Computationally Volume-Hiding Structured Encryption

Seny Kamara and Tarik Moataz

Brown University, Providence, USA
`seny@brown.edu, tarik_moataz@brown.edu`

Abstract. We initiate the study of structured encryption schemes with computationally-secure leakage. Specifically, we focus on the design of volume-hiding encrypted multi-maps; that is, of encrypted multi-maps that hide the response length to computationally-bounded adversaries. We describe the first volume-hiding STE schemes that do not rely on naïve padding; that is, padding all tuples to the same length. Our first construction has efficient query complexity and storage but can be lossy. We show, however, that the information loss can be bounded with overwhelming probability for a large class of multi-maps (i.e., with lengths distributed according to a Zipf distribution). Our second construction is not lossy and can achieve storage overhead that is asymptotically better than naïve padding for Zipf-distributed multi-maps. We also show how to further improve the storage when the multi-map is highly *concentrated* in the sense that it has a large number of tuples with a large intersection. We achieve these results by leveraging computational assumptions; not just for encryption but, more interestingly, to hide the volumes themselves. Our first construction achieves this using a pseudo-random function whereas our second construction achieves this by relying on the conjectured hardness of the planted densest subgraph problem which is a planted variant of the well-studied densest subgraph problem. This assumption was previously used to design public-key encryptions schemes (Applebaum et al., *STOC '10*) and to study the computational complexity of financial products (Arora et al., *ICS '10*).

1 Introduction

A structured encryption (STE) scheme encrypts a data structure in such a way that it can be privately queried. An STE scheme is secure if it reveals nothing about the structure and query beyond a well-specified and "reasonable" leakage profile [15,12]. An important special case of STE is searchable symmetric encryption (SSE) which relies on encrypted multi-maps [15,12,29,28,36,11,10,7,8,4,18,5,16] to achieve optimal-time search. Another example is graph encryption which encrypts various kinds of graphs [12,33]. STE has received a lot of attention due to its potential applications to cloud storage and database security. In recent years, much of the work on STE has focused on supporting more complex queries like Boolean [11,37,21,25] and range queries [37,21,20,38], more complex structures like relational databases [26] and on improving security, for example achieving forward-privacy [40,7,8,19,1].

Leakage. One aspect of STE that is still poorly understood is its leakage. There are currently two approaches to dealing with leakage. The first is cryptanalysis; that is, designing leakage attacks against various leakage profiles so that we can better understand their concrete security. This was initiated by Islam, Kuzu and Kantarcioglu in the context of SSE [24] and expanded to PPE by Naveed, Kamara and Wright [35] and to ORAM by Kellaris, Kollios, Nissim and O'Neill [30]. While there has been some progress on designing leakage attacks against STE [24,9,30,32], these attacks remain mostly of theoretical interest due to the strong assumptions they rely on. Assumptions like knowledge of at least $80\% - 90\%$ of client data in addition to knowledge of 5% of client queries [24,9], or assuming clients make queries uniformly at random, often in addition to assumptions about how client data is distributed [30,32]. Nevertheless, these attacks do provide us with some guidance as to which leakage profiles to avoid when designing schemes. Another line of work related to leakage was initiated recently by Kamara, Moataz and Ohrimenko in [27] where they propose designing general-purpose techniques to suppress specific leakage patterns. In [27], they show how to do this for the query equality pattern (also known as the search pattern) without making use of ORAM simulation and, therefore, without incurring its poly-logarithmic multiplicative overhead.

Computationally-secure leakage. In this work, we consider a new approach to dealing with leakage. Our work starts from the observation that the presence of leakage does not necessarily imply that this leakage can be exploited. In fact, it could be that the leakage is not exploitable because it does not convey enough useful information to the adversary. Alternatively, it could be that the leakage does convey enough information but no computationally-bounded adversary can extract it. In other words, the leakage could be computationally-secure. The possibility of designing STE schemes with computationally-secure leakage patterns is interesting for several reasons. From a theoretical point of view, as far as we know, this question has never been considered before and it raises some intriguing foundational questions; like what kind of computational assumptions would lend themselves to the design of secure leakage patterns? The traditional assumptions used in cryptography are usually algebraic or number-theoretic in nature and it is not clear how such assumptions could be used. From a more practical perspective, the ability to leverage "computationally-secure leakage" in the design of STE schemes could lead to a whole new set of techniques and, ultimately, to highly-efficient zero- or low-leakage schemes—computationally speaking.

Volume-hiding EMMs. In this work, we initiate the study of computationally-secure leakage. In particular, we focus on the design of volume-hiding encrypted multi-maps or, more precisely, of encrypted multi-maps that hide the response length to computationally-bounded adversaries. [1] We focus on encrypted multi-maps because they are by far the most important encrypted structure; this is illustrated by the fact that they are central to the design of optimal-time single-keyword SSE [15,12,29,10,7,34,8,19], of sub-linear Boolean SSE [11,25], of graph

[1] Our constructions also reveal the query equality—even to a bounded adversary—but the latter can be suppressed using the cache-based transform from [27].

encryption [12,33], of encrypted range structures [20,17] and of encrypted relational databases [26]. We consider the response length leakage pattern for several reasons. The first is that it is a very difficult leakage pattern to suppress. In fact, though encrypted search has been investigated since 2000, the first non-trivial construction to even partially hide the response length is the recent PBS scheme of [27]. [2] In fact, response lengths are leaked even by ORAM-based solutions. The second reason we focus on response lengths is because of the recent volume attacks of Kellaris et al. [30] or its extension by Grubbs et al. [23]. Again, while these attacks are mostly of theoretical interest, they do suggest that the design of volume-hiding encrypted structures is well-motivated.

1.1 Naïve Approaches

To better understand our techniques and the improvements they provide, we first describe two possible naïve approaches to designing volume-hiding EMMs. Recall that a multi-map is a data structure that stores a set of pairs $\{(\ell, \mathbf{v})\}$, where ℓ is a label from a label space $\mathbb{L}$ and $\mathbf{v}$ is a tuple of values from some value space $\mathbb{V}$. Multi-maps support get and put operations. Get takes as input a label ℓ and returns its associated tuple $\mathbf{v}$ whereas Put takes as input a label/value pair $(\ell, \mathbf{v})$ and stores it. We denote the get operation by $\mathbf{v} := \mathsf{MM}[\ell]$ and the put operation by $\mathsf{MM}[\ell] := \mathbf{v}$.

Naïve padding. The first approach to designing a volume-hiding multi-map encryption scheme is to pad the tuples of the plaintext multi-map MM to their maximum response length $t = \max_{\ell \in \mathbb{L}} \#\mathsf{MM}[\ell]$ and encrypt the padded multi-map with any standard multi-map encryption scheme [12,10,7,1]. It is easy to see that this hides the response lengths. Unfortunately, it also induces a non-trivial storage overhead.

Using ORAM. We now describe a volume-hiding construction based on ORAM. Note that, as far as we know, this construction has not appeared before and may be of independent interest.[3] We first represent the multi-map MM as a dictionary by generating $N \overset{def}{=} \sum_{\ell \in \mathbb{L}} \#\mathsf{MM}[\ell]$ pairs of the form $\left\{(\ell, v)_{\ell \in \mathbb{L}, v \in \mathsf{MM}[\ell]}\right\}$ and storing them in a dictionary DX. We then add $t-1$ dummy label/value pairs to DX, where t is the maximum response length of a label in MM. DX is then stored and managed using ORAM. To get the tuple associated with a label ℓ, we first obliviously access DX. There are two cases: if $\#\mathsf{MM}[\ell] = t$, then we retrieve all pairs associated with ℓ; otherwise if $\#\mathsf{MM}[\ell] < t$, we retrieve an additional $t - \#\mathsf{MM}[\ell]$ dummies.

It is clear that this hides the response length since the ORAM simulation hides the query equality and, therefore, an adversary can't distinguish between

[2] The PBS construction has two variants. One can hide the response length on non-repeating sub-patterns but has a probability of failure in the sense that the client might not receive all its query responses. The second variant is always correct but reveals the sequence response length on non-repeating sub-patterns.

[3] Kellaris, Kollios, Nissim and O'Neil show in [31] how to use differential privacy to perturb the response length in ORAM. This is different from this naïve approach which completely hides the response length.

a dummy label and a real label. From an efficiency perspective, if we use a state-of-the-art ORAM [41] then the storage overhead is $O(N)$. The communication complexity, however, is $O(t \cdot \log^2 N)$ which includes a multiplicative poly-logarithmic factor in addition to logarithmic round complexity.

1.2 Our Techniques and Contributions

In this work, we describe two volume-hiding multi-map encryption schemes: VLH and AVLH. Both our constructions work by first transforming an input multi-map into a volume-hiding multi-map and encrypting the result with a custom multi-map encryption scheme that itself makes black-box use of a standard multi-map encryption scheme. These constructions avoid the limitations of the naïve approaches described above either by improving on the storage of naïve padding or avoiding the multiplicative poly-logarithmic overhead of the ORAM-based solution.

A time-efficient construction. Our first construction relies on a simple transformation we call the *pseudo-random transform* which is parameterized by a public parameter λ and makes use of a small-domain pseudo-random function as follows. Each tuple $\mathbf{v}$ in the multi-map is transformed into a new tuple $\mathbf{v}'$ of size $n' = \lambda + F_K(n)$, where $n = \#\mathbf{v}$. If $n' > n$, then the elements of $\mathbf{v}$ are stored in $\mathbf{v}'$ and the latter is padded to have length n'. If $n' \leq n$ then only the first n' items of $\mathbf{v}$ are stored in $\mathbf{v}'$ which effectively truncates $\mathbf{v}$ (we think of the case $n' = n$ as a padding). Note that the multi-map that results from this process is volume-hiding since each tuple has pseudo-random length. Perhaps surprisingly, we also show that if the lengths of the input multi-map are Zipf-distributed then the storage overhead and the number of truncations can be kept relatively small with overwhelming probability in the number of labels. More precisely, we show that the storage overhead is half that of naïve padding while the number of truncations is equal to $m/\log m$. Our scheme VLH essentially consists of transforming a multi-map using the pseudo-random transform and encrypting it with a standard multi-map encryption scheme. The query complexity of VLH is $O(\lambda + \nu)$, where ν is the largest value in the domain of F. While the pseudo-random transform leads to an efficient construction, it is lossy since tuples can be truncated. In many practical settings, however, truncations are not necessarily an issue. For example, in the case of SSE where EMMs are used to store document identifiers clients can rank the document ids (say, by relevance) at setup time so that truncations only affect the low-ranked documents. Nevertheless, we also consider the problem of designing non-lossy volume-hiding EMMs.

A non-lossy transform. Our second construction relies on a different transformation we call the *dense subgraph transform*. Unlike the pseudo-random transform which introduces truncations, this approach is non-lossy. On the other hand, it is less efficient in terms of query complexity. Note, however, that it is hard to imagine any non-lossy construction being able to hide the response length of a query and having query complexity $o(t)$, where t is the maximum response length. Our goal, therefore, is to design a non-lossy scheme that improves on the *storage overhead* of the naïve padding approach. At a high-level, our non-lossy transform works by re-arranging the data stored in the multi-map into bins according to a random bi-partite graph. Roughly speaking, we construct a random (regular) bi-partite graph with labels in one set and bins in the other. We then assign the values in a label's tuple to the bins that are incident to the label. The bins are then padded to hide their size. To ensure that this re-arrangement is still efficiently queryable, we show how to represent the structure encoded in the bi-partite graph and the data stored in the bins with a pair of standard data structures; specifically, a multi-map and a dictionary. We show that, with the right choice of parameters, this version of our transformation already yields a volume-hiding multi-map structure with better storage overhead than the naïve padding approach. More precisely, we show that the naïve approach produces a volume-hiding multi-map of size $S_{\mathsf{NV}} = \Omega(N)$, where N is the size of the original multi-map, whereas our approach yields a volume-hiding multi-map of size $O(N)$ with overwhelming probability in N. Interestingly, we also show that if the tuple-lengths of the input multi-map are Zipf-distributed then our transformation yields a multi-map of size $o(S_{\mathsf{NV}})$ with overwhelming probability. We note that this version of the transformation already makes use of computational assumptions. In particular, it uses a pseudo-random function to generate the edges of the random bi-partite graph which allows us to "compress" the size of our data structures by storing random seeds as opposed to all the graph's edges. To query our transformed multi-map on some label ℓ, it suffices to retrieve the bins incident to ℓ. Intuitively, this is volume-hiding because the bins are padded and the number of bins is fixed. Furthermore, it hides other leakage patterns because the tuple values are assigned to bins randomly.

Concentration and planted subgraphs. The version of the transformation described so far already improves over naïve padding (with overwhelming probability) but we show that for a certain class of multi-maps we can do even better—though at the cost of increased query complexity. Specifically, we consider multi-maps that have a large number of tuples with a large intersection. We refer to this property as *concentration* and describe a version of the dense subgraph transform that leverages the multi-map's concentration to improve storage efficiency even more. At a high-level, the idea is as follows. A concentrated multi-map has a number of redundant values which our transformation assigns to multiple bins. In our improved transform, we instead assign each of these redundant values to a single bin and add edges between these bins and a large subset of the labels whose tuples they appear in. The rest of the bi-partite graph is generated (pseudo-)randomly as above. This has the benefit of inducing smaller bins and, therefore, of requiring less padding. The bi-partite graph, however, is not random anymore (even ignoring our use of a pseudo-random function to generate edges). We observe, however, that by adding the edges to the bins of the redundant values, we are effectively *planting* a small dense subgraph inside of a larger random graph. And while the resulting graph is clearly not random anymore, it can be shown to be computationally indistinguishable from a random graph. In fact, this reduces to the *planted densest subgraph problem* which has been used in the past by Applebaum et al. in the context of cryptography

[2] and by Arora et al. in the context of computational complexity [3]. Based on this assumption, we can show that for multi-maps with concentration parameters within a certain range (in turn determined by the densest subgraph assumption) the transformed multi-map is of size $O\big(N - m^{0.5+\delta} \cdot \text{polylog}(m)\big)$ with overwhelming probability, where m is the number of labels in the original multi-map and $\delta \geq 0$. If the input multi-map is Zipf-distributed, then the output multi-map has size $o(S_{\mathsf{NV}})$.

Our non-lossy construction. As mentioned above, the dense subgraph transform produces multi-maps that we represent using a combination of a dictionary and a standard multi-map. To encrypt this particular representation, we design a new scheme called AVLH. The resulting construction has query complexity $O\left(t \cdot \frac{N - m^{0.5+\delta} \cdot \text{polylog}(m)}{m \cdot \text{polylog}(m)}\right)$ for multi-maps with concentration parameters within a certain range.

Dynamism. Our VLH and AVLH constructions are for static multi-maps. While there are many important applications of static EMMs, we describe how to extend these constructions to handle updates. This results in two additional constructions, $\mathsf{VLH}^{\mathsf{d}}$ and $\mathsf{AVLH}^{\mathsf{d}}$. The former handles three kinds of updates: tuple addition, tuple deletion and tuple edits; and the latter handles tuple edits.

1.3 Related Work

Structured encryption was introduced by Chase and Kamara [12] as a generalization of searchable symmetric encryption which was first considered by Song, Wagner and Perrig [39] and formalized by Curtmola, Garay, Kamara and Ostrovsky [15]. Multi-map encryption schemes are a special case of STE and have been used to achieve optimal-time single-keyword SSE [15,29,10,7,19,8], sublinear Boolean SSE [11,25], encrypted range search [20,17], encrypted relational databases [26] and graph encryption [12,33]. The first leakage attack against volume leakage was described by Kollios, Kellaris, Nissim and O'Neill [30] under the assumption of uniform query distributions. In [27], Kamara, Moataz and Ohrimenko describe an STE scheme called PBS which partially hides the volume pattern. More precisely, the first variant of PBS reveals only the sequence response length (i.e., the sum of the response lengths of a given query sequence) on non-repeating query sequences. The second variant reveals nothing (beyond a public parameter independent of the volume) on non-repeating query sequences. While there are schemes that hide the response length at setup time [42] or use differential privacy to perturb response lengths [31], our techniques hide the pattern entirely at query time. The planted densest graph problem was first used as a computational assumption by Applebaum, Barak and Wigderson in [2] for the purpose of designing public-key encryption schemes under new assumptions. It was later used by Arora, Barak, Brunnermeier and Ge to study the computational complexity of financial products [3].

2 Preliminaries

Notation. The set of all binary strings of length n is denoted as $\{0,1\}^n$, and the set of all finite binary strings as $\{0,1\}^*$. We write $x \leftarrow \chi$ to represent an element x being sampled from a distribution χ, and $x \overset{\$}{\leftarrow} X$ to represent an element x being sampled uniformly at random from a set X. The output x of an algorithm $\mathcal{A}$ is denoted by $x \leftarrow \mathcal{A}$. Given a sequence $\mathbf{v}$ of n elements, we refer to its ith element as v_i or $\mathbf{v}[i]$. If S is a set then $\#S$ refers to its cardinality and 2^S to its powerset.

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

3 Definitions

Structured encryption schemes encrypt data structures in such a way that they can be privately queried. There are several natural forms of structured encryption. The original definition of [12] considered schemes that encrypt both a structure and a set of associated data items (e.g., documents, emails, user profiles etc.). In [13], the authors also describe *structure-only* schemes which only encrypt structures. Another distinction can be made between *interactive* and *non-interactive* schemes. Interactive schemes produce encrypted structures that are queried through an interactive two-party protocol, whereas non-interactive schemes produce structures that can be queried by sending a single message, i.e, the token. One can also distinguish between *response-hiding* and *response-revealing* schemes: the former reveal the response to queries whereas the latter do not. We recall here the syntax of an interactive response-hiding structured encryption scheme.

Definition 1 (Structured encryption). *An interactive response-hiding structured encryption scheme* $\Sigma_{\mathsf{DS}} = (\mathsf{Setup}, \mathsf{Query})$ *for data type* DS *consists of the following polynomial-time algorithms and protocols:*

– $(K, \mathsf{EDS}) \leftarrow \mathsf{Setup}_{\mathbf{C}}(1^k, \mathsf{DS})$*: is a probabilistic algorithm that takes as input a security parameter* 1^k *and a structure* DS *of type* DS *and outputs a secret key* K *and an encrypted structure* EDS*.*
– $(r, \perp) \leftarrow \mathsf{Query}_{\mathbf{C},\mathbf{S}}(\mathsf{tk}; \mathsf{EDS})$*: is an interactive protocol executed between a client* $\mathbf{C}$ *and a server* $\mathbf{S}$*. The client inputs a token* tk *and the server inputs an encrypted structure* EDS*. The client receives a response* r *and the server receives* $\perp$*.*

We refer the reader to, for example [1], for syntax definitions of dynamic STE.

Security. The standard notion of security for STE guarantees that: (1) an encrypted structure reveals no information about its underlying structure beyond the setup leakage $\mathcal{L}_{\mathsf{S}}$; (2) that the query protocol reveals no information about the structure and the queries beyond the query leakage $\mathcal{L}_{\mathsf{Q}}$. If this holds for non-adaptively chosen operations then the scheme is said to be non-adaptively secure. If, on the other hand, the operations can be chosen adaptively, the scheme is said to be adaptively-secure.

Definition 2 (Adaptive security of interactive STE). *Let* $\Sigma = (\mathsf{Setup}, \mathsf{Query})$ *be an interactive STE scheme and consider the following probabilistic experiments where* $\mathcal{A}$ *is a stateful semi-honest adversary,* $\mathcal{S}$ *is a stateful simulator,* $\mathcal{L}_{\mathsf{S}}$ *and* $\mathcal{L}_{\mathsf{Q}}$ *are leakage profiles and* $z \in \{0,1\}^*$*:*

$\mathbf{Real}_{\Sigma,\mathcal{A}}(k)$*: given* z *the adversary* $\mathcal{A}$ *outputs a structure* DS *and receives* EDS *from the challenger, where* $(K, \mathsf{EDS}) \leftarrow \mathsf{Setup}(1^k, \mathsf{DS})$*. The adversary then* adaptively *chooses a polynomial number of queries and, for each, executes the* Query *protocol with the challenger, where the adversary plays the server and the challenger plays the client. Finally,* $\mathcal{A}$ *outputs a bit* b *that is output by the experiment.*

$\mathbf{Ideal}_{\Sigma,\mathcal{A},\mathcal{S}}(k)$*: given* z *the adversary* $\mathcal{A}$ *generates a structure* DS *which it sends to the challenger. Given* z *and leakage* $\mathcal{L}_{\mathsf{S}}(\mathsf{DS})$ *from the challenger, the simulator* $\mathcal{S}$ *returns an encrypted structure* EDS *to* $\mathcal{A}$*. The adversary then* adaptively *chooses a polynomial number of queries and, for each one, executes the* Query *protocol with the simulator, where the adversary plays the server and the simulator plays the client (note that here, the simulator is allowed to deviate from* Query*). Finally,* $\mathcal{A}$ *outputs a bit* b *that is output by the experiment.*

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

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

Modeling leakage. Every STE scheme is associated with leakage which itself can be composed of multiple *leakage patterns*. The collection of all these leakage patterns forms the scheme's *leakage profile*. Leakage patterns are (families of) functions over the various spaces associated with the underlying data structure. For concreteness, we borrow the nomenclature introduced in [27] and recall some well-known leakage patterns that we make use of in this work:

Let $F : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^{\log \nu}$ be a pseudo-random function, $\mathsf{rank} : \mathbb{R}^n \to \mathbb{R}^n$ be ranking function and $\lambda \in \mathbb{N}$ be a public parameter. Consider the transform PRT defined as follows:

```
– $\mathsf{PRT}(1^k, \lambda, \mathsf{MM})$:
    1. sample a key $K \xleftarrow{\$} \{0,1\}^k$;
    2. instantiate an empty multi-map $\mathsf{MM}'$;
    3. for all $\ell \in \mathbb{L}_{\mathsf{MM}}$,
        (a) let $\mathbf{r} := \mathsf{MM}[\ell]$ and $n_\ell = \#\mathbf{r}$;
        (b) compute $\mathbf{r}' := \mathsf{rank}(\mathbf{r})$;
        (c) let $n'_\ell = \lambda + F_K(\ell \| n_\ell)$;
        (d) if $n'_\ell > n_\ell$, set $\mathsf{MM}'[\ell] := \big(\mathbf{r}', \perp_1, \ldots, \perp_{n'_\ell - n_\ell}\big)$;
        (e) otherwise, set $\mathsf{MM}'[\ell] := \big(r'_1, \cdots, r'_{n'_\ell}\big)$;
    4. output $\mathsf{MM}'$.
– $\mathsf{Get}(\ell, \mathsf{MM})$: output $\mathsf{MM}[\ell]$.
```

Fig. 1. The pseudo-random transform.

- the *query equality pattern* is the function family $\mathsf{qeq} = \{\mathsf{qeq}_{k,t}\}_{k,t\in\mathbb{N}}$ with $\mathsf{qeq}_{k,t} : \mathbb{D}_k \times \mathbb{Q}_k^t \to \{0,1\}^{t\times t}$ such that $\mathsf{qeq}_{k,t}(\mathsf{DS}, q_1, \ldots, q_t) = M$, where M is a binary $t \times t$ matrix such that $M[i,j] = 1$ if $q_i = q_j$ and $M[i,j] = 0$ if $q_i \neq q_j$. The query equality pattern is referred to as the search pattern in the SSE literature;
- the *response identity pattern* is the function family $\mathsf{rid} = \{\mathsf{rid}_{k,t}\}_{k,t\in\mathbb{N}}$ with $\mathsf{rid}_{k,t} : \mathbb{D}_k \times \mathbb{Q}_k^t \to [2^{[n]}]^t$ such that $\mathsf{rid}_{k,t}\big(\mathsf{DS}, q_1, \ldots, q_t\big) = (\mathsf{DS}[q_1], \ldots, \mathsf{DS}[q_t])$. The response identity pattern is referred to as the access pattern in the SSE literature;
- the *response length pattern* is the function family $\mathsf{rlen} = \{\mathsf{rlen}_{k,t}\}_{k,t\in\mathbb{N}}$ with $\mathsf{rlen}_{k,t} : \mathbb{D}_k \times \mathbb{Q}_k^t \to \mathbb{N}^t$ such that $\mathsf{rlen}_{k,t}(\mathsf{DS}, q_1, \ldots, q_t) = \big(|\mathsf{DS}[q_1]|, \ldots, |\mathsf{DS}[q_t]|\big)$;
- the *domain size pattern* is the function family $\mathsf{dsize} = \{\mathsf{dsize}k, t\}_{k,t\in\mathbb{N}}$ with $\mathsf{dsize}k, t : \mathbb{D}_k \to \mathbb{N}$ such that $\mathsf{dsize}_{k,t}(\mathsf{DS}) = \#\mathbb{Q}$.
- the *total response length pattern* is the function family $\mathsf{trlen} = \{\mathsf{trlen}_k\}_{k\in\mathbb{N}}$ with $\mathsf{trlen}_k : \mathbb{D}_k \to \mathbb{N}$ such that $\mathsf{trlen}_k(\mathsf{DS}) = \sum_{q\in\mathbb{Q}_k} |\mathsf{DS}[q]|$;

4 The Pseudo-Random Transform

We describe the pseudo-random transform (PRT) in Figure 1 and provide a high level description below.

Overview. PRT is a data structure transformation that takes as input a multi-map MM, a security parameter k and a public parameter λ. It first generates a random key K and initializes an empty multi-map MM'. For each label ℓ in the multi-map, it ranks the tuple $\mathbf{r} := \mathsf{MM}[\ell]$;[4] resulting in a ranked tuple $\mathbf{r}'$. It then evaluates a PRF on the label ℓ concatenated to the length n_ℓ of $\mathbf{r}$. The

[4] The ranking function can be any ordering defined by the user; including standard ranking algorithms from information retrieval.

output of this PRF evaluation is then added to λ in order to compute a new length n'_ℓ. There are two possible cases that can occur at this point: (1) if n'_ℓ is larger than n_ℓ, the ranked response is padded with dummies and inserted in $\mathsf{MM}'[\ell]$; (2) if n'_ℓ is at most n_ℓ, the ranked response is truncated to its first n'_ℓ elements and inserted in $\mathsf{MM}'[\ell]$. Note that for ease of exposition and without loss of generality, we consider the case where $n'_\ell = n_\ell$ a padding. Finally, the transform outputs the multi-map MM'. The get algorithm simply outputs the tuple corresponding to the label ℓ.

A note on probabilistic analysis. Throughout this work, we model pseudo-random functions as random functions for the purposes of probabilistic analysis. It should be understood that all our bounds will have an additional negligible value in the security parameter.

4.1 Analyzing the Number of Truncations

For any label ℓ of the multi-map, the transform can pad or truncate its ranked response depending on the output of the PRF. In this Section, we will analyze the number of truncations induced by our transformation. The number of truncations is defined as

$$\#\{\ell \in \mathbb{L}_{\mathsf{MM}} : \#\mathsf{MM}'[\ell] < \#\mathsf{MM}[\ell]\}.$$

In the worst-case, the number of truncations can be $\#\mathbb{L}_{\mathsf{MM}}$ which occurs when every label in MM is truncated. We will show, however, that in practice this is very unlikely to occur. In particular, we will show that for real-world distributions of response lengths, the number of truncations is small with high probability. Note that if we set $\lambda \geq \max_{\ell\in\mathbb{L}} \#\mathsf{MM}[\ell]$, then truncations can never occur since $\#\mathsf{MM}'[\ell] \geq \max_{\ell\in\mathbb{L}} \#\mathsf{MM}[\ell] \geq \#\mathsf{MM}[\ell]$. We therefore only consider settings in which $\lambda < \max_{\ell\in\mathbb{L}} \#\mathsf{MM}[\ell]$.

Zipf-distributed multi-maps. To get a concrete bound on the number of truncations, we have to make an assumption on how the response lengths of the multi-map are distributed. Here, we will assume that they are distributed according to the Zipf distribution which is a standard assumption in information retrieval [14,43]. We note that our analysis can be extended to any power-law distribution. More precisely, we say that a multi-map MM is $\mathcal{Z}_{a,b}$-distributed if its r^{th} response has length

$$\frac{r^{-b}}{H_{a,b}} \cdot N$$

where $N \overset{def}{=} \sum_{\ell\in\mathbb{L}} \#\mathsf{MM}[\ell]$ is the volume of MM and $H_{a,b}$ is the harmonic number $\sum_{i=1}^{a} i^{-b}$. Throughout, we will consider multi-maps that are $\mathcal{Z}_{m,1}$-distributed where $m = \#\mathbb{L}_{\mathsf{MM}}$. From this assumption, it follows that the set of all response lengths is

$$L = (L_1, \ldots, L_m) = \left(\frac{N}{1 \cdot H_{m,1}}, \ldots, \frac{N}{m \cdot H_{m,1}}\right),$$

Note that we consider the case where $b = 1$ for ease of exposition but our analyses generalize to any b.

Theorem 1. *If* MM *is* $\mathcal{Z}_{m,1}$*-distributed, then with probability at least* $1-\varepsilon$ *the number of truncations is at most*

$$m \cdot \left(\frac{N}{\nu \cdot H_{m,1}^2} \cdot \left(H_{\rho,2} - \frac{\lambda \cdot H_{m,1}}{N} \cdot H_{\rho,1} \right) + \sqrt{\frac{\ln(1/\varepsilon)}{2m}} \right),$$

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

Due to space limitations, the proof of Theorem 1 is in the full version of this work. Note that the worst case information loss (i.e., the total number of pairs lost due to truncations) can be computed as $\sum_{i\in[\sigma]}(N/(i \cdot H_{m,1})) - \lambda)$, where σ is the number of truncations.

4.2 Analyzing the Storage Overhead

As detailed above, PRT can truncate or pad the responses in the multi-map. This has a direct impact on the storage overhead of the transformed multi-map since padding increases the storage overhead while truncations decrease it. In the following, we show that the size of the transformed multi-map MM' can be upper bounded with high probability without any assumptions on the distribution of response lengths.

Theorem 2. *With probability at least* $1-\varepsilon$*, the size of the transformed multi-map is at most*

$$m \cdot \left(\frac{\nu-1}{2} + (\nu-1) \cdot \sqrt{\frac{\ln(1/\varepsilon)}{2m}} + \lambda \right),$$

where $\lambda \geq 0$.

Due to space limitations, the proof of Theorem 2 is in the full version.

4.3 Concrete Parameters

In this Section, we will provide concrete parameters for PRT. Our goal is to find parameters that will provide a good balance between a small number of truncations and a small storage overhead. To study this, we first introduce two naïve transformations that achieve extreme tradeoffs between truncations and storage:

- the *naïve padding transform* is a transformation that pads the response of every label with dummies $\perp$ so that the length of the new responses are all set to the maximum response's length $\max_{\ell\in\mathbb{L}_{\mathsf{MM}}} \#\mathsf{MM}[\ell]$. Note that there are no truncations in this case and the size of the transformed multi-map is

$$S_{\mathsf{NV}} \overset{def}{=} m \cdot \max_{\ell\in\mathbb{L}_{\mathsf{MM}}} \#\mathsf{MM}[\ell].$$

- the *naïve truncating transform* truncates the responses of every label to the minimum response length $\min_{\ell\in\mathbb{L}} \#\mathsf{MM}[\ell]$. Note that the number of truncations in this case is $T_{\mathsf{NV}} \overset{def}{=} \#\mathbb{L}_{\mathsf{MM}} = m$ and the storage overhead is $m \cdot \min_{\ell\in\mathbb{L}_{\mathsf{MM}}} \#\mathsf{MM}[\ell]$, which is optimal.

Let $\mathsf{STE_{EMM}} = (\mathsf{Setup}, \mathsf{Get})$ be a static multi-map encryption scheme and PRT be the pseudo-random transform. Consider the scheme $\mathsf{VLH} = (\mathsf{Setup}, \mathsf{Get})$ defined as follows:

- $\mathsf{Setup}(1^k, \lambda, \mathsf{MM})$:
  1. generate a PRT-transform of MM by computing $\mathsf{MM}' = \mathsf{PRT}(1^k, \lambda, \mathsf{MM})$;
  2. encrypt the transform by computing
  $$(K, st, \mathsf{EMM}) \leftarrow \mathsf{STE_{EMM}}.\mathsf{Setup}\big(1^k, \mathsf{MM}'\big);$$
  3. output (K, st, EMM).
- $\mathsf{Get}_{\mathbf{C},\mathbf{S}}((K, st, \ell), \mathsf{EMM})$: $\mathbf{C}$ and $\mathbf{S}$ execute
  $$(r, \perp) \leftarrow \mathsf{STE_{EMM}}.\mathsf{Get}_{\mathbf{C},\mathbf{S}}\big((K, st, \ell), \mathsf{EMM}\big).$$

Fig. 2. VLH: A volume-hiding multi-map encryption scheme.

In the following Corollary, we set concrete values for λ and s so that we can achieve the best of both worlds. Specifically, we show that if the input multi-map is $\mathcal{Z}_{m,1}$-distributed, then by setting the output length of the PRF to $s = \log(L_1 + 1)$, where L_1 is the maximum response length, and setting $\lambda = O(\nu \cdot \alpha)$, where $1/2 < \alpha < 1$, then we can achieve storage overhead $\alpha \cdot S_{\mathsf{NV}}$ with $\beta \cdot T_{\mathsf{NV}}$ truncations with high probability, where β is a function of α and m.

Corollary 1. *Let $1/2 < \alpha < 1$. If MM is $\mathcal{Z}_{m,1}$-distributed and if*

$$\log \nu = \log\big(L_1 + 1\big) \quad \text{and} \quad \lambda = (\nu - 1) \cdot (2\alpha - 1)/4$$

then with probability at least:

- $1 - \exp\big(-m \cdot (2\alpha - 1)^2/8\big)$, *the total volume of the transformed multi-map is at most $\alpha \cdot \mathcal{S}_{\mathsf{NV}}$.*
- $1 - \exp\big(-2m/\log^2 m\big)$, *the number of truncations is at most*

$$\frac{1}{\log m} \cdot H_{\lfloor \frac{4}{2\alpha - 1} \rfloor, 2} \cdot T_{\mathsf{NV}}.$$

Due to space limitations, the proof of Corollary 1 is in the full version.

5 A Volume-Hiding Multi-Map Encryption Scheme

In this Section, we use the PRT to construct a volume-hiding multi-map encryption scheme. Our construction is described in detail in Figure 2 and works as follows.

Overview. The construction, $\mathsf{VLH} = (\mathsf{Setup}, \mathsf{Get})$, makes black-box use of an underlying multi-map encryption scheme $\mathsf{STE_{MM}} = (\mathsf{Setup}, \mathsf{Get})$. $\mathsf{VLH}.\mathsf{Setup}$ takes as input a security parameter k, a public parameter λ and a multi-map MM. It applies the PRT transform on MM which results in a new multi-map MM'. It then encrypts MM with $\mathsf{STE_{MM}}$, resulting in an encrypted multi-map

EMM, a state st and a key K which it returns as its own output. To execute a Get query ℓ on EMM, the client and the server execute $\mathsf{STE}_{\mathsf{MM}}.\mathsf{Get}$ on ℓ and EMM.

Efficiency. Assuming that $\mathsf{STE}_{\mathsf{MM}}$ is an optimal-time multi-map encryption scheme [15,10,7,1], the get complexity of VLH is $O(\lambda+n'_\ell)$, where $n'_\ell \in \{0, \cdots, \nu-1\}$. Therefore, the worst-case complexity is $O(\lambda+\nu)$ while the best-case complexity is $O(\lambda)$. The expected complexity is $O(\lambda+2^{s-1})$.

The storage overhead of VLH is

$$O(N) = O\bigg(\sum_{\ell\in\mathbb{L}_{\mathsf{MM}}} \#\mathsf{MM}'[\ell]\bigg) = O\bigg(\lambda\cdot m + \sum_{\ell\in\mathbb{L}_{\mathsf{MM}}} n'_\ell\bigg),$$

where, again, $n'_\ell \in \{0, \cdots, \nu-1\}$. So based on Corollary 1, when $\lambda = (\nu-1)\cdot(2\alpha-1)/4$ and $1/2 < \alpha < 1$, the storage overhead of VLH is

$$O\big(\alpha\cdot(\nu-1)\cdot m\big)$$

with high probability.

Correctness. The correctness of VLH is affected by the number of truncations induced by PRT. Based on Corollary 1, we can show that the number of truncations performed by VLH is at most $O(m/\log m)$ under the same assumptions stated in the corollary.

Security. We now describe the leakage profile of VLH assuming $\mathsf{STE}_{\mathsf{MM}}$ is instantiated with one of the standard optimal-time multi-map encryption schemes [15,10,7,1] all of which have leakage profile

$$\Lambda_{\mathsf{MM}} = (\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{Q}}) = \big(\mathsf{trlen}, (\mathsf{qeq}, \mathsf{rlen})\big).$$

Theorem 3. *If $\mathsf{STE}_{\mathsf{EMM}}$ is a $(\mathsf{trlen}, (\mathsf{qeq}, \mathsf{rlen}))$-secure multi-map encryption scheme and F in PRT is a pseudo-random function, then VLH is a $\big(\mathsf{dsize}, \mathsf{qeq}\big)$-secure multi-map encryption scheme.*

Due to space limitations, the proof of Theorem 3 is in the full version of this work. We observe that if we consider λ to be a public parameter, then dsize will leak an approximation of m rather than the exact value of m as stated in the theorem since $\lambda^{-1}\cdot\mathsf{trlen} = \lambda^{-1}\cdot\sum_{i\in[m]}\#\mathsf{MM}[\ell] = m + \lambda^{-1}\cdot\sum_{i\in[m]} r_i$, where r_i is generated uniformly at random. Note that this differs slightly from standard EMM schemes as their setup leakage is usually the sum of all pairs in the multi-map.

6 DST: The Densest Subgraph Transform

In this section, we introduce a new data structure transformation called the *dense subgraph transform* (DST). Unlike the PRT which achieves efficient storage by increasing truncations (and therefore losing information), this new transform improves on the storage complexity of PRT without losing any information.

The transformation is randomized and, surprisingly, we can show that, with high probability, it incurs no asymptotic storage overhead. Furthermore, we can also show that, for the case of Zipf-distributed multi-maps, it produces multi-maps that are asymptotically-smaller than the naïve padding and truncating transforms described in Section 4.3.

The DST takes a multi-map MM as input and creates a new multi-map MM' that is volume-hiding. This new structure results from re-arranging the data in the input multi-map according to a random bi-partite graph. To ensure this re-arrangement is still efficiently queryable, we represent it using a pair of standard data structures which include a multi-map MM_{G} and a dictionary DX. As we will show, the storage complexity of the final representation depends on certain properties of the bi-partite graph which are, in turn, inherited from the original multi-map.

Below, we provide a high-level overview of our transformation. A more detailed description is given in Section 6.1. The overview is divided in two parts: (1) a variant for general multi-maps; and (2) a variant for what we refer to as *concentrated* multi-maps. Note that the transformation handles both cases but achieves better results for the later. We then provide a more detailed description in Figure 6.

General multi-maps. Given a multi-map MM we begin creating a bi-partite graph with $\mathbb{L}_{\mathsf{MM}}$ as the top vertices and a set of n empty bins as the bottom vertices. For each label/vertex ℓ in $\mathrm{V}_{\mathsf{top}}$, we randomly select t bins and insert in each bin a single value of the tuple $\mathsf{MM}[\ell]$. Here, t is the maximum tuple size in the multi-map. If $\#\mathsf{MM}[\ell] < t$, then some of the selected bins won't receive a value. At the end of this process, we pad all bins so that they all have the same size. Note that this process creates a bi-partite graph where the edges incident to some top vertex/label ℓ correspond to the bins selected for that label/vertex. We now create two data structures to represent and efficiently process this bi-partite graph. The first is a dictionary that maps bin identifiers to the bin's contents. The second is a multi-map that maps a label to the identifiers of the bins associated to it. To retrieve the values associated to a given label ℓ, we query the multi-map on ℓ to retrieve its t bin identifiers and then query the dictionary on each of the t bin identifier to retrieve the contents of the bins.

It is already clear from this high-level description that all labels will have exactly the same response length: $t \cdot \alpha$, where α is the maximum size of a bin. It can be shown that with the right choice of parameters, this transformation results in a small amount of padding compared to the naïve approach.

Concentrated multi-maps. The storage overhead of our approach can be greatly improved when the multi-map satisfies a certain property we refer to as *concentration*. At a high level, a multi-map is concentrated if there exists a large number of values that appear in the tuples of a large number of labels. More formally, we define this property as follows.

Definition 3 (Concentrated multi-maps). *Let $\mu, \nu > 0$. We say that a multi-map MM is (μ, ν)-concentrated if there exists a set of μ labels $\ell_1, \cdots, \ell_\mu \in$*

$\mathbb{L}_{\mathsf{MM}}$ *such that,*

$$\# \bigcap_{i=1}^{\mu} \mathsf{MM}[\ell_i] = \nu.$$

We refer to this set of labels as MM*'s concentrated component and denote it* $\widehat{\mathbb{L}}_{\mathsf{MM}}$*. Throughout, we will assume the existence of an efficient algorithm* FindComp *that takes as input a multi-map and outputs the multi-map's concentrated component* $\widehat{\mathbb{L}}_{\mathsf{MM}}$*. If no such component exists, the algorithm outputs* $\perp$*.*

A (μ, ν)-concentrated multi-map has μ labels with an intersection of size ν which means that there is some redundancy in the structure. Unfortunately, the previous approach does not take advantage of this since it stores all the values in the multi-map independently. To exploit this redundancy, we proceed as follows. We dedicate a random subset of the bins to store the tuple values of the multi-map's concentrated component. Because the component's tuples have a large intersection, we will avoid storing the same values over and over again. At a high-level, we modify the process as follows. We first choose a random subset of ν bins and store, in each one, one value from the intersection $\cap_{\ell \in \widehat{\mathbb{L}}_{\mathsf{MM}}} \mathsf{MM}[\ell]$. We then add an edge between a random subset of size τ of these bins and the labels/vertices in the concentrated component. This results in μ labels/vertices sharing a large portion of the bins. In the special case of $\tau = \nu$, then this will result in μ labels/vertices that share the same bins. Notice the improved storage overhead as we don't store the values in the intersection in multiple bins. For the remaining labels, we follow a similar process to the one presented in the generic case. We sometimes refer to the value $(\mu - \frac{\nu}{\tau}) \cdot \tau$ as the multi-map's concentration, for $\tau > 0$.

Finding the concentration component. Our DST transform relies on an efficient algorithm FindComp to find the concentration component of a multi-map. We now describe such an algorithm. Informally, this algorithm will try different combinations of labels, compute the intersection of their tuples, and only retain the combination for which the intersection was the highest and that verify some specific conditions on its size. The algorithm first determines the set of labels $\widetilde{\mathbb{L}}_{\mathsf{MM}}$ with tuples of size $\Omega(n^{0.5+\delta})$, for some positive n. For $i \in [\lambda]$, it selects μ labels uniformly at random with replacement from $\widetilde{\mathbb{L}}_{\mathsf{MM}}$. We refer to this set as $\overline{\mathbb{L}}^{i}_{\mathsf{MM}}$. The algorithm then computes $\nu_i = \# \left(\bigcap_{\ell \in \overline{\mathbb{L}}^{i}_{\mathsf{MM}}} \mathsf{MM}[\ell] \right)$ where λ is the number of times the random selection is computed. The algorithm finally determines $\rho = \mathsf{argmax}_{i \in [\lambda]} \{\nu_i : \nu_i \in \Omega(n^{0.5+\delta})\}$ and outputs $\widehat{\mathbb{L}}_{\mathsf{MM}} = \overline{\mathbb{L}}^{\rho}_{\mathsf{MM}}$ if such ρ exists and $\widehat{\mathbb{L}}_{\mathsf{MM}} = \perp$ otherwise. Notice that the algorithm runs in $O(\lambda \cdot \mu)$ time. So it is sufficient to choose λ and μ to be polynomial in m. In our setting, we need to set the parameters to align with the the densest subgraph assumption (described in Definition 5) so we need $\mu = \Omega(m^{0.5+\delta})$ and for some positive δ. We note that this algorithm is only an example and we believe that more efficient algorithms can be designed.

A storage optimization. Notice that the auxiliary multi-map MM_{G} associates to every label a randomly selected set of t bins. In particular this means that the size of MM_{G} is $O(m \cdot t)$ which could be rather large. Fortunately, storing the identifiers of each bin is not necessary. Instead, we can choose the bins to assign to a label using a pseudo-random function and store the key in MM_{G}. This will reduce the size of MM_{G} to $O(m)$.

6.1 Detailed Description

We now provide a detailed description of the DST. The pseudo-code is in Figures 4 and 5. The transform makes black-box use of three pseudo-random functions F, H and G.

Setup. The Setup algorithm takes as input a security parameter 1^k, two integers n and τ and a multi-map MM. It instantiates a bi-partite graph $\mathrm{G} = (\mathrm{V}_{\mathsf{top}}, \mathrm{V}_{\mathsf{bot}}, \mathrm{E})$ where the top vertices $\mathrm{V}_{\mathsf{top}} = \mathbb{L}_{\mathsf{MM}}$ are the labels in MM, the bottom vertices $\mathrm{V}_{\mathsf{bot}}$ are n empty bins denoted $\mathbf{B} = \{B_1, \ldots, B_n\}$ and the set of edges E is empty.

The set of edges are generated as follows. Setup first computes the concentrated component of the multi-map $\widehat{\mathbb{L}}_{\mathsf{MM}} := \mathsf{FindComp}(\mathsf{MM})$. If no concentrated component exists, FindComp outputs $\perp$. If $\#\widehat{\mathbb{L}}_{\mathsf{MM}} \neq \perp$, it then pseudo-randomly chooses ν bins $\mathbf{B}' = \{B'_1, \ldots, B'_\nu\}$, where $\nu = \#\big(\bigcap_{\ell \in \widehat{\mathbb{L}}_{\mathsf{MM}}} \mathsf{MM}[\ell]\big)$. More precisely, it samples a k-bit value $\mathsf{rand}^\star$ uniformly at random and chooses the bins indexed by the set

$$\Big\{F_{K_1}(\mathsf{rand}^\star \| 1), \ldots, F_{K_1}(\mathsf{rand}^\star \| \nu)\Big\}.$$

Note that all these ν positions have to be *distinct*. If not, then it keeps resampling a new k-bit value $\mathsf{rand}^\star$ uniformly at random until no collisions are found. Note however that the probability p that no collision occurs, modeling F as a random function, is equal to

$$p = \prod_{i=0}^{\nu-1} \left(1 - \frac{i}{n}\right) \geq \left(1 - \frac{\nu}{n}\right)^{\nu} \approx e^{-\nu^2/n},$$

which tends to 1 when $\nu = o(n)$– which aligns with the concrete parameters that we will detail in Section 6.4.

For all $\ell \in \widehat{\mathbb{L}}_{\mathsf{MM}}$, it: (1) adds an edge between ℓ and $t - \tau$ bins outside of $\mathbf{B}'$; and (2) adds an edge between ℓ and τ bins in $\mathbf{B}'$. Note that this separation between the labels is necessarily for our reduction to the densest subgraph problem to hold.

To do the former, it indexes the bins in $\mathbf{B} \setminus \mathbf{B}'$ from 1 to $n - \tau$, samples a k-bit value $\mathsf{rand}_{\ell,1}$ uniformly at random, and chooses the bins indexed by the set

$$\Big\{H_{K_2}(\mathsf{rand}_{\ell,1} \| 1) + \mathsf{slide}_1, \ldots, H_{K_2}(\mathsf{rand}_{\ell,1} \| t - \tau) + \mathsf{slide}_{t-\tau}\Big\},$$

where slide_i, for $i \in \{1, \cdots, t - \tau\}$, is an integer used to deterministically map back the smaller output of H in $[n - \tau]$ to the corresponding bin identifier in $[n]$

and is computed as follows. First, it orders the set of bins in $\mathbf{B}'$ in a numerical order such that

$$\mathbf{B}' = \left(B_{\mathsf{pos}_1}, \cdots, B_{\mathsf{pos}_\nu}\right),$$

where $\mathsf{pos}_i < \mathsf{pos}_j$, for $i, j \in [\nu]$. Then it defines the following quantities based on which the slide value is determined– refer to Figure 3 for an illustration of the computation,

$$\mathsf{gap}_i = \begin{cases} [1, \mathsf{pos}_i - 1] & \text{if } i = 1 \\ ]\mathsf{pos}_{i-1} - (i-1), \mathsf{pos}_i - (i-1)[& \text{if } i \in \{2, \cdots, \nu\} \\ ]\mathsf{pos}_{i-1} - (i-1), n - \nu] & \text{if } i = \nu + 1 \end{cases}$$

Then, for $i \in \{1, \cdots, n-\nu\}$, identify $j \in \{1, \cdots, \nu+1\}$ such that $H_{K_2}(\mathsf{rand}_{\ell,1} \| i) \in \mathsf{gap}_j$, then set $\mathsf{slide}_i = j - 1$.

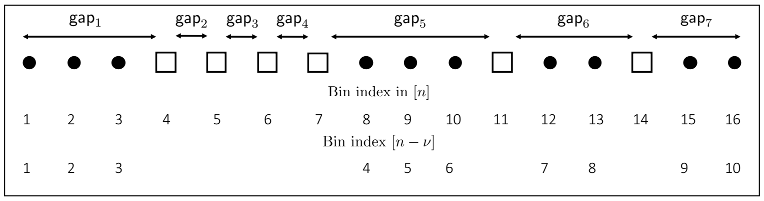

Fig. 3. Gaps computation for $n = 16$ and $\nu = 6$. $\square$ denotes bins being part of $\mathbf{B}' = \{B_4, B_5, B_6, B_7, B_{11}, B_{14}\}$ while $\bullet$ denotes bins in $\mathbf{B} \setminus \mathbf{B}'$.

Note that $\mathsf{rand}_{\ell,1}$ has also to be chosen in such a way that the selected $t - \tau$ positions are distinct. If not, similarly to above, it resamples a new k-bit value uniformly at random until no collision occurs. The probability that no collision occurs is approximately equal to $e^{-(t-\tau)^2/(n-\nu)}$ which tends to 1 when $t = o(n)$– which aligns with our concrete parameterization as we are going to detail in Section 6.4.

To do the latter, it samples $\mathsf{rand}_{\ell,2}$ uniformly at random and adds an edge between ℓ and all bins indexed by

$$\left\{ j_{G_{K_3}(\mathsf{rand}_{\ell,2}\|1)}, \cdots, j_{G_{K_3}(\mathsf{rand}_{\ell,2}\|\tau)} \right\},$$

where $j_i = F_{K_1}(\mathsf{rand}^\star \| i)$, for $i \in [\nu]$. If a collision is found, then it keeps resampling a new k-bit value uniformly at random until all τ positions are distinct. The probability that no collision occurs is approximately $e^{-\tau^2/\nu}$ which tends to 1 given our parametrization.

For each $\ell \notin \widehat{\mathbb{L}}_{\mathsf{MM}}$, it samples a k-bit value rand_ℓ uniformly at random and adds an edge between ℓ and all bins indexed by the set

$$\left\{ F_{K_1}(\mathsf{rand}_\ell \| 1), \ldots, F_{K_1}(\mathsf{rand}_\ell \| t) \right\}.$$

Again, if a collision is found, it keeps resampling a new k-bit value uniformly at random until all positions are distinct. The probability that no collision occurs is approximately equal to $e^{-t^2/n}$. Notice that at the end of this process, each vertex has degree exactly t.

Now Setup will use the graph to load the bins in $\mathrm{V}_{\mathsf{bot}}$ as follows. For each $\ell \notin \widehat{\mathbb{L}}_{\mathsf{MM}}$, it stores one value from the tuple $\mathsf{MM}[\ell]$ in one of the bins that are incident to ℓ. When inserting into a bin, the algorithm concatenates each value with ℓ (this will be helpful at query time). If $\#\mathsf{MM}[\ell] < t$, then some of the incident bins will not receive any value. For all $\ell \in \widehat{\mathbb{L}}_{\mathsf{MM}}$, it stores one element from $\mathsf{MM}[\ell] \backslash \mathbf{r}_\ell$ in the bins from $\mathbf{B} \backslash \mathbf{B}'$ that are incident to ℓ—again concatenating each value with ℓ, where

$$\mathbf{r}_\ell = (r_1, \cdots, r_\tau) \subseteq \bigcap_{\ell \in \widehat{\mathbb{L}}_{\mathsf{MM}}} \mathsf{MM}[\ell]$$

Also if $\#\mathsf{MM}[\ell] < t$, then some of the incident bins will not receive any value. Finally, it stores each value from the set

$$\mathbf{r}' := \bigcap_{\ell \in \widehat{\mathbb{L}}_{\mathsf{MM}}} \mathsf{MM}[\ell]$$

in a distinct bin in $\mathbf{B}'$ in such a way that every bin in $\mathbf{B}'$ will contain one value in $\mathbf{r}'$. Here, the algorithm concatenates the values with $\star$. The algorithm then pads all the bins to have the same size.

Finally, it creates a dictionary DX and a multi-map MM_{G}. The dictionary maps bin identifiers to bin contents. The multi-map MM_{G} maps labels $\ell \notin \widehat{\mathbb{L}}_{\mathsf{MM}}$ to rand_ℓ and labels $\ell \in \widehat{\mathbb{L}}_{\mathsf{MM}}$ to $(\mathsf{rand}_{\ell,1}, \mathsf{rand}_{\ell,2}, \mathsf{rand}^\star)$. It outputs $\mathsf{MM}' = (\mathsf{MM}_{\mathrm{G}}, \mathsf{DX})$.

The storage complexity of MM' is $O(m + n \cdot \lambda)$, where λ is the maximum load of a bin.

Get. Get operations on $\mathsf{MM}' = (\mathsf{MM}_{\mathrm{G}}, \mathsf{DX})$ work as follows. Given a label ℓ, we first query MM_{G} on ℓ. If $\ell \notin \widehat{\mathbb{L}}_{\mathsf{MM}}$, then MM_{G} returns rand_ℓ from which we compute the bin identifiers $\{F_{K_1}(\mathsf{rand}_\ell \| i)\}_{i=1}^{t}$. We can then query DX on the bin identifiers to recover the bins and output the elements concatenated with ℓ. If $\ell \in \widehat{\mathbb{L}}_{\mathsf{MM}}$, MM_{G} returns a triple $(\mathsf{rand}_{\ell,1}, \mathsf{rand}_{\ell,2}, \mathsf{rand}^\star)$ from which we can compute the sets

$$\left\{H_{K_2}(\mathsf{rand}_{\ell,1} \| i) + \mathsf{slide}_i\right\}_{i=1}^{t-\tau} \quad \text{and} \quad \left\{j_{G_{K_3}(\mathsf{rand}_{\ell_2} \| i)}\right\}_{i=1}^{\tau}$$

where $j_i = F_{K_1}(\mathsf{rand}^\star \| i)$, for $i \in [\nu]$, which we, in turn, use to query DX and recover the bins. From these bins the algorithm recovers the elements concatenated with ℓ and $\star$. The complexity of gets is $O(t \cdot \lambda)$ where, again, λ is the maximum load of a bin.

Let $n \in \mathbb{N}$ be a public parameter, $F : \{0,1\}^k \times \{0,1\}^* \to [n]$, $G : \{0,1\}^k \times \{0,1\}^* \to [n']$ and $H : \{0,1\}^k \times \{0,1\}^* \to [n'']$ be two pseudo-random functions with $n' < n'' < n$. Consider the transform DST defined as follows:

```
– $\mathsf{DST}(1^k, \mathsf{param}, \mathsf{MM})$:
  1. parse $\mathsf{param}$ as $(n, \tau)$, instantiate an empty dictionary $\mathsf{DX}$, an empty
     multi-map $\mathsf{MM}_{\mathrm{G}}$, and a bi-partite graph $\mathrm{G} = \big((\mathbb{L}_{\mathsf{MM}}, \mathbf{B}), \mathrm{E}\big)$ where $\mathbf{B} =$
     $(B_1, \cdots, B_n)$ and $\mathrm{E} = \emptyset$;
  2. compute $\widehat{\mathbb{L}}_{\mathsf{MM}} \leftarrow \mathsf{FindComp}(\mathsf{MM})$, set $\nu = \#\big(\bigcap_{\ell \in \widehat{\mathbb{L}}_{\mathsf{MM}}} \mathsf{MM}[\ell]\big)$ and
     $t := \max_{\ell \in \mathbb{L}_{\mathsf{MM}}} \mathsf{MM}[\ell]$;
  3. sample three keys $K_1 \xleftarrow{\$} \{0,1\}^k$, $K_2 \xleftarrow{\$} \{0,1\}^k$ and $K_3 \xleftarrow{\$} \{0,1\}^k$;
  4. for all $\ell \in \mathbb{L}_{\mathsf{MM}} \setminus \widehat{\mathbb{L}}_{\mathsf{MM}}$,
     (a) sample $\mathsf{rand}_\ell \xleftarrow{\$} \{0,1\}^k$ and output
             $(i_1, \cdots, i_t) := \Big\{F_{K_1}(\mathsf{rand}_\ell \| 1), \ldots, F_{K_1}(\mathsf{rand}_\ell \| t)\Big\},$
         if there exist distinct $i, j \in [t]$ for which $i_i = i_j$ redo the sampling.
         Add to E
             $\Big\{(\ell, i_j) : j \in [t]\Big\};$
     (b) parse $\mathsf{MM}[\ell]$ as $(r_1, \cdots, r_{n_\ell})$ and put $r_j \| \ell$ in $B_{i_j}$ for all $j \in [n_\ell]$;
  5. if $\widehat{\mathbb{L}}_{\mathsf{MM}} \neq \perp$, sample $\mathsf{rand}^\star \xleftarrow{\$} \{0,1\}^k$ and set $\mathbf{B}' = (B_{i_1}, \cdots, B_{i_\nu})$ where
             $(i_1, \cdots, i_\nu) := \Big\{F_{K_1}(\mathsf{rand}^\star \| 1), \ldots, F_{K_1}(\mathsf{rand}^\star \| \nu)\Big\},$
     if there exist distinct $i, j \in [\tau]$ for which $i_i = i_j$ redo the sampling.
     Otherwise set $\mathbf{B}' = \perp$;
  6. compute
             $\mathbf{r}' := \bigcap_{\ell \in \widehat{\mathbb{L}}_{\mathsf{MM}}} \mathsf{MM}[\ell] = (r'_1, \cdots, r'_\nu);$
  7. put $r'_j \| \star$ in $B_j$ for all $j \in [\nu]$ and $B_j \in \mathbf{B}'$;
  8. for all $\ell \in \widehat{\mathbb{L}}_{\mathsf{MM}}$,
     (a) sample $\mathsf{rand}_{\ell,1} \xleftarrow{\$} \{0,1\}^k$ and output
             $(i_1, \cdots, i_{t-\tau}) := \Big\{H_{K_2}(\mathsf{rand}_{\ell,1} \| 1), \ldots, H_{K_2}(\mathsf{rand}_{\ell,1} \; \| t - \tau)\Big\},$
         if there exist distinct $i, j \in [t-\tau]$ for which $i_i = i_j$, redo the sampling.
         Add to E
             $\Big\{(\ell, i_j + \mathsf{slide}_j) : j \in [t - \tau]\Big\};$
         where $\mathsf{slide}_j$ is computed as follows
           i. order $\mathbf{B}'$ in a numerical order such that $\mathbf{B}' := (B_{\mathsf{pos}_1}, \cdots, B_{\mathsf{pos}_\nu})$;
          ii. if $i_j \in [1, \mathsf{pos}_1]$, set $\mathsf{slide}_j = 0$;
         iii. if $i_j \in ]\mathsf{pos}_{i-1} - (i-1), \mathsf{pos}_i - (i-1)[$, set $\mathsf{slide}_j = i - 1$, for any
              $i \in \{2, \cdots, \nu\}$;
          iv. if $i_j \in ]\mathsf{pos}_\nu - \nu, n - \nu[$, set $\mathsf{slide}_j = \nu$;
```

Fig. 4. DST: The Dense Subgraph Transform (Part 1).

```
– $\mathsf{DST}(1^k, \mathsf{param}, \mathsf{MM})$:
    8. for all $\ell \in \widehat{\mathbb{L}}_{\mathsf{MM}}$,
        (b) sample $\mathsf{rand}_{\ell,2} \xleftarrow{\$} \{0,1\}^k$ and set $\mathbf{r}_\ell = (r_{i_1}, \cdots, r_{i_\tau}) \subseteq \mathbf{r}'$ where
                $(i_1, \cdots, i_\tau) := \Big\{ G_{K_3}(\mathsf{rand}_{\ell,2}|1), \ldots, G_{K_3}(\mathsf{rand}_{\ell,2}\|\tau) \Big\},$
            if there exist distinct $i, j \in [\tau]$ for which $i_i = i_j$, redo the sampling.
            Add
                $\Big\{ (\ell, F_{K_1}(\mathsf{rand}^\star\|i_j) : j \in [\tau] \Big\}$
            to E;
        (c) parse $\mathsf{MM}[\ell]$ as $(r_1, \cdots, r_{n_\ell})$;
        (d) for all $r_j \in \mathsf{MM}[\ell] \setminus \mathbf{r}_\ell$, then put $r_j\|\ell$ in $B_{i_j + \mathsf{slide}_j}$;
    9. set $\theta = \max_{i \in [n]} \#B_i$ and set for all $i \in [n]$
            $B_i = (B_i, \bot_1, \cdots, \bot_{\theta - \#B_i});$
    10. for all $i \in [n]$, set $\mathsf{DX}[i] = B_i$;
    11. for all $\ell \in \mathbb{L}_{\mathsf{MM}}$, if $\ell \in \widehat{\mathbb{L}}_{\mathsf{MM}}$, set $\mathsf{MM}[\ell] := (\mathsf{rand}_{\ell,1}, \mathsf{rand}_{\ell,2}, \mathsf{rand}^\star)$, other-
        wise set $\mathsf{MM}[\ell] := \mathsf{rand}_\ell$;
    12. output the key $K = (K_1, K_2, K_3)$ and $\mathsf{MM}' = (\mathsf{DX}, \mathsf{MM}_{\mathrm{G}})$.
– $\mathsf{Get}(K, \ell, \mathsf{MM})$:
    1. parse $K$ as $(K_1, K_2, K_3)$ and $\mathsf{MM}$ as $(\mathsf{DX}, \mathsf{MM}_G)$ and instantiate an
       empty set $\mathsf{Result}$;
    2. if $\mathsf{MM}_{\mathrm{G}}[\ell] = \mathsf{rand}$, then
        (a) add $\mathsf{DX}[\ell_i]$ to $\mathsf{Result}$, where for all $i \in [t]$,
                $\ell_i := F_{K_1}(\mathsf{rand}\|i);$
        (b) keep all values of the form $\cdot\|\ell$;
    3. if $\mathsf{MM}_{\mathrm{G}}[\ell] = (\mathsf{rand}_1, \mathsf{rand}_2, \mathsf{rand}^\star)$, then
        (a) add $\mathsf{DX}[\ell_i]$ to $\mathsf{Result}$, where for all $i \in [t - \tau]$,
                $\ell_i := H_{K_2}(\mathsf{rand}_1\|i) + \mathsf{slide}_i,$
            and for all $i \in [\tau]$,
                $\ell_i := j_{G_{K_3}(\mathsf{rand}_2\|i)}$
            where $(j_1, \cdots, j_\nu) = F_{K_1}(\mathsf{rand}^\star\|1), \ldots, F_{K_1}(\mathsf{rand}^\star\|\nu)$;
        (b) keep all values of the form $\cdot\|\ell$ or $\cdot\|\star$;
    4. output $\mathsf{Result}$.
```

Fig. 5. DST: The Dense Subgraph Transform (Part 2).

6.2 Analyzing the Load of a Bin

As seen in the previous Section, an important quantity to evaluate the query and storage efficiency of our transformation is the maximum load of a bin. In this Section, we will show that, with high probability, the maximum load can be upper bounded by $(N - N_{\mathsf{ds}})/n$ where N_{ds} is the size of the concentrated component and n is the number of bins. Before stating our result, we recall a generalization of Chernoff's inequality for the binomial distribution.

Lemma 1. *Let $X_1, \ldots, X_m$ be independent random variables over $\{0,1\}$ such that $\Pr[X_i = 1] = p_i$ and $\Pr[X_i = 0] = 1 - p_i$. If $X = X_1 + \cdots + X_m$, then*

$$\Pr[X \geq \mathrm{E}[X] + \theta] \leq \exp\left(- \frac{\theta^2}{2(\mathrm{E}[X] + \theta/3)} \right).$$

Theorem 4. *With probability at least $1 - \varepsilon$, the maximum load of a bin is at most*

$$\frac{N - N_{\mathsf{ds}}}{n} + \frac{\ln(1/\varepsilon)}{3}\left(1 + \sqrt{1 + \frac{18(N - N_{\mathsf{ds}})}{n \cdot \ln(1/\varepsilon)}}\right),$$

where $N_{\mathsf{ds}} = (\mu - \frac{\nu}{\tau}) \cdot \tau$, for $\tau > 0$.

Due to space limitations, the proof of Theorem 4 is in the full version.

6.3 Query and Storage Efficiency

We now give the storage and query efficiency of the DST transform.

Storage efficiency. The output of DST consists of a multi-map MM_G and a dictionary DX. The multi-map MM_G has tuples of size 1 or 3 depending on the label. That is, the size of the multi-map is upper bounded by $2m$. The dictionary DX stores the content of the padded bins. From Theorem 4 and the union bound, we have that the size of the dictionary is at most

$$N - N_{\mathsf{ds}} + \frac{n \cdot \ln(1/\varepsilon)}{3} \cdot \left(1 + \sqrt{1 + \frac{18(N - N_{\mathsf{ds}})}{n \cdot \ln(1/\varepsilon)}}\right)$$

with probability $1 - n \cdot \varepsilon$.

Get efficiency. The Get algorithm first retrieves either a random value or a pair of random values from MM_G. In the former case, t PRF evaluations are computed and t bins are retrieved. In the later case, $2t + \nu$ PRF evaluations are computed (using F, H and G) and t bins are retrieved. [5] Assuming that both MM_G and DX are data structures with optimal query complexity, the Get query complexity is at most

$$t \cdot \frac{N - N_{\mathsf{ds}}}{n} + \frac{t \cdot \ln(1/\varepsilon)}{3} \cdot \left(1 + \sqrt{1 + \frac{18(N - N_{\mathsf{ds}})}{n \cdot \ln(1/\varepsilon)}}\right)$$

with probability $1 - n \cdot \varepsilon$.

[5] Note that the computation of the slide_i's is $O(\nu)$. These evaluations can be performed once and stored at the client which reduces the total PRF evaluations at query time to $2t$.

6.4 Concrete Parameters

In this Section, we propose concrete parameters for the DST. In particular, we will be interested in parameters that guarantee better storage overhead than the naïve padding transform. Note that we do not compare to the naïve truncating approach since the DST does not lose any information.

General multi-maps. Recall that the naïve padding transform has a storage overhead

$$S_{\mathsf{NV}} \overset{def}{=} m \cdot \max_{\ell \in \mathbb{L}_{\mathsf{MM}}} \#\mathsf{MM}[\ell] = \Omega(N),$$

where $N \overset{def}{=} \sum_{\ell \in \mathbb{L}} \#\mathsf{MM}[\ell]$. From Theorem 4, we have the following corollary.

Corollary 2. *Let $n \geq 1$ and $m \geq 0$. If $N > n \log n$, then with probability at least $1 - 1/e^{N/5n}$, the size of the resulting multi-map is at most $O(N)$.*

Notice that if the original multi-map is $\mathcal{Z}_{m,1}$-distributed (but not necessarily concentrated), then $S_{\mathsf{NV}} = N \cdot m / H_{m,1}$ where $H_{m,1} = \Theta(\log m)$ is the harmonic number (please refer to Section 4). It follows that, in this case, $N = o(S_{\mathsf{NV}})$ so the storage overhead of DST is *small-o* of the overhead of the naïve padding transform.

Concentrated multi-maps. We now consider a multi-map MM with a concentrated component of size $(\mu - \frac{\nu}{\tau}) \cdot \tau$. We show below that in this case, the storage overhead induced by DST can be considerably smaller than the storage overhead of the naïve padding transform. The following Corollary is a consequence of Theorem 4.

Corollary 3. *Let $n \geq 1$ and $m \geq 0$. If $N > n \log n$,*

$$\mu = O\Big(m^{0.5+\delta} \cdot \mathrm{polylog}(m)\Big) \quad \textit{and} \quad \tau = O\Big(\mathrm{polylog}(m)\Big),$$

for some $\delta \geq 0$, then with probability at least $1 - 1/e^{N/5n}$, the size of the resulting multi-map is at most

$$O\Big(N - m^{0.5+\delta} \cdot \mathrm{polylog}(m)\Big).$$

As above, if the original multi-map MM is $\mathcal{Z}_{m,1}$-distributed, then the storage overhead of DST is *small-o* of the overhead of the naïve padding transform.

A remark on security. As we will see in Section 7, the parameters μ and τ have to satisfy certain constraints for our multi-map encryption scheme to be secure. In particular, the parameters have to be chosen in such a way that they verify the densest subgraph assumption which we detail in Definition 5. We note here that to satisfy both this assumption and the constraints of Corollary 3, it is sufficient that for some positive δ,

$$\mu = O\Big(m^{0.5+\delta} \cdot \mathrm{polylog}(m)\Big), \quad \tau = O(t) = O\Big(\mathrm{polylog}(m)\Big)$$

and,

$$\nu = O\left(m^{0.5+\delta} \cdot \text{polylog}(m)\right), \quad n = \Theta\left(m \cdot \text{polylog}(m)\right).$$

Note that this is only an example and is not the only choice of parameters that can be used. The intuition is that the larger the multi-map's concentration is, the better storage overhead DST will achieve. More precisely, multi-maps with larger values of μ and τ will achieve better storage gain as long as the DSP problem is hard.

7 AVLH: Advanced Volume Hiding Multi-Map Encryption Scheme

In this Section, we use the DST to construct a volume-hiding multi-map encryption scheme. Our construction is described in detail in Figure 6 and works as follows.

Overview. The construction, $\mathsf{AVLH} = (\mathsf{Setup}, \mathsf{Get})$, makes black-box use of an underlying response-hiding dictionary encryption scheme $\mathsf{STE}^{\mathsf{RH}}_{\mathsf{DX}} = (\mathsf{Setup}, \mathsf{Get})$. $\mathsf{AVLH.Setup}$ takes as input a security parameter k, a public parameter param, and a multi-map MM. It first applies the DST transform on MM which results in a key $K_1 = (K_{1,1}, K_{1,2}, K_{1,3})$ and two structures: a multi-map MM_{G} and a dictionary DX. It then encrypts the dictionary DX, resulting in an encrypted dictionary EDX, a state st_{DX} and a key K_2. It finally outputs a key $K = (K_1, K_2)$, a state $st = (\mathsf{MM}_{\mathrm{G}}, st_{\mathsf{DX}})$ and an encrypted multi-map $\mathsf{EMM} = \mathsf{EDX}$. To execute $\mathsf{AVLH.Get}$, the client differentiates two cases: if $\mathsf{MM}_{\mathrm{G}}[\ell]$ is a tuple composed of a single value rand, then the client and server execute the $\mathsf{STE}^{\mathsf{RH}}_{\mathsf{DX}}.\mathsf{Get}$ on ℓ_i where ℓ_i is a new label equal to $F_{K_{1,1}}(\mathsf{rand}\|i)$, for all $i \in [t]$, and $t = \max_{\ell \in \mathbb{L}_{\mathsf{MM}}} \#\mathsf{MM}[\ell]$. In this case the client $\mathbf{C}$ only outputs values of the form $\cdot\|\ell$. Otherwise, if $\mathsf{MM}_{\mathrm{G}}[\ell]$ is a tuple composed of a triple $(\mathsf{rand}_1, \mathsf{rand}_2, \mathsf{rand}^\star)$, then the client and server execute $\mathsf{STE}^{\mathsf{RH}}_{\mathsf{DX}}.\mathsf{Get}$ on ℓ_i where now ℓ_i is equal to $j_{G_{K_{1,3}}(\mathsf{rand}_2)\|i}$ where $j_l = F_{K_{1,1}}(\mathsf{rand}^\star\|l)$, for $l \in [\nu]$ and $i \in [\tau]$, and $H_{K_{1,2}}(\mathsf{rand}\|i) + \mathsf{slide}_i$ for all $i \in \{1, \cdots, t-\tau\}$. Note that slide_i, for which the computation was detailed in Section 6.1, is used to deterministically map the smaller output of H in $[n-\tau]$ into a value in $[n]$. In this case, the client $\mathbf{C}$ only outputs values of the form $\cdot\|\ell$ or $\cdot\|\star$.

Efficiency. Assuming that $\mathsf{STE}^{\mathsf{RH}}_{\mathsf{DX}}$ is an optimal-time dictionary encryption scheme [15,10,7,1], the get complexity of AVLH is $O(t{\cdot}\lambda)$ where $t = \max_{\ell \in \mathbb{L}_{\mathsf{MM}}} \#\mathsf{MM}[\ell]$ and λ is the load of a bin. Given the parameters detailed in the previous section and if $N > n \cdot \log n$ then the get complexity is

$$O\left(t \cdot \frac{N - m^{0.5+\delta} \cdot \text{polylog}(m)}{m \cdot \text{polylog}(m)}\right).$$

for some $\delta > 0$, when

$$\mu = O\left(m^{0.5+\delta} \cdot \text{polylog}(m)\right), \quad \tau = O(t) = O\left(\text{polylog}(m)\right)$$

and,

$$\nu = O\Big(m^{0.5+\delta} \cdot \mathrm{polylog}(m)\Big), \quad n = \Theta\Big(m \cdot \mathrm{polylog}(m)\Big).$$

The storage overhead of AVLH is, with high probability,

$$O\big(\sum_{i=1}^{n} \#\mathsf{DX}[\ell_i]\big) = O(n \cdot \lambda) = O(N - m^{0.5+\delta} \cdot \mathrm{polylog}(m)).$$

7.1 Security

We will now study the security of our construction. More precisely, we will show that it is volume-hiding in the sense that its query leakage does not include the response length. The proof relies on a computational assumption known as the densest subgraph assumption. We first recall this assumption and then proceed to stating our security theorem.

The densest subgraph problem. The hardness of the (decisional) *densest subgraph problem* problem was first used by Applebaum, Barak, and Wigderson in [2] to design public-key encryption schemes based on new assumptions. It was later used by Arora et al. [3] to study the hardness of financial products. Informally, the DSP asks whether it is possible to distinguish between a random regular bi-partite graph and a random regular bi-partite graph with a planted random subgraph.

Definition 4 (The (decision) densest subgraph problem.). *Let $m, n, t, \mu, \nu, \tau > 0$. The decisional unbalanced expansion problem is to distinguish between the two following distributions:*

- *$\mathcal{R}$ samples an (m, n, t)-bi-partite graph uniformly at random. In other words, for each vertex in $\mathrm{V}_{\mathsf{top}}$ it samples t neighbors from $\mathrm{V}_{\mathsf{bot}}$ uniformly at random.*
- *$\mathcal{P}$ is obtained as follows. First, two sets $T \subset \mathrm{V}_{\mathsf{top}}$ and $B \subset \mathrm{V}_{\mathsf{bot}}$, such that $\#T = \mu$ and $\#B = 2\nu$, are sampled uniformly at random. Then, for each vertex in T, we choose $t - \tau$ random neighbors in $\mathrm{V}_{\mathsf{bot}}$ and τ random neighbors in B. For each vertex in $\mathrm{V}_{\mathsf{top}} \setminus T$, we choose t random neighbors in $\mathrm{V}_{\mathsf{bot}}$.*

The following hardness assumption, used in [2,3], is based on state-of-the-art algorithms of Bhaskara, Charikar, Chlamtac, Feige, and Vijayaraghavan in [6].

Definition 5 (The DSP assumption). *There is no $\varepsilon > 0$ and* PPT *adversary $\mathcal{A}$ that can distinguish between $\mathcal{R}$ and $\mathcal{P}$ with advantage ε when*

$$n = o(m \cdot t), \quad \left(\frac{\mu \cdot \tau^2}{\nu}\right)^2 = o\left(\frac{m \cdot t^2}{n}\right), \quad \nu = \Omega(n^{0.5+\delta}),$$

$$\mu = \Omega(m^{0.5+\delta}) \quad \textit{and} \quad \tau = \widetilde{O}(\sqrt{t})$$

for some positive δ.

Leakage profile. We now describe the leakage profile of AVLH assuming $\mathsf{STE}_{\mathsf{DX}}^{\mathsf{RH}}$ is instantiated with one of the standard optimal-time dictionary encryption schemes [15,10,7,1] all of which have leakage profile

$$\Lambda_{\mathsf{DX}} = (\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{Q}}) = \big(\mathsf{trlen}, \mathsf{qeq}\big).$$

Theorem 5. *If* $\mathsf{STE}_{\mathsf{DX}}^{\mathsf{RH}}$ *is a* $(\mathsf{trlen}, \mathsf{qeq})$*-secure dictionary encryption scheme, F, G and H are pseudo-random functions, and the DSP assumptions holds, then* AVLH *is a* $\big((\mathsf{trlen}, \mathsf{conc}), \mathsf{qeq}\big)$*-secure multi-map encryption scheme; where* conc *is the leakage pattern that outputs a multi-map's concentration.*

Due to space limitations, the proof of Theorem 5 is in the full version of this work. The leakage pattern conc is due to the fact that we leak the size of the bins in trlen which is a function of the concentration.

Improving communication complexity. The communication (query) complexity of AVLH is equal to $O(t \cdot \lambda)$ where λ is the size of the bin and t the maximum response length. In the following we introduce a simple modification of AVLH such that the communication complexity becomes sub-linear in λ.

At a high level, the idea consists of replacing the retrieval of the entire bin's content by an oblivious retrieval that only fetches the value of interest (note that a bin will always contain at most one value associated to any label). Therefore this technique would reduce the overhead from λ to the overhead of a single oblivious access into an array of size λ. The (informal) modified AVLH works as follows. At setup time, we parse the content of each bin as an array (a RAM) and encrypt it using a computationally-secure state-of-the-art ORAM algorithm. Note that now, instead of using a response-hiding dictionary, we use a response-revealing one. The get algorithm works similarly to the one in AVLH except that the dictionary's get algorithm outputs an ORAM that we access separately. In terms of efficiency, the communication complexity becomes $O(t \cdot \sqrt{\lambda})$ assuming that we use square-root ORAM [22] as the underlying ORAM.[6] Note that we can achieve better communication complexity by leveraging techniques from [27]. The storage complexity however remains the same since square-root does not asymptotically increase the load of the bin.

8 Dynamic Volume Hiding Multi-Map Encryption Schemes

In this section, we show how to extend both VLH and AVLH to be dynamic. In particular, we will be interested in the following class of updates:

- *tuple addition*: this update operation adds a new tuple $(\ell, \mathbf{v})$ to the multi-map where ℓ is a label that was not part of the original label space $\mathbb{L}_{\mathsf{MM}}$.
- *tuple deletion*: this update operation removes an entire label/tuple pair $(\ell, \mathbf{v})$ from the multi-map.
- *editing*: this update operation modifies the content of a specific tuple $\mathbf{v}$ associated to ℓ by replacing an old value $v_{\mathsf{old}} \in \mathbf{v}$ by a new one v_{new}.

[6] Note that one cannot use tree-based ORAM schemes such as Path ORAM [41] as the security is function of the size of the RAM. In our case, under realistic parameters, the bin's load is very small to consider any of these schemes.

Let $\mathsf{STE}_{\mathsf{DX}}^{\mathsf{RH}} = (\mathsf{Setup}, \mathsf{Get})$ be a response-hiding dictionary encryption scheme and DST the densest subgraph transform. Consider the scheme $\mathsf{AVLH} = (\mathsf{Setup}, \mathsf{Get})$ defined as follows:

- $\mathsf{Setup}(1^k, \mathsf{param}, \mathsf{MM})$:
  1. generate a DST-transform of MM by computing
  $$(K_1, \mathsf{MM}_{\mathrm{G}}, \mathsf{DX}) \leftarrow \mathsf{DST}(1^k, \mathsf{param}, \mathsf{MM});$$
  2. encrypt DX by computing
  $$(K_2, st_{\mathsf{DX}}, \mathsf{EDX}) \leftarrow \mathsf{STE}_{\mathsf{DX}}^{\mathsf{RH}}.\mathsf{Setup}\big(1^k, \mathsf{DX}\big);$$
  3. set $K = (K_1, K_2)$, $st = (\mathsf{MM}_{\mathrm{G}}, st_{\mathsf{DX}})$, and $\mathsf{EMM} = \mathsf{EDX}$ and output (K, st, EMM).
- $\mathsf{Get}_{\mathbf{C},\mathbf{S}}((K, st, \ell), \mathsf{EMM})$:
  1. $\mathbf{C}$ parses K as $((K_{1,1}, K_{1,2}, K_{1,3}), K_2)$, st as $(\mathsf{MM}_{\mathrm{G}}, st_{\mathsf{DX}})$ and $\mathbf{S}$ parses EMM as EDX;
  2. if $\mathsf{MM}_{\mathrm{G}}[\ell] = \mathsf{rand}$, then
     (a) $\mathbf{C}$ and $\mathbf{S}$ execute $\mathsf{STE}_{\mathsf{DX}}^{\mathsf{RH}}.\mathsf{Get}_{\mathbf{C},\mathbf{S}}((K_2, st_{\mathsf{DX}}, \ell_i), \mathsf{EDX})$, for all $i \in [t]$, where
     $$\ell_i := F_{K_{1,1}}(\mathsf{rand}\|i);$$
     (b) $\mathbf{C}$ outputs values of the form $\cdot\|\ell$;
  3. if $\mathsf{MM}_{\mathrm{G}}[\ell] = (\mathsf{rand}_1, \mathsf{rand}_2, \mathsf{rand}^\star)$, then
     (a) $\mathbf{C}$ and $\mathbf{S}$ execute $\mathsf{STE}_{\mathsf{DX}}^{\mathsf{RH}}.\mathsf{Get}_{\mathbf{C},\mathbf{S}}((K_2, st_{\mathsf{DX}}, \ell_i), \mathsf{EDX})$, where for all $i \in [\tau]$,
     $$\ell_i := j_{G_{K_{1,3}}(\mathsf{rand}_2\|i)},$$
     and $(j_1, \cdots, j_\nu) = (F_{K_{1,1}}(\mathsf{rand}^\star\|1), \cdots, F_{K_{1,1}}(\mathsf{rand}^\star\|\nu))$ and for all $i \in \{1, \cdots, t-\tau\}$,
     $$\ell_i := H_{K_{1,2}}(\mathsf{rand}_1\|i) + \mathsf{slide}_i,$$
     where slide_j is computed as follows
        i. order
        $$\Big\{F_{K_{1,1}}(\mathsf{rand}^\star\|i)\Big\}_{i\in[\nu]}$$
        as $(\mathsf{pos}_1, \cdots, \mathsf{pos}_\nu)$;
        ii. if $H_{K_{1,2}}(\mathsf{rand}\|i) \in [1, \mathsf{pos}_1]$, set $\mathsf{slide}_i = 0$;
        iii. if $H_{K_{1,2}}(\mathsf{rand}\|i) \in]\mathsf{pos}_{j-1} - (j-1), \mathsf{pos}_j - (j-1)[$, set $\mathsf{slide}_i = j-1$, for any $j \in \{2, \cdots, \nu\}$;
        iv. if $H_{K_{1,2}}(\mathsf{rand}\|i) \in]\mathsf{pos}_\nu - \nu, n - \nu]$, set $\mathsf{slide}_i = \nu$;
     (b) $\mathbf{C}$ outputs all values of the form $\cdot\|\ell$ or $\cdot\|\star$.

Fig. 6. AVLH: An Advanced Volume Hiding Multi-Map Encryption Scheme.

In particular, we do not consider updates that add or remove a value to/from an existing tuple in the multi-map. In the following, we detail how to extend VLH to handle these three update operations and AVLH to handle the third update operation.

8.1 VLH^d: a dynamic variant of VLH.

The pseudo-code of VLH^d is in the full version and it works as follows.

Overview. $\mathsf{VLH}^\mathsf{d} = (\mathsf{Setup}, \mathsf{Get}, \mathsf{Put})$ makes black-box use of a dynamic response-hiding multi-map encryption scheme $\mathsf{STE}^{\mathsf{RH}}_{\mathsf{EMM}} = (\mathsf{Setup}, \mathsf{Get}, \mathsf{Put}, \mathsf{Remove})$ and of the volume-hiding multi-map encryption scheme $\mathsf{VLH} = (\mathsf{Setup}, \mathsf{Get})$.[7] Both the Setup algorithm and the Get protocol are exactly the same as of those of VLH. The Put algorithm takes as input an update u and processes it as follows. If $u = (\mathsf{add}, (\ell, \mathbf{v}))$, then the client first computes the PRT transform on a single-pair multi-map defined as $\{(\ell, \mathbf{v})\}$ and outputs a new single-pair multi-map $\{(\ell, \mathbf{v}')\}$. The client and server then execute $\mathsf{STE}^{\mathsf{RH}}_{\mathsf{EMM}}.\mathsf{Put}$ on the label/tuple pair $(\ell, \mathbf{v}')$. If $u = (\mathsf{rm}, \ell)$, then the client and server execute $\mathsf{STE}^{\mathsf{RH}}_{\mathsf{EMM}}.\mathsf{Remove}$ on the label ℓ. If $u = (\mathsf{edit}, (\ell, v_{\mathsf{old}}, v_{\mathsf{new}}))$, then the client and server first execute $\mathsf{VLH}.\mathsf{Get}$, the client receives the tuple $\mathbf{v}$ associated to the label ℓ. The client and server then execute $\mathsf{STE}^{\mathsf{RH}}_{\mathsf{EMM}}.\mathsf{Remove}$ on the label ℓ. The client locally replaces the value v_{old} by v_{new} in the tuple $\mathbf{v}$ and then executes $\mathsf{STE}^{\mathsf{RH}}_{\mathsf{EMM}}.\mathsf{Put}$ with the server on the modified label/tuple pair.

Efficiency analysis. In our analysis, assume $\mathsf{STE}^{\mathsf{RH}}_{\mathsf{EMM}}$ is an optimal-time dynamic multi-map encryption scheme [29,10,7]. It is clear that the get and storage complexity of VLH^d are exactly the same as VLH. The Put complexity varies depending on the type of the update operation. If u is a tuple addition or a tuple edit, then the Put complexity is $O(\lambda + n'_\ell)$ where $n'_\ell \in \{0, \cdots, 2^s - 1\}$. The worst-case is $O(\lambda + 2^s)$ while the best case is $O(\lambda)$. The expected complexity is $O(\lambda + 2^{s-1})$. If u is a *tuple deletion*, then the put complexity has constant time.

Security analysis. We now describe the leakage of VLH^d assuming that $\mathsf{STE}^{\mathsf{RH}}_{\mathsf{EMM}}$ is instantiated with one of the standard optimal-time forward-private multi-map encryption schemes [7,8,1] all of which have leakage profile

$$\Lambda_{\mathsf{MM}} = (\mathcal{L}_\mathsf{S}, \mathcal{L}_\mathsf{Q}, \mathcal{L}_\mathsf{U}) = (\mathsf{trlen}, (\mathsf{qeq}, \mathsf{rlen}), (\mathsf{op}, \mathsf{rlen}))$$

Theorem 6. *If* $\mathsf{STE}^{\mathsf{RH}}_{\mathsf{EMM}}$ *is a* $(\mathsf{trlen}, (\mathsf{qeq}, \mathsf{rlen}), (\mathsf{op}, \mathsf{rlen}))$*-secure multi-map encryption scheme,* F *in* PRT *is a pseudo-random function and* VLH *is a* $\big(m, \mathsf{qeq}\big)$*-secure multi-map encryption scheme, then* VLH^d *is a* $\big(m, \mathsf{qeq}, (\mathsf{op}, \mathsf{ueq})\big)$*-secure multi-map encryption scheme.*

The update equality pattern ueq leaks if and when a label edit has occurred. The proof of this theorem is similar to Theorem 3 and deferred to the full version of this work.

[7] Note that the same multi-map encryption scheme $\mathsf{STE}^{\mathsf{RH}}_{\mathsf{EMM}} = (\mathsf{Setup}, \mathsf{Get}, \mathsf{Put}, \mathsf{Remove})$ has to be used as the underlying multi-map encryption scheme for VLH.

8.2 $\mathsf{AVLH}^{\mathsf{d}}$: dynamic variant of AVLH.

The pseudo-code of $\mathsf{AVLH}^{\mathsf{d}}$ is in the full version and it works as follows.

Overview. The construction, $\mathsf{AVLH}^{\mathsf{d}}$, makes black box use of a dynamic response-hiding dictionary $\mathsf{STE}^{\mathsf{RH}}_{\mathsf{EDX}} = (\mathsf{Setup}, \mathsf{Get}, \mathsf{Put}, \mathsf{Remove})$ and of the volume hiding multi-map encryption scheme $\mathsf{AVLH} = (\mathsf{Setup}, \mathsf{Get})$.[8] The Setup algorithm and the Get protocol are exactly the same as of those of AVLH. The Put algorithm takes as input an update u and processes it as follows. Parse u as $(\mathsf{edit}, (\ell, \mathbf{v}))$, the client and server execute $(r, \bot) \leftarrow \mathsf{AVLH}.\mathsf{Get}_{\mathbf{C},\mathbf{S}}\big((K, st, \ell), \mathsf{EMM}\big)$ where $r = (B_{i_1}, \cdots, B_{i_t})$ and the client here does not dismiss any value from the retrieved bins. The client and server then execute $\mathsf{STE}^{\mathsf{RH}}_{\mathsf{EDX}}.\mathsf{Remove}$ on all retrieved bins. The client then identifies the bin that contains the value $\mathbf{v}_{\mathsf{old}} \| \ell$ (or $\mathbf{v}_{\mathsf{old}} \| \star$) that it replaces with $v_{\mathsf{new}} \| \ell$ (or by $\mathbf{v}_{\mathsf{new}} \| \star$ if concentrated). The client and server then execute $\mathsf{STE}_{\mathsf{EDX}}.\mathsf{Put}$ on the pairs (i_j, B_{i_j}), for all $j \in [t]$.

Efficiency analysis. We assume $\mathsf{STE}^{\mathsf{RH}}_{\mathsf{EDX}}$ is an optimal-time dynamic dictionary encryption scheme [29,10,7]. Clearly, the get and the storage complexity of $\mathsf{AVLH}^{\mathsf{d}}$ are exactly the same as AVLH. The Put complexity is equal to $O(t \cdot \lambda)$, where $t = \max_{\ell \in \mathbb{L}_{\mathsf{MM}}} \#\mathsf{MM}[\ell]$ is the maximum response length and λ is the size of the bin– which is the same as the get complexity. Refer to Section 7 for a more detailed and concrete analysis of the bin size λ.

Security analysis. We now describe the leakage of $\mathsf{AVLH}^{\mathsf{d}}$ assuming that $\mathsf{STE}^{\mathsf{RH}}_{\mathsf{EDX}}$ is instantiated with one of the standard optimal-time forward-private dictionary encryption scheme [7,8,1] all of which have a leakage profile at most

$$\Lambda_{\mathsf{DX}} = (\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{Q}}, \mathcal{L}_{\mathsf{U}}) = (\mathsf{trlen}, \mathsf{qeq}, \mathsf{op})$$

Theorem 7. *If* $\mathsf{STE}^{\mathsf{RH}}_{\mathsf{EDX}}$ *is a* $(\mathsf{trlen}, \mathsf{qeq}, \mathsf{op})$*-secure dictionary encryption scheme and* AVLH *is a* $\big(\mathsf{trlen}, \mathsf{qeq}\big)$*-secure multi-map encryption scheme, then* $\mathsf{VLH}^{\mathsf{d}}$ *is a* $\big(\mathsf{trlen}, \mathsf{qeq}, (\mathsf{op}, \mathsf{ueq})\big)$*-secure multi-map encryption scheme.*

The proof of this theorem is similar to Theorem 5 and deferred to the full version of this work.

References

1. G. Amjad, S. Kamara, and T. Moataz. Breach-resistant structured encryption. *IACR Cryptology ePrint Archive*, 2018:195, 2018.
2. B. Applebaum, B. Barak, and A. Wigderson. Public-key cryptography from different assumptions. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 171–180. ACM, 2010.
3. S. Arora, B. Barak, M. Brunnermeier, and R. Ge. Computational complexity and information asymmetry in financial products. *Communications of the ACM*, 54(5):101–107, 2011.

[8] Note that the same dictionary encryption scheme $\mathsf{STE}^{\mathsf{RH}}_{\mathsf{EDX}} = (\mathsf{Setup}, \mathsf{Get}, \mathsf{Put}, \mathsf{Remove})$ has to be used as the underlying dictionary encryption scheme for AVLH.

4. G. Asharov, M. Naor, G. Segev, and I. Shahaf. Searchable symmetric encryption: Optimal locality in linear space via two-dimensional balanced allocations. In *(STOC '16)*, pages 1101–1114, New York, NY, USA, 2016. ACM.
5. G. Asharov, G. Segev, and I. Shahaf. Tight tradeoffs in searchable symmetric encryption. In *Annual International Cryptology Conference*, pages 407–436. Springer, 2018.
6. A. Bhaskara, M. Charikar, E. Chlamtac, U. Feige, and A. Vijayaraghavan. Detecting high log-densities: an o (n 1/4) approximation for densest k-subgraph. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 201–210. ACM, 2010.
7. R. Bost. Sophos - forward secure searchable encryption. In *ACM (CCS '16)*, 20016.
8. R. Bost, B. Minaud, and O. 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.
9. D. Cash, P. Grubbs, J. Perry, and T. Ristenpart. Leakage-abuse attacks against searchable encryption. In *ACM (CCS '15)*, pages 668–679. ACM, 2015.
10. D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *(NDSS '14)*, 2014.
11. D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology - CRYPTO '13*. Springer, 2013.
12. M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *ASIACRYPT '10*, pages 577–594. Springer, 2010.
13. M. Chase and S. Kamara. Structured encryption and controlled disclosure. Technical Report 2011/010.pdf, IACR Cryptology ePrint Archive, 2010.
14. S. Chaudhuri, K. W. Church, A. C. König, and L. Sui. Heavy-tailed distributions and multi-keyword queries. In *ACM SIGIR 2007*.
15. R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In *(CCS '06)*, 2006.
16. I. Demertzis, D. Papadopoulos, and C. Papamanthou. Searchable encryption with optimal locality: Achieving sublogarithmic read efficiency. In *Annual International Cryptology Conference*, pages 371–406. Springer, 2018.
17. I. Demertzis, S. Papadopoulos, O. Papapetrou, A. Deligiannakis, and M. Garofalakis. Practical private range search revisited. In *Proceedings of the 2016 International Conference on Management of Data*, pages 185–198. ACM, 2016.
18. I. Demertzis and C. Papamanthou. Fast searchable encryption with tunable locality. In *SIGMOD'17*, 2017.
19. M. Etemad, A. Küpçü, C. Papamanthou, and D. Evans. Efficient dynamic searchable encryption with forward privacy. *Proceedings on Privacy Enhancing Technologies*, 2018(1):5–20, 2018.
20. S. Faber, S. Jarecki, H. Krawczyk, Q. Nguyen, M. Rosu, and M. Steiner. Rich queries on encrypted data: Beyond exact matches. In *European Symposium on Research in Computer Security (ESORICS '15). Lecture Notes in Computer Science*, volume 9327, pages 123–145, 2015.
21. B. A. Fisch, B. Vo, F. Krell, A. Kumarasubramanian, V. Kolesnikov, T. Malkin, and S. M. Bellovin. Malicious-client security in blind seer: a scalable private dbms. In *IEEE Symposium on Security and Privacy*, pages 395–410. IEEE, 2015.
22. O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM*, 43(3):431–473, 1996.

23. P. Grubbs, M. Lacharité, B. Minaud, and K. G. Paterson. Pump up the volume: Practical database reconstruction from volume leakage on range queries. In D. Lie, M. Mannan, M. Backes, and X. Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 315–331. ACM, 2018.
24. M. S. Islam, M. Kuzu, and M. Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *(NDSS '12)*, 2012.
25. S. Kamara and T. Moataz. Boolean searchable symmetric encryption with worst-case sub-linear complexity. In *Advances in Cryptology - EUROCRYPT '17*, 2017.
26. S. Kamara and T. Moataz. SQL on structurally-encrypted databases. In *ASIACRYPT*, 2018.
27. S. Kamara, T. Moataz, and O. Ohrimenko. Structured encryption and leakage suppression. In *Annual International Cryptology Conference*, pages 339–370. Springer, 2018.
28. S. Kamara and C. Papamanthou. Parallel and dynamic searchable symmetric encryption. In *Financial Cryptography and Data Security (FC '13)*, 2013.
29. S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *ACM (CCS '12)*, 2012.
30. G. Kellaris, G. Kollios, K. Nissim, and A. O. Neill. Generic attacks on secure outsourced databases. In *ACM Conference on Computer and Communications Security (CCS '16)*, 2016.
31. G. Kellaris, G. Kollios, K. Nissim, and A. O'Neill. Accessing data while preserving privacy. *CoRR*, abs/1706.01552, 2017.
32. M.-S. Lacharité, B. Minaud, and K. G. Paterson. Improved reconstruction attacks on encrypted data using range query leakage. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 297–314. IEEE, 2018.
33. X. Meng, S. Kamara, K. Nissim, and G. Kollios. Grecs: Graph encryption for approximate shortest distance queries. In *(CCS 15)*, 2015.
34. I. Miers and P. Mohassel. Io-dsse: Scaling dynamic searchable encryption to millions of indexes by improving locality. Cryptology ePrint Archive, Report 2016/830, 2016. http://eprint.iacr.org/2016/830.
35. M. Naveed, S. Kamara, and C. V. Wright. Inference attacks on property-preserving encrypted databases. In *ACM Conference on Computer and Communications Security (CCS)*, CCS '15, pages 644–655. ACM, 2015.
36. M. Naveed, M. Prabhakaran, and C. Gunter. Dynamic searchable encryption via blind storage. In *IEEE Symposium on Security and Privacy (S&P '14)*, 2014.
37. V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S.-G. Choi, W. George, A. Keromytis, and S. Bellovin. Blind seer: A scalable private dbms. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 359–374. IEEE, 2014.
38. R. Poddar, T. Boelter, and R. A. Popa. Arx: A strongly encrypted database system. Technical Report 2016/591.
39. D. Song, D. Wagner, and A. Perrig. Practical techniques for searching on encrypted data. In *IEEE S&P*, pages 44–55. IEEE Computer Society, 2000.
40. E. Stefanov, C. Papamanthou, and E. Shi. Practical dynamic searchable encryption with small leakage. In *(NDSS'14)*, 2014.
41. E. Stefanov, M. van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas. Path oram: An extremely simple oblivious ram protocol. In *(CCS)*, 2013.
42. Y. Zhang, A. O'Neill, M. Sherr, and W. Zhou. Privacy-preserving network provenance. *PVLDB*, 10(11):1550–1561, 2017.
43. G. K. Zipf. The psycho-biology of language. 1935.