i\}_{i=0,\dots,r-1}, \mathsf{Result})$ be a structured encryption scheme for multi-maps with r rounds of interaction. To define adaptive security, we consider the following real and ideal experiments where $\mathcal{A}$ is a stateful, honest-but-curious PPT adversary, $\mathcal{S} = (\mathsf{SimSetup}, \mathsf{SimQuery})$ is a stateful, PPT simulator and $\mathcal{L}_{\mathsf{Setup}}$ and $\mathcal{L}_{\mathsf{Query}}$ are the leakage functions for the setup and query process respectively.

$\mathbf{Real}_{\Sigma,\mathcal{A}}(1^\lambda)$:

1. The adversary $\mathcal{A}$ selects an input multi-map MM and gives it to the challenger.
2. The challenger $\mathcal{C}$ executes $(K, \mathsf{EMM}) \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{MM})$ and sends EMM to the adversary $\mathcal{A}$.
3. The adversary $\mathcal{A}$ will adaptively pick a polynomial number of queries $\mathtt{key}_1, \dots, \mathtt{key}_{\mathtt{poly}(\lambda)}$. For each query $\mathtt{key}_j$, the challenger $\mathcal{C}$ executes the $\{\mathsf{Query}^i, \mathsf{Reply}^i\}_{i=0,\dots,r-1}$ with $\mathcal{A}$. $\mathcal{A}$ receives $\mathsf{Request}_i \leftarrow \mathsf{Query}^i(K, \mathtt{key}_j, \mathsf{Response}_{i-1})$ for all i and $\mathtt{key}_j$.
4. Finally, the adversary $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$.

$\mathbf{Ideal}_{\Sigma,\mathcal{A},\mathcal{S}}(1^\lambda)$:

1. The adversary $\mathcal{A}$ generates an input multi-map MM.
2. The simulator receives $\mathcal{L}_{\mathsf{Setup}}(\mathsf{MM})$ and returns an encrypted multi-map EMM.
3. The adversary $\mathcal{A}$ will adaptively pick a polynomial number of queries $\mathtt{key}_1, \dots, \mathtt{key}_{\mathtt{poly}(\lambda)}$.
   For each query $\mathtt{key}_j$, the simulator $\mathcal{S}$ computes $\mathsf{Request}_i$ using only $\mathcal{L}_{\mathsf{Query}}(\mathtt{key}_1, \dots, \mathtt{key}_i, \mathsf{MM})$ as input, for all $i \in \{0, \dots, r-1\}$. $\mathsf{Request}_i$ is forwarded to the adversary $\mathcal{A}$.
4. Finally, the adversary $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$.

Definition 2 (Adaptive Security for r-Interactive, Structured Encryption for MMs)**.** *The r-interactive, structured encryption scheme for multi-maps Σ is* adaptively $(\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Query}})$-secure *if there exists a stateful, PPT simulator $\mathcal{S}$ such that for all PPT adversaries $\mathcal{A}$:*

$$|\Pr[\mathbf{Real}_{\Sigma,\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathbf{Ideal}_{\Sigma,\mathcal{A},\mathcal{S}}(1^\lambda) = 1]| \le \mathtt{negl}(\lambda).$$

2.3 Typical Leakage Functions

In this section, we describe typical leakage function that have been considered for structured encryption for multi-maps. We will follow the terminology introduced by Kamara *et al.* [KMO18].

- **Query Equality**: This leakage reports the equality pattern leaking whether two queries are to the same key or not. Formally, for a sequence of queried keys $\mathtt{key}_1, \dots, \mathtt{key}_t$, $\mathsf{qeq}(\mathtt{key}_1, \dots, \mathtt{key}_t) = M$ consists of a $t \times t$ matrix such that $M[i][j] = 1$ if and only if $\mathtt{key}_i = \mathtt{key}_j$.
- **Response Length**: This leakage consists of the number of values (volume) associated with queried keys. Formally, for a sequence of queried keys $\mathtt{key}_1, \dots, \mathtt{key}_t$ and a multi-map MM, That is, $\mathsf{rlen}(\mathsf{MM}, \mathtt{key}_1, \dots, \mathtt{key}_t) = (|\mathsf{MM}[\mathtt{key}_i]|)_{i \in [t]}$ where $\mathsf{MM}[\mathtt{key}_i]$ refers to the tuple of values associated with $\mathtt{key}_i$.
- **Maximum Response Length**: This leakage pattern consists of the maximum number of values associated with any key in the multi-map. Formally, for any multi-map MM, $\mathsf{mrlen}(\mathsf{MM}) = \ell = \max_{\mathtt{key} \in \mathcal{K}} |\mathsf{MM}[\mathtt{key}]|$.

- **Domain Size**: This leakage pattern refers to the total number of values in the multi-map. Formally, for any multi-map MM, $\mathsf{dsize}(\mathsf{MM}) = n = \sum_{\mathtt{key} \in \mathcal{K}} |\mathsf{MM}[\mathtt{key}]|$.

From a high level, the goal of volume-hiding STE schemes is to suppress response length leakage. We will formally define volume-hiding leakage functions in the next section.

2.4 Volume-Hiding Leakage Functions

Roughly speaking, a *volume-hiding* leakage function ensures that the number of values associated with any single key, the *volume* of the key, is not revealed and that only the maximum volume of a key is leaked.

We start with the following definition.

Definition 3. *The* signature *of a multi-map* MM *is the sequence of pairs* $((\mathtt{key}, \ell(\mathtt{key})))_{\mathtt{key} \in \mathcal{K}}$ *where* $\ell(\mathtt{key})$ *is the length of tuple of values associated with* $\mathtt{key}$ *in the multi-map.*

We next define the game $\mathbf{VHGame}_{\eta}^{\mathcal{A},\mathcal{L}}(n, \ell)$ for leakage functions $\mathcal{L} = (\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Query}})$, adversary $\mathcal{A}$, and $\eta \in \{0, 1\}$.

$\mathbf{VHGame}_{\eta}^{\mathcal{A},\mathcal{L}}(n, \ell)$**:**

1. $\mathcal{A}$ generates two signatures $S_0 = \{(\mathtt{key}, \ell_0(\mathtt{key}))\}_{\mathtt{key} \in \mathcal{K}}$ and $S_1 = \{(\mathtt{key}, \ell_1(\mathtt{key}))\}_{\mathtt{key} \in \mathcal{K}}$ with n total values and maximum volume ℓ. That is,
   - $\sum_{\mathtt{key} \in \mathcal{K}} \ell_0(\mathtt{key}) = \sum_{\mathtt{key} \in \mathcal{K}} \ell_1(\mathtt{key}) = n$;
   - $\max_{\mathtt{key} \in \mathcal{K}} \ell_0(\mathtt{key}) = \max_{\mathtt{key} \in \mathcal{K}} \ell_1(\mathtt{key}) = \ell$.
2. The challenger $\mathcal{C}$ receives signatures S_0 and S_1 from the adversary $\mathcal{A}$ and generates a multi-map MM with the signature S_η. Specifically, the challenger $\mathcal{C}$ generates $\ell_\eta(\mathtt{key})$ arbitrary values for each $\mathtt{key} \in \mathcal{K}$. The challenger $\mathcal{C}$ then sends $\mathcal{L}_{\mathsf{Setup}}(\mathsf{MM})$ to the adversary $\mathcal{A}$.
3. The adversary adaptively picks keys $\mathtt{key}_1, \ldots, \mathtt{key}_t$ for query operations. For each $\mathtt{key}_i$, the challenger $\mathcal{C}$ will compute $\mathcal{L}_{\mathsf{Query}}(\mathsf{MM}, \mathtt{key}_1, \ldots, \mathtt{key}_i)$ which is sent to the adversary.
4. Finally, the adversary $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$.

We denote by $p_{\eta}^{\mathcal{A},\mathcal{L}}(n, \ell)$ the probability that $\mathcal{A}$ outputs 1 when playing game $\mathbf{VHGame}_{\eta}^{\mathcal{A},\mathcal{L}}(n, \ell)$.

Definition 4 (Volume-Hiding Leakage Functions)**.** *A leakage function* $\mathcal{L} = (\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Query}})$ *is* volume-hiding *if and only if for all adversaries* $\mathcal{A}$ *and for all values* $n \geq \ell \geq 1$,

$$p_0^{\mathcal{A},\mathcal{L}}(n, \ell) = p_1^{\mathcal{A},\mathcal{L}}(n, \ell).$$

Given the above definition of volume-hiding leakage functions, we can now define volume-hiding, STE schemes for multi-maps.

Definition 5 (Volume-Hiding, r-Interactive, Structured Encryption for MMs)**.** *A* r*-interactive, structured encryption scheme for multi-maps,* Σ*, is* volume-hiding *if there exists a leakage function* $\mathcal{L} = (\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Query}})$ *such that*

1. Σ *is adaptively* $\mathcal{L}$*-secure according to Definition 2;*
2. $\mathcal{L}$ *is a volume-hiding leakage function according to Definition 4.*

We stress that our definition of a volume-hiding leakage function is formalized through a game in which the adversary only receives the leakage associated with the setup and queries of the STE scheme. Specifically, the adversary does not see any encrypted multi-map from the setup phase or requests and responses from queries. As a result, we are able to formalize "volume-hiding" as a property of the leakage function as opposed to the specific implementation of any STE scheme. As a consequence, our definition applies to any adversary $\mathcal{A}$ regardless of their computational power. However, this does not imply that a volume-hiding STE scheme is secure with respect to unbounded adversaries as the STE scheme is proven to be only $\mathcal{L}$-secure against computational adversaries.

2.5 Differentially Private Volume-Hiding Leakage Functions

To properly define the notion of volume-hiding within the framework of differential privacy, we consider a *sanitizer* San. A sanitizer San is a randomized algorithm that takes a signature S and samples a *sanitized signature*. Roughly speaking, we will say that a pair $(\mathcal{L}, \mathsf{San})$, consisting of a leakage function $\mathcal{L}$ and a sanitizer San, is (ϵ, δ)-*differentially private* if the probabilities that an adversary $\mathcal{A}$ outputs 1 in games $\mathbf{VHGame}_0$ and $\mathbf{VHGame}_1$, respectively played on the sanitized versions $\mathsf{San}(S_0)$ and $\mathsf{San}(S_1)$ of two *neighboring signatures* S_0 and S_1, are related through ϵ and δ. Let us now proceed more formally.

Definition 6. *Signatures* $S_0 = (\mathtt{key}, \ell_0(\mathtt{key}))_{\mathtt{key}\in\mathcal{K}}$ *and* $S_1 = (\mathtt{key}, \ell_1(\mathtt{key}))_{\mathtt{key}\in\mathcal{K}}$ *are* neighbors *if there exist* $\mathtt{key}_0, \mathtt{key}_1 \in \mathcal{K}$ *such that*

1. *for all* $\mathtt{key} \notin \{\mathtt{key}_0, \mathtt{key}_1\}$, $\ell_0(\mathtt{key}) = \ell_1(\mathtt{key})$*;*
2. $\ell_0(\mathtt{key}_0) = \ell_1(\mathtt{key}_0) + 1$*;*
3. $\ell_0(\mathtt{key}_1) = \ell_1(\mathtt{key}_1) - 1$.

We define game $\mathbf{dpVH}_\eta$, for $\eta = 0, 1$, as in the previous section with the only difference that the multi-set $\mathcal{D}$ is constructed by applying the sanitizer San to signature S_η.

$\mathbf{dpVH}_\eta^{\mathcal{A},\mathcal{L},\mathsf{San}}(n)$**:**

1. $\mathcal{A}$ outputs neighboring signatures $S_0 = (\mathtt{key}, \ell_0(\mathtt{key}))_{\mathtt{key}\in\mathcal{K}}$ and $S_1 = (\mathtt{key}, \ell_1(\mathtt{key}))_{\mathtt{key}\in\mathcal{K}}$ with n total values. That is, $\sum_{\mathtt{key}\in\mathcal{K}} \ell_0(\mathtt{key}) = \sum_{\mathtt{key}\in\mathcal{K}} \ell_1(\mathtt{key}) = n$.
2. The challenger $\mathcal{C}$ receives signatures S_0 and S_1 from $\mathcal{A}$ and samples a sanitized signature $S_{\mathsf{San}} = \{(\mathtt{key}, \ell_{\mathsf{San}}(\mathtt{key})\}_{\mathtt{key}\in\mathcal{K}}$ by running San on S_η.
3. The challenger $\mathcal{C}$ constructs a multi-map MM with signature S_{San} by picking $\ell_{\mathsf{San}}(\mathtt{key})$ arbitrary values for each $\mathtt{key} \in \mathcal{K}$. $\mathcal{C}$ then sends $\mathcal{L}_{\mathsf{Setup}}(\mathsf{MM})$ to $\mathcal{A}$.
4. The adversary $\mathcal{A}$ adaptively picks keys $\mathtt{key}_1, \ldots, \mathtt{key}_t$ for query operations. For each $\mathtt{key}_i$, the challenger $\mathcal{C}$ computes and sends $\mathcal{L}_{\mathsf{Query}}(\mathsf{MM}, \mathtt{key}_1, \ldots, \mathtt{key}_i)$ to the adversary.
5. Finally, the adversary $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$.

We denote by $p_{\eta,\mathcal{A}}^{\mathcal{L},\mathcal{S},\mathsf{San}}$ the probability that adversary $\mathcal{A}$ outputs 1 when playing game $\mathbf{dpVH}_\eta^{\mathcal{L},\mathcal{S},\mathsf{San}}$

Definition 7 (Differentially Private Volume-Hiding Leakage Functions)**.** *We say that* $(\mathcal{L}, \mathcal{S})$ *is* (ϵ, δ)-differentially private volume-hiding *if for all* n *and for all adversaries* $\mathcal{A}$ *that output two neighboring signatures with the same total number of values* n*, it holds that*

$$p_{0,\mathcal{A}}^{\mathcal{L},\mathcal{S},\mathsf{San}}(n) \leq e^\epsilon \cdot p_{1,\mathcal{A}}^{\mathcal{L},\mathcal{S},\mathsf{San}}(n) + \delta.$$

If $\delta = 0$, we will simply say ϵ-differential private instead of (ϵ, δ)-differential private. Note also that a volume-hiding leakage function is $(0, \mathtt{negl}(n))$-differentially private with respect to the identity sanitizer Id that, for every signatures S, returns the signature itself.

Our definition follows the differentially private literature (see [DMNS06] as an example) where the sanitizer is used to preprocesses the data-base (the multi-map in our setting) before being stored. In our construction of Section 4, the multi-map will be implicitly sanitized at query time. The effect on the mitigating leakage will be identical to sanitization at setup time but it will result in a smaller server storage.

3 Volume-Hiding EMM

In this section, we describe our volume-hiding structured encryption scheme for multi-maps. To our knowledge, our construction is the first to achieve both asymptotically optimal query and storage complexity. Furthermore, we show that our construction is concretely efficient using experiments. In Section 5, we show that our constructions use less server storage and improve query overhead by a factor of 10-16x over the previous best constructions [KM19].

One of the major tools that will be used in our construction is cuckoo hashing, which we now describe.

Cuckoo Hashing. Cuckoo hashing was introduced by Pagh and Rodler [PR04] and consists of two algorithms Build and Search. Given n key-value pairs $(\mathtt{key}_1, \mathtt{value}_1), \ldots, (\mathtt{key}_n, \mathtt{value}_n)$, Build constructs $\mathtt{Table}$ consisting two arrays T_1 and T_2 each with the capacity to hold $t = (1+\alpha)n$ pairs for any constant $\alpha > 0$. The Build algorithm inserts the pairs one at the time using two hash functions $h_1, h_2 : \mathcal{K} \to [t]$. To insert pair $X = (\mathtt{key}_i, \mathtt{value}_i)$, Build places X in location $T_1[h_1(\mathtt{key}_i)]$. If the location is empty, we are done. If the location is currently occupied by another pair, $Y = (\mathtt{key}_j, \mathtt{value}_j)$, Y is evicted and the algorithm attempts to inserting Y in location $T_2[h_2(\mathtt{key}_j)]$. Again if the location is empty, we are done. Otherwise we evict the pair Z found in $T_2[h_2(\mathtt{key}_j)]$ and try to insert Z in the location of T_1 specified by h_1, and so on. Note that cuckoo hashing guarantees that a pair $(\mathtt{key}_i, \mathtt{value}_i)$ is found either at $T_1[h_1(\mathtt{key}_i)]$ or $T_2[h_2(\mathtt{key}_i)]$. Thus, Search only retrieves two table locations. We say that cuckoo hashing fails if a key-value pair is not inserted after $\Theta(\log n)$ evictions. It is possible to show that all n key-value pairs will be successfully inserted with large constant probability. Unfortunately, this failure probability is too large for privacy reasons and this has been used to compromise privacy of oblivious RAMs (see, e.g.,[KLO12]).

Cuckoo Hashing with a Stash. As a result, we resort to a modification of cuckoo hashing presented by Kirsch, Mitzenmacher and Wieder [KMW09] that introduces a stash S of some fixed capacity s. After $\Theta(\log n)$ evictions, if a key-value pair has not been inserted yet, the pair is be inserted into the stash. Cuckoo hashing with a stash fails if strictly more than s items are attempted to be inserted into the stash. The introduction of the stash reduces the failure probability exponentially in the stash size:

Theorem 3. *For constants $d, \alpha > 0$, there exists algorithm* Build *that stores n pairs into two arrays T_1 and T_2 of size $(1+\alpha)n$ and a stash S. Then. the probability that, after all n pairs have been inserted with at most $d \log n$ evictions, S has size greater than s is $O(n^{-s})$.*

For a proof, see Theorem 2.1 of [KMW09] and the extension to non-constant sized stashes in [GM11].

High-Level Description of vhMM. We start by describing our volume-hiding, multi-map encryption scheme from an intuitive level. Consider any multi-map MM storing $\{\mathtt{key}_i, \vec{\mathtt{v}}_i\}_{i \in [m]}$ with n total values and a maximum volume of ℓ. The Setup algorithm will store the n values using cuckoo hashing with a stash with two tables, T_1 and T_2, of size $t = (1+\alpha)n$ for any constant $\alpha > 0$. We will use a PRF F that output values in the range $[t]$. Consider the j-th value, $\vec{\mathtt{v}}_i[j]$ associated with $\mathtt{key}_i$. Then, $(\mathtt{key}_i, \vec{\mathtt{v}}_i[j])$ will be assigned locations $T_1[F_K(\mathtt{key}_i \,||\, j \,||\, 0)]$, $T_2[F_K(\mathtt{key}_i \,||\, j \,||\, 1)]$ or the stash using the cuckoo hashing insertion algorithm. If we set the stash to be $f(n) = \omega(1)$, then the insertion will fail with probability at most $O(n^{f(n)}) = \mathtt{negl}(n)$. All empty table locations will be filled with a dummy value. All table locations are then encrypted. The resulting encrypted table will be the encrypted multi-map EMM outsourced to the server. The small stash containing at most $f(n) = \omega(1)$ values will be stored by the client.

We now move onto the query operation for our STE scheme. The Query algorithm executed by the client will simply send the 2ℓ values $\{F_K(\mathtt{key}_i \,||\, j \,||\, 0), F_K(\mathtt{key}_i \,||\, j \,||\, 1)\}_{j \in [\ell]}$ to the server. The server executes Reply by returning the encrypted values located at $\{T_1[F_K(\mathtt{key}_i \,||\, j \,||\, 0)], T_2[F_K(\mathtt{key}_i \,||\, j \,||\, 1)]\}_{j \in [\ell]}$. The client retrieves the tuple of associated values by decrypting all table locations in the server's response as well as checking the stash.

Formal Description of vhMM. We now formally present our first volume-hiding STE scheme for multi-maps vhMM. In particular, our construction will use pseudorandom family of functions $\mathcal{F} = \{F_s\}_{s \in \{0,1\}^\star}$ and an IND-CPA secure encryption scheme $\mathcal{E} = (\mathsf{Enc}, \mathsf{Dec})$. Furthermore, our construction is parameterized by a function $f(n) = \omega(1)$ and a constant $\alpha > 0$.

$(K, \mathsf{EMM}) \leftarrow \mathsf{vhMM.Setup}(1^\lambda, \mathsf{MM} = \{\mathtt{key}_i, \vec{\mathtt{v}}_i\}_{i \in [m]})$**:**

```
1. Randomly select PRF seed $K_{\mathsf{PRF}} \leftarrow \{0,1\}^\lambda$.
2. Randomly select encryption key $K_{\mathsf{Enc}} \leftarrow \{0,1\}^\lambda$.
3. Create two empty arrays, $T_1, T_2$ of length $t = (1+\alpha)n$ where $n = \sum_{i \in [m]} |\vec{\mathtt{v}}_i|$.
4. Initialize $\mathtt{Stash} \leftarrow \emptyset$.
5. For each $i \in [m]$ and each $j \in [|\vec{\mathtt{v}}_i|]$:
    (a) Insert $(\mathtt{key}_i, \vec{\mathtt{v}}_i[j])$ using the cuckoo hashing with a stash insertion algorithm where $(\mathtt{key}_i, \vec{\mathtt{v}}_i[j])$
        is assigned to one of $T_1[F_{K_{\mathsf{PRF}}}(\mathtt{key}_i \,||\, j \,||\, 0)]$, $T_2[F_{K_{\mathsf{PRF}}}(\mathtt{key}_i \,||\, j \,||\, 1)]$ or $\mathtt{Stash}$.
6. If $\mathtt{Stash}$ contains more than $f(n)$ items, abort.
7. For each location in $T_1$ or $T_2$ that is empty, insert $(\perp, \perp)$ into the location.
8. For each $i \in [t]$:
    (a) Set $T_1[i] \leftarrow \mathsf{Enc}(K_{\mathsf{Enc}}, T_1[i])$.
    (b) Set $T_2[i] \leftarrow \mathsf{Enc}(K_{\mathsf{Enc}}, T_2[i])$.
9. Set the private key as $K \leftarrow (K_{\mathsf{PRF}}, K_{\mathsf{Enc}}, \mathtt{Stash})$.
10. Set $\mathsf{EMM} \leftarrow (T_1, T_2)$.
11. Return $(K, \mathsf{EMM})$.
```

$\mathsf{Request} \leftarrow \mathsf{vhMM.Query}(K, \mathtt{key})$:

```
1. Parse $K$ as $(K_{\mathsf{PRF}}, K_{\mathsf{Enc}}, \mathtt{Stash})$.
2. Return $(F_{K_{\mathsf{PRF}}}(\mathtt{key} \,||\, i \,||\, 0), F_{K_{\mathsf{PRF}}}(\mathtt{key} \,||\, i \,||\, 1))_{i \in [\ell]}$.
```

$\mathsf{Response} \leftarrow \mathsf{vhMM.Reply}(\mathsf{Request}, \mathsf{EMM})$:

```
1. Parse Request as $\{\mathsf{ind}_{0,i}, \mathsf{ind}_{1,i}\}_{i \in [\ell]}$.
2. Return $(T_1[\mathsf{ind}_{0,i}], T_2[\mathsf{ind}_{1,i}])_{i \in [\ell]}$.
```

$\vec{\mathtt{v}} \leftarrow \mathsf{vhMM.Result}(K, \mathtt{key}, \mathsf{Response})$:

```
1. Parse $K$ as $(K_{\mathsf{PRF}}, K_{\mathsf{Enc}}, \mathtt{Stash})$.
2. Parse Response as $(\mathtt{ct}_{0,i}, \mathtt{ct}_{1,i})_{i \in [\ell]}$.
3. Set $\vec{\mathtt{v}} \leftarrow \emptyset$.
4. For each $i \in [\ell]$:
    (a) Set $(\mathtt{key}', \mathtt{value}') \leftarrow \mathsf{Dec}(K_{\mathsf{Enc}}, \mathtt{ct}_{0,i})$.
    (b) If $\mathtt{key}' = \mathtt{key}$, $\vec{\mathtt{v}} \leftarrow \vec{\mathtt{v}} \cup \{\mathtt{value}'\}$.
    (c) Set $(\mathtt{key}', \mathtt{value}') \leftarrow \mathsf{Dec}(K_{\mathsf{Enc}}, \mathtt{ct}_{1,i})$.
    (d) If $\mathtt{key}' = \mathtt{key}$, $\vec{\mathtt{v}} \leftarrow \vec{\mathtt{v}} \cup \{\mathtt{value}'\}$.
5. For each $(\mathtt{key}', \mathtt{value}') \in \mathtt{Stash}$:
    (a) If $\mathtt{key}' = \mathtt{key}$, $\vec{\mathtt{v}} \leftarrow \vec{\mathtt{v}} \cup \{\mathtt{value}'\}$.
```

6. Return $\vec{v}$.

Security. We now describe the leakage of vhMM due to algorithms Setup and Query. From a high level, the EMM reveals to the adversarial server $(2+2\alpha)n$ encrypted key-value pairs. As a result, $\mathcal{L}_{\mathsf{Setup}} = \mathsf{dsize}$ which is the leakage functions that reveals the number n of total values in MM. For each query, the adversarial server learns two things. First, the adversarial server learns query equality by observing identical 2ℓ PRF values. Secondly, the adversary learns the maximum volume of a key in MM. So, $\mathcal{L}_{\mathsf{Query}} = (\mathsf{qeq}, \mathsf{mrlen})$. Note, that $\mathcal{L}_{\mathsf{Query}}$ and $\mathcal{L}_{\mathsf{Setup}}$ do not include the response length leakage rlen and thus, intuitively, vhMM is volume-hiding. We prove the following two lemmata in Appendix A.

Lemma 1. *Let $\mathcal{L} = (\mathsf{dsize}, (\mathsf{qeq}, \mathsf{mrlen}))$. For every constant $\alpha > 0$ and $f(n) = \omega(1)$, vhMM is an adaptive $\mathcal{L}$-secure non-interactive STE scheme for multi-maps.*

Lemma 2. *Leakage function $\mathcal{L} = (\mathsf{dsize}, (\mathsf{qeq}, \mathsf{mrlen}))$ is volume-hiding.*

By combining the two lemmas above we obtain the main theorem of this section

Theorem 4. *Non-interactive STE for multi-maps vhMM is* volume-hiding.

Efficiency. By Theorem 3, the probability that the $\mathtt{Stash}$ grows larger than $f(n)$ is at most $n^{-f(n)} = \mathtt{negl}(n)$. The outsourced EMM contains exactly $(2+2\alpha)n$ encryptions of key-value pairs. The client generates 2ℓ PRF values during $\mathsf{vhMM.Query}$ and the server responds with 2ℓ key-value encryptions during $\mathsf{vhMM.Reply}$. The client stores two private keys and the $\mathtt{Stash}$ consisting of at most $f(n)$ key-value pairs.

Theorem 5. *Non-interactive STE for multi-maps vhMM aborts with negligible probability. It requires $(2+2\alpha)n$ storage on the server. Each query requires 4ℓ communication between the client and the server. The client stores $O(1)$ private keys as well as at most $f(n) = \omega(1)$ key-value pairs.*

3.1 Improving Communication

In this section, we present an improvement to our construction vhMM from the previous section. First, we take a careful look at vhMM to look for any wasteful parts. In particular, consider the way that each of the n key-value pairs of the input multi-map are assigned to table locations. For vhMM, any value $\vec{v}_i[j]$ associated with $\mathtt{key}_i$ will be assigned to locations $F_{K_{\mathsf{PRF}}}(\mathtt{key}_i \,||\, j \,||\, 0)$ and $F_{K_{\mathsf{PRF}}}(\mathtt{key}_i \,||\, j \,||\, 1)$. As a result, $\mathsf{vhMM.Query}$ requires the client to send 2ℓ PRF values. This seems quite wasteful as the 2ℓ table locations associated with $\mathtt{key}_i$ are very structured. We show that we can modify vhMM such that the $\mathsf{Request}$ generated by Query can be compressed to only contain a single PRF value as opposed to 2ℓ values. To do this, we will use *delegatable PRFs.*

Delegatable PRFs. A *family of delegatable PRFs* (dPRF), first discussed in [KPTZ13], enables the owner of the secret seed K to *delegate* an untrusted party to compute $F_K(x)$ for all values $x \in S$, where S is a subset taken from a specified family of subsets. The delegation is obtained by computing a *token* $\mathtt{tok}_S$ that allows the computation of the PRF for all $x \in S$, without any further intervention of the owner without accessing the secret seed. The security requirement is that all values $F_K(x)$ for $x \notin S$ remain indistinguishable from truly random values even for an adversary that has access to $\mathtt{tok}_S$. Note that this can be trivially achieved by having the owner release the value of $F_K(x)$ for all $x \in S$. Thus, the goal of dPRFs is to construct tokens of size $o(|S|)$. To construct efficient dPRFs, we will use the famous GGM PRF construction which we describe next.

The GGM PRF construction. The GGM construction [GGM86] builds a family of PRFs $\mathcal{F} = \{F_K(\cdot)\}$ from a length-doubling pseudorandom generator G as follows. For any input v of length λ bits, $G(v)$ will result in 2λ bits. For convenience, we denote the first λ bits of the $G(v)$ as $G_0(v)$ and the last λ bits of $G(v)$ as $G_1(v)$. Consider a function $F_{\{0,1\}^\lambda} : \{0,1\}^t \to \{0,1\}^\lambda$ with a private key and output of the same length λ. The GGM construction uses a binary tree of height t where each node of the tree is labeled with a binary string encoding the path from the root to the node itself using the convention that left is encoded by a "0" and right is encoded by a "1" and the root at level 0 is labeled with the empty string $\perp$.

Every node with label $x_1, \ldots, x_i \in \{0,1\}$ is associated with value $F_K(x_1, \ldots, x_i)$ computed in the following recursive manner. The root, labeled with the empty string $\perp$, is assigned the value $F_K(\perp) := K$. If a node is associated with value v, then its left child is associated with $G_0(v)$, the first λ bits of $G(v)$, and its right child is associated with $G_1(v)$, the last λ bits of $G(v)$. This recursive rule is equivalent to defining $F_K(x_1, \ldots, x_i) := G_{x_i}(G_{x_{i-1}}(\cdots G_{x_1}(K) \cdots))$. The PRF output $F_K(X)$ for any t-bit string $X \in \{0,1\}^t$ is the value associated with the leaf with label X. The authors of [GGM86] show that values associated with leaf nodes are indistinguishable from random values for any computational adversary.

Delegating Prefixes. In [KPTZ13], the above GGM construction of a PRF is used to implement a delegatable PRF for subsets of strings with matching prefixes. More formally, the construction of [KPTZ13] consists of two efficient algorithms: algorithm $\mathsf{dPRF.GenTok}$ that takes as input the private dPRF key K and an s-bit prefix $X = x_1, \ldots, x_s$ with $s \leq t$ and outputs a token $\mathtt{tok}_X$; and algorithm $\mathsf{dPRF.Eval}$ that takes the token for the s-bit prefix X, $\mathtt{tok}_X$ and string $Y \in \{0,1\}^{t-s}$ and computes the value $F_K(X \mid\mid Y)$. In other words, $\mathsf{dPRF.Eval}$ enables computation of the PRF evaluation of any string whose prefix is X.

$\mathsf{dPRF.GenTok}$ for a prefix X is implemented by returning $\mathtt{tok}_X := F_K(X)$ where F is the GGM PRF construction. For any string Y with prefix X, $\mathsf{dPRF.Eval}$ is implemented by simply computing the values the node labeled Y from its ancestor node labeled by X in the GGM PRF construction. This can be done since the pseudorandom generator G is public. For security of dPRF, we only consider adversaries that perform prefix-free string queries. Prefix-free string queries refer to the fact that each query must not be a prefix of any query. In [KPTZ13], it is shown that the resulting queried tokens are computationally indistinguishable from random values.

High-Level Description of dprfMM. We now describe the modifications to vhMM using prefix delegatable PRFs. The main idea is to replace all instances of a PRF in vhMM with a prefix dPRF. For cuckoo hashing, a value $\vec{\mathsf{v}}_i[j]$ associated with $\mathtt{key}_i$ will be assigned to the locations $F_K(\mathtt{key}_i \mid\mid j \mid\mid 0)$ in table 1 and $F_K(\mathtt{key}_i \mid\mid j \mid\mid 1)$ in table 2 where F is the GGM PRF construction. Furthermore, each $j \in [\ell]$ will be represented using a $\lceil \log_2 \ell \rceil$-bit string.

We modify Query such that to query any $\mathtt{key}$, the client will construct a token $\mathtt{tok}_{\mathtt{key}}$ that will delegate the computation for the PRF for any value whose prefix matches $\mathtt{key}$. In particular, this enables the adversarial server to compute the set of PRF values $\{F_K(\mathtt{key} \mid\mid i \mid\mid 0), F_K(\mathtt{key} \mid\mid i \mid\mid 1)\}_{i \in [\ell]}$ as they all have the shared prefix $\mathtt{key}$. Note that $\mathtt{tok}_{\mathtt{key}}$ consists of the single PRF value $F_K(\mathtt{key})$. As a result, dprfMM reduces the client-to-server communication from 2ℓ PRF values to just a single PRF value.

Detailed Description of dprfMM. The functions Setup and Reply remain the same as vhMM except replacing the PRF F with the prefix dPRF F construction above.

```
$\mathsf{Request} \leftarrow \mathsf{dprfMM.Query}(K, \mathtt{key})$:
  1. Parse $K$ as $(K_{\mathsf{PRF}}, K_{\mathsf{Enc}}, \mathtt{Stash})$.
  2. Compute $\mathtt{tok}_{\mathtt{key}} \leftarrow \mathsf{dPRF.GenTok}(K_{\mathsf{PRF}}, \mathtt{key})$.
  3. Return $\mathtt{tok}_{\mathtt{key}}$.

$\mathsf{Response} \leftarrow \mathsf{dprfMM.Reply}(\mathsf{Request}, \mathsf{EMM})$:
  1. Parse $\mathsf{Request}$ as $\mathtt{tok}_{\mathtt{key}}$.
  2. For each $i \in [\ell]$:
     (a) Compute $F_{K_{\mathsf{PRF}}}(\mathtt{key} \mid\mid i \mid\mid 0) \leftarrow \mathsf{dPRF.Eval}(\mathtt{tok}_{\mathtt{key}}, i \mid\mid 0)$.
     (b) Compute $F_{K_{\mathsf{PRF}}}(\mathtt{key} \mid\mid i \mid\mid 1) \leftarrow \mathsf{dPRF.Eval}(\mathtt{tok}_{\mathtt{key}}, i \mid\mid 1)$.
  3. Return $\{T_1[F_{K_{\mathsf{PRF}}}(\mathtt{key} \mid\mid i \mid\mid 0)], T_2[F_{K_{\mathsf{PRF}}}(\mathtt{key} \mid\mid i \mid\mid 1)]\}_{i \in [\ell]}$.
```

Security. We note that the leakage of dprfMM is identical to vhMM where $\mathcal{L}_{\mathsf{Setup}} = \mathsf{dsize}$ and $\mathcal{L}_{\mathsf{Query}} = (\mathsf{qeq}, \mathsf{mrlen})$. The proof of security can be found in Appendix B.

Theorem 6. dprfMM *is an adaptive* $\mathcal{L} = (\mathsf{dsize}, (\mathsf{qeq}, \mathsf{mrlen}))$*-secure STE scheme for multi-maps when* $f(n) = \omega(1)$ *and any constant* $\alpha > 0$*.*

Theorem 7. dprfMM *is volume-hiding.*

Efficiency. Note that dprfMM reduces the client-to-server communication from 2ℓ PRF values to one PRF seed of length equal to the security parameter. Server-to-client communication stays 2ℓ which is asymptotically optimal.

Theorem 8. *Construction* dprfMM *requires* $(2 + 2\alpha)n$ *storage on the server. Each query requires* $\ell + 1$ *communication between the client and the server. The client stores* $O(1)$ *private keys as well as at most* $f(n) = \omega(1)$ *key-value pairs. Both the client and server perform* $O(\ell)$ *computation.*

4 Differentially Private Volume-Hiding EMM

In this section, we describe our construction of a STE scheme for multi-maps which is differentially private volume-hiding. In particular, we will slightly modify the dprfMM to be differentially private while improving the query complexity significantly. In the resulting scheme, the client will have to fetch some information from the server before being able to actually issue the query and thus the scheme is not non-interactive.

Note, all volume-hiding constructions pad queries to the maximum volume resulting in $\Omega(\ell)$ query complexity. To be completely volume-hiding, this is necessary to hide the queried key with the largest volume as opposed to other queries with smaller volumes. For differentially private volume-hiding, we can relax our privacy guarantees. Instead, we will perturb the volume of each key by a small amount related to the privacy budget ϵ. As a result, the query complexity for any key is dependent only on the volume of the key as opposed to the maximum volume. We now implement these ideas to obtain our differentially private volume-hiding scheme.

High-Level Description of dpMM**.** To obtain dpMM, we perform a slight modification to dprfMM. Setup will store all the values of the multi-map using cuckoo hashing via a delegatable PRF. Additionally, the server will also construct an additional cuckoo hash table for a *count table*. Specifically, the count table will store $\ell(\mathtt{key})$, denoting the number of values associated with $\mathtt{key}$, for each $\mathtt{key}$ in the input multi-map with non-zero volume.

Queries will be two rounds as opposed to one round. First, the client downloads $\ell(\mathtt{key})$ using cuckoo hashing from the server. Next, the client will generate $\mathtt{tok}_{\mathtt{key}}$ using the dPRF. However, the client will also generate an integer that is dependent on the volume of $\mathtt{key}$ denoted by $\ell(\mathtt{key})$. In particular, the client will also send the value $f(\ell(\mathtt{key}), \lambda) := 2\ell(\mathtt{key}) + l^*(\lambda) + \mathsf{noise}(\mathtt{key})$ along with $\mathtt{tok}_{\mathtt{key}}$. The value $\mathsf{noise}(\mathtt{key})$ is drawn according to distribution $\mathsf{Lap}(2/\epsilon)$, which is the Laplacian distribution with parameter $2/\epsilon$. We point out that this value will be identical each time $\mathtt{key}$ is queried. This is accomplished by drawing from a Laplacian distribution using pseudorandom bits derived using $\mathtt{key}$. We pick the value $l^*(\lambda)$ later.

Reply is modified such that the server will only expand the dPRF at values $F_K(\mathtt{key} \mid\mid i \mid\mid 0), F_K(\mathtt{key} \mid\mid i \mid\mid 1)$ where $1 \leq i \leq f(\ell(\mathtt{key}), \lambda)$. Similarly, the server will only return the encrypted values at table locations for these values. As a result, the query complexity is $f(\ell(\mathtt{key}), \lambda)$. Note that if $\mathsf{noise}(\mathtt{key}) = \mathsf{Lap}(2/\epsilon) < -l^*(\lambda)$, then our construction might be lossy as some possible values associated with $\mathtt{key}$ stored in the cuckoo hash tables will not be returned to the client. As a result, we pick $l^*(\lambda) = \omega(\log \lambda)$ such that $\Pr[\mathsf{Lap}(2/\epsilon) < -l^*(\lambda)]$ is negligible in λ. Therefore, all associated values will be returned except with negligible probability.

Detailed Description of dpMM**.** Setup will be almost identical except that the client will also store a count table in the EMM using cuckoo hashing. For an input multi-map $\{\mathtt{key}_i, \vec{\mathrm{v}}_i\}_{i\in[m]}$, the client will store $\mathsf{CT} := \{\mathtt{key}_i, \ell(\mathtt{key}_i)\}_{i\in[m]}$ where $\ell(\mathtt{key}_i)$ is the number of values associated with $\mathtt{key}_i$. Note, dpMM will consist of four functions $\mathsf{Query}^0, \mathsf{Reply}^0, \mathsf{Query}^1$ and Reply^1 which constitutes the additional round of interaction.

$(K, \mathsf{EMM}) \leftarrow \mathsf{dpMM.Setup}(1^\lambda, \mathsf{MM} = \{\mathtt{key}_i, \vec{\mathrm{v}}_i\}_{i\in[m]})$**:**

1. Execute $(K, \mathsf{EMM}) \leftarrow \mathsf{dprfMM.Setup}(1^\lambda, \mathsf{MM})$.

```
2. Create two empty arrays, $\mathsf{CT}_1, \mathsf{CT}_2$ of length $t = (1+\alpha)n$.
3. Generate a $\mathtt{Stash}_{\mathsf{CT}} \leftarrow \emptyset$.
4. For each $i \in [m]$:
    (a) Insert $(\mathtt{key}_i, \ell(\mathtt{key}_i))$ using the cuckoo hashing with a stash insertion algorithm where $(\mathtt{key}_i, \ell(\mathtt{key}_i))$
        is assigned to one of $T_1[F_{K_{\mathsf{PRF}}}(CT \mid\mid \mathtt{key}_i \mid\mid 1)]$, $T_2[F_{K_{\mathsf{PRF}}}(CT \mid\mid \mathtt{key}_i \mid\mid 1)]$ or $\mathtt{Stash}_{\mathsf{CT}}$.
5. If $\mathtt{Stash}_{\mathsf{CT}}$ contains more than $f(n)$ items, abort.
6. For each location in $\mathsf{CT}_1$ or $\mathsf{CT}_2$ that is empty, insert $(\perp, \perp)$ into the location.
7. For each $i \in [t']$:
    (a) Set $\mathsf{CT}_1[i] \leftarrow \mathsf{Enc}(K_{\mathsf{Enc}}, \mathsf{CT}_1[i])$.
    (b) Set $\mathsf{CT}_2[i] \leftarrow \mathsf{Enc}(K_{\mathsf{Enc}}, \mathsf{CT}_2[i])$.
8. Set the private key as $K \leftarrow K \cup \mathtt{Stash}_{\mathsf{CT}}$.
9. Set $\mathsf{EMM} \leftarrow \mathsf{EMM} \cup \{\mathsf{CT}_1, \mathsf{CT}_2\}$.
10. Return $(K, \mathsf{EMM})$.
```

$\mathsf{Request}_0 \leftarrow \mathsf{dpMM.Query}^0(K, \mathtt{key})$:

```
1. Parse $K$ as $(K_{\mathsf{PRF}}, K_{\mathsf{Enc}}, \mathtt{Stash})$.
2. Return $\mathtt{tok}_{CT||\mathtt{key}} \leftarrow \mathsf{dPRF.GenTok}(K_{\mathsf{PRF}}, CT \mid\mid \mathtt{key})$.
```

$\mathsf{Response}_0 \leftarrow \mathsf{dpMM.Reply}^1(\mathsf{Request}_0, \mathsf{EMM})$:

```
1. Parse Request as $\mathtt{tok}_{CT||\mathtt{key}}$.
2. Get $F_{K_{\mathsf{PRF}}}(CT \mid\mid \mathtt{key} \mid\mid 0), F_{K_{\mathsf{PRF}}}(CT \mid\mid \mathtt{key} \mid\mid 1)$ using $\mathsf{dPRF.Eval}(\mathtt{tok}_{CT||\mathtt{key}}, 0)$ and $\mathsf{dPRF.Eval}(\mathtt{tok}_{CT||\mathtt{key}}, 1)$.
3. Return $\mathsf{CT}_1[F_{K_{\mathsf{PRF}}}(CT \mid\mid \mathtt{key} \mid\mid 0)], \mathsf{CT}_2[F_{K_{\mathsf{PRF}}}(CT \mid\mid \mathtt{key} \mid\mid 1)]$.
```

$\mathsf{Request}_1 \leftarrow \mathsf{dpMM.Query}^1(K, \mathtt{key}, \mathsf{Response}_0)$:

```
1. Use $\mathtt{Stash}_{\mathsf{CT}}$ and $\mathsf{Response}_0$ to retrieve $\ell(\mathtt{key})$. If $\ell(\mathtt{key})$ is not found, set $\ell(\mathtt{key}) \leftarrow 0$.
2. Parse $K$ as $(K_{\mathsf{PRF}}, K_{\mathsf{Enc}}, \mathtt{Stash})$.
3. Compute $\mathtt{tok}_{\mathtt{key}} \leftarrow \mathsf{dPRF.GenTok}(K_{\mathsf{PRF}}, \mathtt{key})$.
4. Compute $\mathsf{noise}(\mathtt{key}) \leftarrow \mathsf{Lap}_{\mathtt{key}}(2/\epsilon)$.
5. Compute $X \leftarrow \ell(\mathtt{key}) + l^*(\lambda) + \mathsf{noise}(\mathtt{key})$.
6. Return $(\mathtt{tok}_{\mathtt{key}}, X)$.
```

$\mathsf{Response}_1 \leftarrow \mathsf{dpMM.Reply}^1(\mathsf{Request}_1, \mathsf{EMM})$:

```
1. Parse Request as $(\mathtt{tok}_{\mathtt{key}}, X)$.
2. For each $i \in [X]$:
    (a) Compute $F_{K_{\mathsf{PRF}}}(\mathtt{key} \mid\mid i \mid\mid 0) \leftarrow \mathsf{dPRF.Eval}(\mathtt{tok}_{\mathtt{key}}, i \mid\mid 0)$.
    (b) Compute $F_{K_{\mathsf{PRF}}}(\mathtt{key} \mid\mid i \mid\mid 1) \leftarrow \mathsf{dPRF.Eval}(\mathtt{tok}_{\mathtt{key}}, i \mid\mid 1)$.
```

	Densest Subgraph Transform [KM19]				dprfMM				dpMM			
Input Multi-Map												
Number of Values (n)	2^{16}	2^{18}	2^{20}	2^{22}	2^{16}	2^{18}	2^{20}	2^{22}	2^{16}	2^{18}	2^{20}	2^{22}
Plaintext Raw Byte Size (MB)	1.05	4.19	16.78	67.11	1.05	4.19	16.78	67.11	1.05	4.19	16.78	67.11
EMM Storage												
Server (MB)	5.53	22.74	88.25	384.40	5.45	21.81	87.24	348.97	6.81	27.26	109.05	436.21
Client Stash (KB)	N/A	N/A	N/A	N/A	0.16	0.50	1.52	4.84	0.21	0.63	1.97	6.18
Query Communication												
Upload (Bytes)	16	16	16	16	16	16	16	16	36	36	36	36
Download (Bytes Per Result)	675.2	780.8	841.6	1008.0	64	64	64	64	64	64	64	64
CPU Costs												
Query (Client ms)	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
Reply (Server ms Per Result)	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
Result (Client ms Per Result)	0.01	0.01	0.01	0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01

Table 2: Microbenchmarks for server and network costs comparing volume-hiding STE schemes. We denote n as the total number of values in the input multi-map. If ℓ is the maximum volume of any key, then the first two column constructions must download ℓ results. On the other hand, the number of results for dpMM will be significantly smaller than ℓ on average. The above results apply for any input multi-map structure, query distribution as well as any value ℓ. We denote milliseconds by ms.

3. Return $\{T_1[F_{K_{\mathsf{PRF}}}(\mathtt{key} \,||\, i \,||\, 0)], T_2[F_{K_{\mathsf{PRF}}}(\mathtt{key} \,||\, i \,||\, 1)]\}_{i \in [X]}$.

dpMM.Result is identical to dprfMM except that the server's response will contain less than ℓ encrypted key-value pairs.

Correctness. The probability that not all values are retrieved is upper bounded by the probability that $\mathsf{noise}(\mathtt{key}) < -l^*(\lambda)$. Since $\mathsf{noise}(\mathtt{key})$ is distributed according to $\mathsf{Lap}(2/\epsilon)$, we obtain that this probability is $O(e^{l^*(\lambda)})$ which is negligible in λ, as $l^*(\lambda) = \omega(\log \lambda)$.

Security. We note that dpMM has almost identical leakage to both dprfMM and vhMM except that dpMM also leaks the value $\ell(\mathtt{key}) + l^*(\lambda) + \mathsf{Lap}_{\mathtt{key}}(2/\epsilon)$ for each queried key. Additionally, dpMM leaks the number of unique keys m from the size of the cuckoo hash table for the count table. We prove that dpMM is a differentially private volume-hiding STE scheme for multi-maps in Appendix C due to lack of space.

Theorem 9. dpMM *is ϵ-differentially private volume-hiding.*

Efficiency. Note that dpMM has almost identical efficiency to dprfMM except for two major differences. The number of encrypted key-value pairs in the server's response generated by Reply is $2\ell(\mathtt{key}) + l^*(\lambda) + \mathsf{Lap}_{\mathtt{key}}(2/\epsilon)$. Note, that $l^*(\lambda) + \mathsf{Lap}_{\mathtt{key}}(2/\epsilon) = \omega(\log \lambda)$ except with probability negligible in λ. For keys with volume significantly smaller than the maximum volume ℓ, dpMM achieves much better query complexity. Also, the outsourced EMM consists of two cuckoo hash schemes instead of one. So, we get:

Theorem 10. *For any $\alpha > 0$, construction* dpMM *requires $(2 + 2\alpha)(m + n)$ storage on the server. Each query requires $1 + \ell(\mathtt{key}) + \omega(\log \lambda)$ communication between the client and the server except with probability negligible in λ. The client stores $O(1)$ private keys as well as at most $f(n) = \omega(1)$ key-value pairs. Both the client and server perform $O(\ell(\mathtt{key})) + \omega(\log \lambda)$ computation. Furthermore, the construction* dpMM *is lossy with probability negligible in λ.*

4.1 Discussion of Differential Privacy

In this section, we present a discussion of when differentially private volume-hiding suffices for security as opposed to volume-hiding. Deciding which security definition is completely dependent on the setting. Consider the case of input multi-maps drawn from a distribution of multi-maps with extremely different volume signatures, In this setting, volume-hiding seems necessary.

On the other hand, consider multi-maps that closely follow the Zipf distribution. Here, the volumes of the same key in different possible multi-maps will not differ significantly. Therefore, differentially private volume-hiding would suffice here. Additionally, suppose we wish to hide the identity of queried keys amongst other keys with similar volumes. In this case, differential privacy may also be used as the keys with similar volume will have similar leakage.

To summarize, differentially private volume-hiding is useful when all possible input multi-maps have similar volume signatures. Differentially private volume-hiding can protect input multi-maps that do not have very different volumes. When attempting to protect significantly different input multi-maps, normal volume-hiding security seems necessary. Therefore, the choice of whether to use dprfMM or dpMM depends on the situation.

5 Experimental Evaluation

In this section, we present our experimental evaluation of our two main volume-hiding STE schemes for multi-maps: dprfMM and dpMM. We start by describing the setup of our experiments as well as the choice of parameters of our constructions. Then, we compare both dprfMM and dpMM with the Densest Subgraph Transform construction described in [KM19].

By performing these experiments, we attempt to answer two important questions. First, are the constructions dprfMM and dpMM more concretely efficient compared to the previous most practically efficient construction (Densest Subgraph Transform [KM19])? Secondly, what is the total cost of dprfMM and dpMM?

We will use multipliers to describe improvements in efficiency. When we say construction A is a 2x improvement in communication over construction B, we mean that construction A uses half the communication used in construction B.

5.1 Experimental Setup

Our experiments are performed using the same machine for both the client and the server. The machines used for the client and server are Ubuntu PCs with 12 cores, 3.5 GHz Intel Xeon E5-1650 and 32 GB of RAM. All the results of our experiments have standard deviations less than 10% of their average. Network resources costs are measured at the application layer. Both the client and server binaries are built using the gRPC library [GRP18].

Input Multi-Maps. In our experiments, we will consider multi-maps containing $n \in \{2^{16}, 2^{18}, 2^{20}, 2^{22}\}$ total values. We note that all constructions under experimentation consider general multi-map inputs. As a result, their efficiencies do not depend on the structure of the multi-map but are completely determined by the parameters n and ℓ (the maximum volume) of the input multi-map. All keys and values will be 8 byte strings.

Primitives. In all our experiments, we consider PRFs with 16 byte keys as well as 16 byte outputs. In particular, we implement our PRFs using SHA256. We also use SHA256 as the pseudorandom generator for our delegatable PRF construction. We will use AES in CBC mode as our symmetric encryption scheme. Note that encrypting a single key-value pair which is 16 bytes would result in a 32 byte ciphertext. As a result, naive encryption of the input multi-map will result in a 2x overhead in storage.

5.2 Cost of Densest Subgraph Transform [KM19]

In the concrete parameters described in [KM19], it is shown that the number of bins must be at most $O(n/\log_2 n)$ to ensure that the storage overhead remains linear in the number of total values $O(n)$. Querying in this scheme is equivalent to downloading ℓ bins. Furthermore, each of the bins are padded to the size of the maximum bin. Therefore, it is beneficial to ensure that the bin size remains small. To do this, we can attempt to ensure that the number of bins is large. For practical constants, we pick the number of bins to be $2n/(\log_2 n)$ such that the average bin size will be $\log_2 n/2$. For a concrete EMM implementation, we use a similar delegatable PRF scheme as our own. As a result, the upload consists of a single 16 byte PRF output. Downloading a single result consists of downloading an entire bucket which is padded to the maximum bucket size. For each query, ℓ results will be downloaded to ensure that the construction is volume-hiding. The experimental results of this construction can be found in the first column of Table 2.

5.3 Cost of dprfMM

We implement the dprfMM construction in C++ using OpenSSL for all the underlying cryptographic primitives (SHA256 and AES). For the cuckoo hash table sizes, we will set the constant $\alpha = 0.3$ such that both tables hold a total $2.6n$ encrypted key-value pairs. We set the number of maximum evictions before being placed into the stash at $5 \log_2 n$. A query consists of uploading a single 16 byte PRF output and downloading two encrypted locations per result.

Comparing to the Densest Subgraph Transform [KM19], we see that dprfMM uses smaller server storage overall. Furthermore, as the multi-map increases size, dprfMM starts using significantly smaller amounts of server storage. For example, for the 2^{22} value multi-map, dprfMM uses approximately 40 MB less server storage. In terms of query overhead, dprfMM is significantly better. For each result, dprfMM downloads two encrypted locations resulting in only 64 bytes. This is a 10-16x improvement over the Densest Subgraph Transform [KM19] which requires 675 bytes (10 times more than dprfMM) per result for the multi-map with 2^{16} values and increases to 1 KB (16 times more than dprfMM) per result for the multi-map with 2^{22} values. Looking at the pattern, dprfMM will have even better query overhead improvements as we consider larger multi-maps. The only tradeoff that is made is that the client must store an additional stash. However, the stash is very small in practice consisting of at most 4 KB. This additional client storage is much smaller than the server storage gains of dprfMM.

In terms of CPU cost, we note that dprfMM is almost identical to the Densest Subgraph Transform [KM19]. The main difference is that during Reply, the Densest Subgraph Transform has to decrypt significantly larger strings as opposed to dprfMM. As a result, the CPU cost of dprfMM in Reply is smaller.

5.4 Cost of dpMM

We also implement dpMM in C++ using OpenSSL for all underlying cryptographic primitives. In particular, we modify dprfMM in two ways to obtain dpMM. First, dpMM additionally sends an integer during query as well as another PRF for retrieving an entry from the count table costing another 16 byte PRF value. For our experiments appearing in Table 2, we assume close to the worst case in our experiments for dpMM where the number of keys is $m = n/4$ which causes a very large cuckoo hashing table for counts.

The cost of slightly increased server storage for dpMM is offset by the main benefit of dpMM, which is that the total number of results that will be downloaded is much smaller than the other two constructions. To demonstrate these gains, we consider a natural setting of a multi-map of $n = 2^{20}$ values and $m = n/8 = 2^{17}$ keys following the Zipf distribution with parameters m and 1. As a result, the volumes of the multi-map from largest to smallest will be $(\frac{n}{H_m}, \ldots, \frac{n}{m \cdot H_m})$ where $H_m := \sum_{i \in [m]} i^{-1}$ is the harmonic number. In our case, $H_m \approx 12.36$ when $m = 2^{17}$.

We construct dpMM with a privacy budget of $\epsilon = 0.2$ for strong differential privacy guarantees. To pick the value $l^*(\lambda)$, we will ensure that dpMM is lossy with probability at most 2^{-64}. We know that $\Pr[\mathsf{Lap}(2/\epsilon) \leq 2t/\epsilon] \leq e^{-t}$. By setting $t := 56.1$, we can guarantee that the probability that a single key is truncated is $\Pr[\mathsf{Lap}(2/\epsilon) \leq 2t/\epsilon] \leq 2^{-64-17}$. Therefore, the probability that at least one of the $m = 2^{17}$ is truncated is at most 2^{-64}. By our choice, we set $l^*(\lambda) = 5610$. Note, we are being very pessimistic as we picked a small privacy budget of $\epsilon = 0.2$ and a small truncation probability of 2^{-64}. By choosing larger privacy budgets or larger truncation probabilities which might be suitable in certain settings, dpMM would incur smaller query overhead.

In Figure 1, we show the number of results downloaded by dpMM and dprfMM. It is clear that for all queried keys except the one with the largest volume, dpMM downloads significantly fewer results compared to dprfMM. Consider the average case where keys are chosen uniformly at random from the input multi-map following Zipf's distribution. In this case, the volume of the average key is 8. As a result, the average number of returned results for dpMM is 5618 while dprfMM must return more than 84,000 results (15 times more than dpMM). Since both dpMM and dprfMM communicate 64 bytes per result, dpMM is a 15x improvement

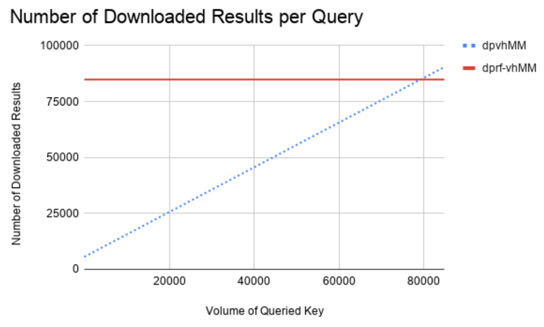

Figure 1: Comparison of number of downloaded results by dprfMM and dpMM.

in query communication over dprfMM. Additionally, we note that the Densest Subgraph Transform [KM19] also returns 84,000 results but each result requires communicating 841.6 bytes. As a result, dpMM is an 150-240x improvement in average query overhead over the Densest Subgraph Transform.

6 Conclusions

In this work, we present a volume-hiding scheme dprfMM that is practically more efficient than any previous volume-hiding scheme. dprfMM always uses less server storage compared to the previous construction. Furthermore, dprfMM improves the communication costs of queries by a factor of 10-16x over the previous best constructions [KM19] when encrypting multi-maps that occupy 1-67 MB in the plaintext and consist of 2^{16}-2^{22} total values. From an asymptotic perspective, dprfMM is the first construction with both asymptotically optimal storage and query overhead. We also present the first formal definition of volume-hiding leakage functions.

In addition, we also introduce the notion of *differentially private volume-hiding* which strikes a better, tunable balance between privacy and efficiency. We present dpMM that is able to significantly improve the average query overhead over the previous best volume-hiding schemes [KM19] by a factor of 150-240x when encrypting multi-maps with 2^{16}-2^{22} total values that occupy 1-67 MB in the plaintext.

Altogether, we significantly further the field of volume-hiding encrypted multi-maps by presenting both conceptual (formal definitions) and algorithmic (constructions that are both asymptotically and practically efficient) contributions.

References

[AKL+18] Gilad Asharov, Ilan Komargodski, Wei-Kai Lin, Kartik Nayak, Enoch Peserico, and Elaine Shi. OptORAMa: Optimal oblivious RAM. Cryptology ePrint Archive, Report 2018/892, 2018.

[AKM18] Ghous Amjad, Seny Kamara, and Tarik Moataz. Breach-resistant structured encryption. Cryptology ePrint Archive, Report 2018/195, 2018. https://eprint.iacr.org/2018/195.

[ANSS16] Gilad Asharov, Moni Naor, Gil Segev, and Ido Shahaf. Searchable symmetric encryption: optimal locality in linear space via two-dimensional balanced allocations. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 1101–1114. ACM, 2016.

[ASS18] Gilad Asharov, Gil Segev, and Ido Shahaf. Tight tradeoffs in searchable symmetric encryption. Cryptology ePrint Archive, Report 2018/507, 2018. https://eprint.iacr.org/2018/507.

[BBO07] Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. Deterministic and efficiently searchable encryption. In *CRYPTO '07*, 2007.

[BCLO09] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'neill. Order-preserving symmetric encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 224–241. Springer, 2009.

[BCO11] Alexandra Boldyreva, Nathan Chenette, and Adam O'Neill. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In *Annual Cryptology Conference*, pages 578–595. Springer, 2011.

[BDCOP04] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *EUROCRYPT '04*, pages 506–522, 2004.

[BMO17] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1465–1482. ACM, 2017.

[Bos16] Raphael Bost. Sophos: Forward secure searchable encryption. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1143–1154. ACM, 2016.

[CGKO11] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. *Journal of Computer Security*, 2011.

[CGPR15] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In *CCS '15*, 2015.

[CJJ$^+$13] David Cash, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Roşu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Annual Cryptology Conference*, pages 353–373. Springer, 2013.

[CJJ$^+$14] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Dynamic searchable encryption in very-large databases: data structures and implementation. In *NDSS*, volume 14, pages 23–26. Citeseer, 2014.

[CK10] Melissa Chase and Seny Kamara. Structured encryption and controlled disclosure. In *EUROCRYPT '10*, pages 577–594. Springer, 2010.

[CT14] David Cash and Stefano Tessaro. The locality of searchable symmetric encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 351–368. Springer, 2014.

[DMNS06] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography*, pages 265–284, 2006.

[DP17] Ioannis Demertzis and Charalampos Papamanthou. Fast searchable encryption with tunable locality. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1053–1067. ACM, 2017.

[DPP18] Ioannis Demertzis, Dimitrios Papadopoulos, and Charalampos Papamanthou. Searchable encryption with optimal locality: Achieving sublogarithmic read efficiency. In *Annual International Cryptology Conference*, pages 371–406. Springer, 2018.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, August 1986.

[GLMP] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. Learning to reconstruct: Statistical learning theory and encrypted database attacks. Cryptology ePrint Archive, Report 2019/011.

[GLMP18] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. Pump up the volume: Practical database reconstruction from volume leakage on range queries. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 315–331, 2018.

[GM11] Michael T. Goodrich and Michael Mitzenmacher. Privacy-preserving access of outsourced data via oblivious ram simulation. In *ICALP*, pages 576–587, 2011.

[GMN$^+$16] Paul Grubbs, Richard McPherson, Muhammad Naveed, Thomas Ristenpart, and Vitaly Shmatikov. Breaking web applications built on top of encrypted data. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1353–1364. ACM, 2016.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software Protection and Simulation on Oblivious RAMs. *J. ACM*, 43(3), 1996.

[Goh03] Eu-Jin Goh. Secure indexes. *IACR Cryptology ePrint Archive*, 2003:216, 2003.

[GRP18] gRPC - an RPC library and framework. `https://github.com/grpc/grpc`, 2018.

[GSB$^+$17] Paul Grubbs, Kevin Sekniqi, Vincent Bindschaedler, Muhammad Naveed, and Thomas Ristenpart. Leakage-abuse attacks against order-revealing encryption. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 655–672. IEEE, 2017.

[IKK12] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*, 2012.

[KKNO16] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. Generic attacks on secure outsourced databases. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1329–1340, 2016.

[KKNO17] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. Accessing data while preserving privacy. *CoRR*, abs/1706.01552, 2017.

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

[KM17] Seny Kamara and Tarik Moataz. Boolean searchable symmetric encryption with worst-case sub-linear complexity. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 94–124. Springer, 2017.

[KM18] Seny Kamara and Tarik Moataz. SQL on structurally-encrypted databases. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 149–180. Springer, 2018.

[KM19] Seny Kamara and Tarik Moataz. Computationally volume-hiding structured encryption. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019*, pages 183–213, 2019.

[KMO18] Seny Kamara, Tarik Moataz, and Olga Ohrimenko. Structured encryption and leakage suppression. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, pages 339–370, 2018.

[KMW09] Adam Kirsch, Michael Mitzenmacher, and Udi Wieder. More robust hashing: Cuckoo hashing with a stash. *SIAM J. Comput.*, 39(4):1543–1561, 2009.

[KPR12] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 965–976. ACM, 2012.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, CCS '13, pages 669–684, 2013.

[LMP18] Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. Improved reconstruction attacks on encrypted data using range query leakage. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 297–314, 2018.

[LN18] Kasper Green Larsen and Jesper Buus Nielsen. Yes, there is an Oblivious RAM lower bound! In *CRYPTO '18*, 2018.

[MM16] Ian Miers and Payman Mohassel. Io-dsse: Scaling dynamic searchable encryption to millions of indexes by improving locality. *IACR Cryptology ePrint Archive*, 2016:830, 2016.

[NKW15] Muhammad Naveed, Seny Kamara, and Charles V. Wright. Inference attacks on property-preserving encrypted databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 644–655, 2015.

[NPG14] Muhammad Naveed, Manoj Prabhakaran, and Carl A Gunter. Dynamic searchable encryption via blind storage. In *2014 IEEE Symposium on Security and Privacy*, pages 639–654. IEEE, 2014.

[PPRY18] Sarvar Patel, Giuseppe Persiano, Mariana Raykova, and Kevin Yeo. PanORAMa: Oblivious RAM with logarithmic overhead. In *FOCS '18*, 2018.

[PPY17] Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. Symmetric searchable encryption with sharing and unsharing. *IACR Cryptology ePrint Archive*, 2017:973, 2017.

[PR04] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *J. Algorithms*, 51(2):122–144, 2004.

[PRZB11] Raluca Ada Popa, Catherine Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 85–100. ACM, 2011.

[PW16] David Pouliot and Charles V Wright. The shadow nemesis: Inference attacks on efficiently deployable, efficiently searchable encryption. In *CCS '16*, 2016.

[PY19] Giuseppe Persiano and Kevin Yeo. Lower bounds for differentially private rams. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019*, volume 11476, pages 404–434, 2019.

[SPS14] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. Practical dynamic searchable encryption with small leakage. In *NDSS*, volume 71, pages 72–75, 2014.

[SvDS$^+$13] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: An Extremely Simple Oblivious RAM Protocol. In *CCS '13*, pages 299–310, 2013.

[SWP00] D. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000*, pages 44–55, 2000.

[ZKP16] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In *USENIX Security Symposium*, pages 707–720, 2016.

A Security Proof of vhMM

Proof of Lemma 1. We construct a stateful simulator $\mathcal{S}$ for both Setup and Query.
$(\mathsf{state}, \mathsf{EMM}) \leftarrow \mathcal{S}.\mathsf{SimSetup}(1^\lambda, \mathsf{dsize}(\mathsf{MM}))$**:**

1. Set $n \leftarrow \mathsf{dsize}(\mathsf{MM})$.
2. Construct two empty arrays T_1, T_2 of length $t = (1+\alpha)n$.
3. Fill each array location with a uniformly random value from $\{0,1\}^\lambda$.
4. Return $(\perp, (T_1, T_2))$.

$(\mathsf{state}, \mathsf{Response}) \leftarrow \mathcal{S}.\mathsf{SimQuery}(1^\lambda, \mathsf{qeq}(\mathtt{key}_1, \ldots, \mathtt{key}_i), \mathsf{mrlen}(\mathsf{MM}))$**:**

1. Using $\mathsf{qeq}(\mathtt{key}_1, \ldots, \mathtt{key}_i)$, determine if $\mathtt{key}_i = \mathtt{key}_j$ for some $j < i$.
2. If $\mathtt{key}_i = \mathtt{key}_j$ for some $j < i$:
   (a) Return $(\mathsf{state}, \mathsf{state}[\mathtt{key}_j])$.
3. Suppose $\mathtt{key}_i$ is unique from all previous queried keys:
   (a) Set $\mathsf{Response} \leftarrow (r_1, \ldots, r_{2\ell})$ where each r_i is drawn uniformly at random from $\{0,1\}^\lambda$.
   (b) Set $\mathsf{state}[\mathtt{key}_i] \leftarrow \mathsf{Response}$.
   (c) Return $(\mathsf{state}, \mathsf{Response})$.

We now show that for all PPT adversaries $\mathcal{A}$, the probability $\mathbf{Real}_{\mathsf{vhMM},\mathcal{A}}(1^\lambda)$ outputs 1 is negligibly different from the probability that $\mathbf{Ideal}_{\mathsf{vhMM},\mathcal{A},\mathcal{S}}(1^\lambda)$ outputs 1. To do this, we use the following sequence of games:

- $\mathbf{Game}_0$ is identical to $\mathbf{Real}_{\mathsf{vhMM},\mathcal{A}}(1^\lambda)$.
- $\mathbf{Game}_1$ remove Step 5 from $\mathsf{vhMM.Setup}$.
- $\mathbf{Game}_2$ replaces the IND-CPA encryption scheme Enc with a random function.
- $\mathbf{Game}_3$ replaces the outputs of both random functions with uniformly random chosen values.
- $\mathbf{Game}_4$ replaces the PRF F with a random function.

Note $\mathbf{Game}_0$ and $\mathbf{Game}_1$ are only negligibly-distinguishable by a computational adversary as cuckoo hashing with a stash fails with negligible probability when $f(n) = \omega(1)$. It can be shown that $\mathbf{Game}_4$ is identical to $\mathbf{Ideal}_{\mathsf{vhMM},\mathcal{A},\mathcal{S}}(1^\lambda)$. □

Proof of Lemma 2. To prove that $\mathcal{L} = (\mathsf{dsize}, (\mathsf{qeq}, \mathsf{mrlen}))$ is volume-hiding, we consider any two multi-map signatures with the same number of values n and maximum volume ℓ. Note, $\mathcal{L}_{\mathsf{Setup}}$ for vhMM consists of $\mathsf{dsize}(\mathsf{MM}) = n$. Similarly, $\mathcal{L}_{\mathsf{Query}}$ consists of only $\mathsf{mrlen}(\mathsf{MM}) = \ell$ and query equality for the queried keys. The leakage of dsize and mrlen are identical for both signatures. Query equality leakage is independent of the input multi-maps. As a result, the input to the adversary in both games with different signatures is identical completing the proof. □

B Security Proof of dprfMM

Proof of Theorem 6. The stateful simulator for construction dprfMM is identical to vhMM for $\mathsf{SimSetup}$. We now present $\mathsf{SimQuery}$ for dprfMM.

$\mathcal{S}.\mathsf{SimQuery}(1^\lambda, \mathsf{qeq}(\mathtt{key}_1, \ldots, \mathtt{key}_i), \mathsf{mrlen}(\mathsf{MM}))$**:**

1. Using $\mathsf{qeq}(\mathtt{key}_1, \ldots, \mathtt{key}_i)$, determine if $\mathtt{key}_i = \mathtt{key}_j$ for some $j < i$.
2. If $\mathtt{key}_i = \mathtt{key}_j$ for some $j < i$:
   (a) Return $(\mathsf{state}, \mathsf{state}[\mathtt{key}_j])$.
3. Suppose $\mathtt{key}_i$ is unique from all previous queried keys:
   (a) Set $\mathsf{Response} \leftarrow r$ where each r is drawn uniformly at random from $\{0,1\}^\lambda$.
   (b) Set $\mathsf{state}[\mathtt{key}_i] \leftarrow \mathsf{Response}$.
   (c) Return $(\mathsf{state}, \mathsf{Response})$.

Note the only difference is that $\mathsf{Response}$ consists of a single random value as opposed to 2ℓ random values.

The sequence of games is identical except that the dPRF (instead of the PRF) is replaced by a random function. The two games being computationally indistinguishable follows from the proof of security in [KPTZ13]. ☐

Proof of Theorem 7. The leakage of dprfMM and vhMM are identical. As a result, the proof follows identically to Theorem 2. ☐

C Security Proof of dpMM

Before proving Theorem 9, we note that extra $l^*(\lambda) + \mathsf{noise}(\mathtt{key})$ downloads are performed by the client, where $\mathsf{noise}(\mathtt{key}) \leftarrow \mathsf{Lap}(2/\epsilon)$. We stress though that the same value of $\mathsf{noise}(\mathtt{key})$ must be used for each query for $\mathtt{key}$ and this can be achieved in two possible ways:

1. We consider the sanitizer San_ϵ that, for each $\mathtt{key}$ of multi-set $\mathcal{D}$, samples $\mathsf{noise}(\mathtt{key}) \leftarrow \mathsf{Lap}(2/\epsilon)$ and adds $l^*(\lambda) + \mathsf{noise}(\mathtt{key})$ dummy values to $\mathtt{key}$, before executing Setup over it.

   This approach has the drawback of a multiplicative overhead of $\omega(\log N)$ for server storage, for a set of size $N = \mathtt{poly}(\lambda)$.

2. We do not sanitize the multi-set $\mathcal{D}$ before executing Setup but each time we query for $\mathtt{key}$ we sample $\mathsf{noise}(\mathtt{key})$. To ensure that for every query of $\mathtt{key}$ we obtain the same value of $\mathsf{noise}(\mathtt{key})$, we fix the random bits to be used by the sampling algorithm as the output of a PRF whose seed is part of the client secret key.

We stress that, as far as differential privacy and the volume-hiding property are concerned, the two approaches are equivalent. So we use the second approach in our construction and the first one in our proof.

The proof of Theorem 9 consists of two parts. First, we prove that our construction is $\mathcal{L}$-secure for an appropriate leakage function $\mathcal{L}$ and then we show that $\mathcal{L}$ is (ϵ, δ)-differentially private volume-hiding when coupled with San_ϵ.

We consider the leakage function $\mathcal{L} = (\mathsf{dsize}, (\mathsf{qep}, \mathsf{dprlen}))$, where $\mathsf{dprlen}(\mathcal{D}, \mathtt{key}_1, \ldots, \mathtt{key}_i)$ is simply rlen, the response length leakage function, applied to the sanitized multi-map $\mathsf{San}(\mathcal{D})$. The proof of $\mathcal{L}$-security follows similarly to the proof of security of vhMM for any San_ϵ.

We next prove that, for every ϵ, $(\mathsf{San}_\epsilon, \mathcal{L})$ is $(\epsilon, 0)$-differential privacy volume-hiding. We remind the reader that sanitizer San_ϵ, described at bullet (1) above, adds extra $l^*(\lambda) + \mathsf{noise}(\mathtt{key})$ values for $\mathtt{key}$, where $\mathsf{noise}(\mathtt{key}) \leftarrow \mathsf{Lap}(2/\epsilon)$.

We use the following well known property of the Laplacian distribution. Specifically, let us denote by h the density function of the Laplacian distribution with parameter γ. Then, for every y, y' we have that

$$h(y) \leq h(y') \cdot e^{\frac{|y-y'|}{\gamma}}.$$

For an adversary $\mathcal{A}$, we denote by $\mathcal{T}_{\mathcal{A}}(S)$ the random variable of the leakage transcript obtained by $\mathcal{A}$ in the game $\mathbf{dpVH}^{\mathcal{A},\mathcal{L},\mathsf{San}_\epsilon}$ on input signature S. We will show that for any transcript $T = (t_1, \ldots, t_q)$ and for any two neighboring signatures, S_0 and S_1,

$$\text{Prob}\left[\mathcal{T}_{\mathcal{A}}(S_0) = T\right] \leq e^{\epsilon} \cdot \text{Prob}\left[\mathcal{T}_{\mathcal{A}}(S_1) = T\right].$$

This will suffice to prove differential privacy.

First of all, we observe that if query $Q_i = Q_j$ and $t_i \neq t_j$, then

$$\text{Prob}\left[\mathcal{T}_{\mathcal{A}}(S_0) = T\right] = \text{Prob}\left[\mathcal{T}_{\mathcal{A}}(S_1) = T\right] = 0.$$

We can thus restrict ourselves to transcripts that are *consistent* with $\mathcal{Q}$; that is, transcripts for which $Q_i = Q_j$ implies $t_i = t_j$. For $b = 0, 1$, we write

$$\text{Prob}\left[\mathcal{T}_{\mathcal{A}}(S_b) = T\right] = \prod_{i=1}^{q} \text{Prob}\left[\mathcal{T}_{\mathcal{A}}(S_b)_i = t_i \mid t_1, \ldots, t_{i-1}\right]$$

where $\mathcal{T}_{\mathcal{A}}(S_b)_i$ denotes the i-th component of the random variable of the transcript. We also denote by $\mathsf{noise}_0(\mathtt{key})$ and $\mathsf{noise}_1(\mathtt{key})$ the random variable relative to the distributions $\mathcal{T}_{\mathcal{A}}(S_0)$ and $\mathcal{T}_{\mathcal{A}}(S_1)$, respectively. Consider the following cases.

1. If S_0 and S_1 do not differ for key Q_i then

$$\text{Prob}\left[\mathcal{T}_{\mathcal{A}}(S_0)_i = t_i \mid t_1, \ldots, t_{i-1}\right] = \text{Prob}\left[\mathcal{T}_{\mathcal{A}}(S_1)_i = t_i \mid t_1, \ldots, t_{i-1}\right].$$

2. Suppose that S_0 and S_1 differ for key $Q_i = \mathtt{key}$ (that is, $\ell_0(\mathtt{key}) \neq \ell_1(\mathtt{key})$) and suppose that the i-th query is the first query for $\mathtt{key}$; that is, for all $j < i$, $Q_j \neq \mathtt{key}$. Then,

$$\begin{aligned} &\text{Prob}\left[\mathcal{T}_{\mathcal{A}}(S_0)_i = t_i \mid t_1, \ldots, t_{i-1}\right] = \text{Prob}\left[\mathcal{T}_{\mathcal{A}}(S_0)_i = t_i\right] \\ &\quad = \text{Prob}\left[\mathsf{noise}_0(\mathtt{key}) = t_i - \ell_0(\mathtt{key}) - \ell^*(\lambda)\right] \\ &\quad \leq e^{\frac{\epsilon}{2}|\ell_0(\mathtt{key}) - \ell_1(\mathtt{key})|} \cdot \text{Prob}\left[\mathsf{noise}_1(key) = t_i - \ell_1(\mathtt{key}) - \ell^*(\lambda)\right] \\ &\quad \leq e^{\frac{\epsilon}{2}} \cdot \text{Prob}\left[\mathsf{noise}_1(\mathtt{key}) = t_i - \ell_1(\mathtt{key}) - \ell^*(\lambda)\right], \\ &\qquad (\text{since } |\ell_0(\mathtt{key}) - \ell_1(\mathtt{key})| = 1) \\ &\quad = e^{\frac{\epsilon}{2}} \cdot \text{Prob}\left[\mathcal{T}_{\mathcal{A}}(S_1)_i = t_i \mid t_1, \ldots, t_{i-1}\right] \end{aligned}$$

3. If instead S_0 and S_1 differ for key $Q_i = \mathtt{key}$ but for some $j < i$, $Q_j = \mathtt{key}$, then, by the consistency of the transcript, we have

$$\text{Prob}\left[\mathcal{T}_{\mathcal{A}}(S_b)_i = t_i \mid t_1, \ldots, t_{i-1}\right] = 1,$$

for $b = 0, 1$.

Since S_0 and S_1 differ for at most two keys we have that

$$\text{Prob}_{\mathcal{A}}\left[\mathcal{T}(S_0) = T\right] = \leq e^{\epsilon} \cdot \text{Prob}_{\mathcal{A}}\left[\mathcal{T}(S_1) = T\right].$$