Volume-Hiding Dynamic Searchable Symmetric Encryption with Forward and Backward Privacy

Yongjun Zhao[1], Huaxiong Wang[1,2], Kwok Yan Lam[1,3]
[1]Strategic Centre for Research in Privacy-Preserving Technologies & Systems
[2]School of Physical and Mathematical Sciences
[3]School of Computer Science and Engineering
{yongjun.zhao,hxwang,kwokyan.lam}@ntu.edu.sg
Nanyang Technological University

June 10, 2021

Abstract

Volumetric leakage in encrypted databases had been overlooked by the community for a long time until Kellaris *et al.* (CCS '16) proposed the first database reconstruction attack leveraging communication volume. Their attack was soon improved and several query recovery attacks were discovered recently. In response to the advancements of volumetric leakage attacks, volume-hiding searchable symmetric encryption (SSE) schemes have been proposed (Kamara and Moataz, Eurocrypt '19 & Patel *et al.*, CCS '19). In these schemes, the database is padded in a clever way so that the volume (*i.e.*, the number of responses) for any search query is the same or computationally indistinguishable while keeping the storage complexity and search complexity as small as possible.

Unfortunately, existing volume-hiding SSE schemes do not support atomic updates (*i.e.*, addition/deletion of an arbitrary keyword-document pair), which is the most common update operation considered in the SSE literature. Meanwhile, recent volumetric attacks (Wang *et al.*, EuroS&P '20 & Blackstone *et al.*, NDSS '20) indeed target dynamic databases.

We initiate a formal study of volume-hiding *dynamic* SSE. We extend the existing definition of volume-hiding leakage function into the dynamic setting and present efficient constructions $\mathsf{VH\text{-}DSSE}$ and $\mathsf{VH\text{-}DSSE}^k$. $\mathsf{VH\text{-}DSSE}$ suffers from non-negligible correctness error. To remedy the disadvantage of $\mathsf{VH\text{-}DSSE}$, we propose a multi-copy construction $\mathsf{VH\text{-}DSSE}^k$ that amplifies correctness by parallel repetition. As a side contribution, both $\mathsf{VH\text{-}DSSE}$ and $\mathsf{VH\text{-}DSSE}^k$ satisfy the strongest notions of backward-privacy, which is the first one in the literature, to the best of our knowledge.

1 Introduction

A searchable symmetric encryption (SSE) scheme [16] encrypts a private database into an encrypted index. An untrusted cloud server holding this encrypted index can answer keyword search queries issued by the database owner efficiently. The main design goal of SSE in terms of efficiency is to avoid a linear scan over the large database during search, which is unavoidable if one uses generic solutions such as multi-party computation (MPC) and fully homomorphic encryption (FHE). State-of-the-art SSE constructions [3, 10, 16, 18] for keyword search achieve optimal search complexity, *i.e.* linear in the number of matching documents.

The efficiency of SSE comes at the cost of revealing some information, called *leakage*, to the server. Informally, an SSE scheme is secure, *by definition*, if the untrusted server does not gain extra information about the private database and the search queries beyond a pre-defined set of leakages. Common leakages in the SSE literature include search patterns, response lengths, database size, *etc.* There is a long-standing and still active line of research [5, 9, 24, 28, 29, 31, 34, 39, 44–47, 52, 64] devoted to studying the security implications of these leakages.

Volumetric Leakage. For SSE schemes that achieve optimal-time search, the number of matching documents is revealed to the server unavoidably. Such information is often referred to as *response length pattern* or *volume pattern.*[1] While this leakage seems very innocent at the first glance, somewhat surprisingly, researchers have demonstrated that it actually imposes severe threats to the actual security of the encrypted database: an attacker can reconstruct the database content [28,31,39], or recover the database owner's search queries [5, 28, 64] using volumetric leakage.

In response to these attacks, Kamara and Moataz [37] initiated the study of volume-hiding SSE. Their idea is to pad the database carefully so that the volumes of search queries are computationally indistinguishable while keeping the size of the (padded) encrypted index and the search complexity small. Patel *et al.* [55] proposed a more efficient scheme dprfMM that achieves asymptotically optimal storage complexity (linear in the size of the original database) and search complexity (linear in the maximum volume).

Unfortunately, the dprfMM scheme proposed by Patel *et al.* [55] does not support updates on the encrypted database. Meanwhile, the dynamic variants proposed by Kamara and Moataz [37] only support a limited class of updates. Let DB be the database and $DB(w)$ be the subset of entries in DB that contain the keyword w. Three types of update operations were considered in [37]: 1. Tuple addition: adding $DB(w^*)$ to DB where w^* is a *brand new* keyword; 2. Tuple deletion: removing *all entries* in $DB(w^*)$ from DB for an existing keyword w^*; 3. Tuple edition: delete all entries in $DB(w^*)$ first and then add back updated $DB(w^*)$. Their VLH construction supports all three types while their aVLH construction only supports the last type.

So far, existing schemes do not support *atomic update* that adds/removes a single keyword-document pair (w, ID) if w already exists in the database. Atomic update is the most flexible and versatile type of update considered in the SSE literature, as it can efficiently emulate both file and tuple updates. Meanwhile, recent volumetric attacks [5, 28, 64] assume that the attacker is able to inject malicious content into the database and observing changes in volumetric leakage in subsequent search queries. Both Decode and Binary attack[2] [5] and replay attack [64] recover search queries with probability 1 by inserting a few dozen malicious files. Given the gaps between the volumetric attacks and defenses in the literature, a formal study of volume-hiding *dynamic* SSE schemes (DSSE) is not just a purely theoretical interest, but also a pressing need in practice.

Forward and Backward Privacy. Putting volumetric leakage aside, restricting other leakages in DSSE schemes is already a challenging research problem. Two security notions that aim to restrict the update leakages in DSSE have been considered in the literature: *forward* and *backward* privacy. Informally, a DSSE scheme is *forward-private* [6,8,21,42] if it is impossible to connect a new update to previous operations, *e.g.*, it is impossible to tell whether the addition is for a previously searched keyword or not. Forward-privacy mitigates certain query recovery attacks [66] on encrypted databases.

[1]We use the terms (*i.e.*, volume, response length, number of matching documents) interchangeably.
[2]We remark that they work against ORAM-based solutions as well.

Backward-privacy [8] is a similar notion that requires search queries should not leak entries after they have been deleted. Several notions of backward-privacy have been proposed in the literature. Among them, BP-I [8] and BP-I* [67] are the two strongest (and incomparable) notions. To the best of our knowledge, there is no DSSE scheme in the literature that satisfies these two notions simultaneously.[3]

Is there a baseline solution? If one does not care about storage complexity, there is a simple baseline solution for volume-hiding *static* SSE: pad the database with dummy entries so that the volumes of all keywords are the same (*e.g.*, equal the maximum volume). However, complications arise immediately when we apply the same design in the *dynamic* setting. First, it is not clear how to replace an existing dummy entry with a new keyword-document pair (w, ID) efficiently and securely, especially when the keyword w may have been searched before. Second, updates may change the maximum volume of the existing database, which means paddings on all other keywords need to be changed accordingly. This will not be an efficient operation. Perhaps more importantly, the server can differentiate an update on a keyword with maximum volume from an update on other keywords, which is a natural violation of privacy during updates. Note that this issue is inherent as long as the maximum response length is included in the leakage profile, even if heavyweight tools like ORAM are used. In short, there does not seem to be any straightforward baseline solution, even if we afford to sacrifice storage complexity. We are thus intrigued to ask the following question:

Can we have a dynamic SSE scheme with the following properties?

1. *volume-hiding;*
2. *supporting atomic update operation;*
3. *asymptotically optimal storage complexity and search complexity;*
4. *with forward and backward privacy.*

1.1 Our Contributions

In this work, we initiate a formal study of volume-hiding DSSE and answer the above question in the affirmative.

- We give the first formal security definition for volume-hiding DSSE, which generalizes the existing notion for static SSE [37, 55] in several aspects. Notably, our volume-hiding notion prevents the leakage profile from containing the maximum response length, which is partly motivated by the issue of update privacy discussed above.[4]

- We describe the first volume-hiding DSSE construction VH-DSSE that supports the addition/deletion of any keyword-document pairs. Our starting point is the state-of-the-art volume-hiding static SSE construction dprfMM in [55]. To upgrade dprfMM into a dynamic SSE efficiently and make it satisfy our strong volume-hiding notion, we modify its inner workings and also borrow techniques from data structure design and the ORAM literature.

[3] Unlike forward-privacy, we are not aware of what concrete attacks backward privacy (regardless of BP-I, BP-I* or weaker variants) can thwart to the best of our knowledge. That said, leaking less information provides a stronger security guarantee in general. It is possible that in the future, new attacks targeting DSSE schemes without backward privacy will be identified.

[4] See Section 4.3 for more discussions on why maximum response length should be hidden.

Table 1: Comparison with recent SSE Schemes. $|\mathcal{W}|$: size of keyword space; $|\mathcal{ID}|$: size of file-identifier space; $\ell_{\max} = \max_{w \in \mathcal{W}} |DB(w)|$: maximum volume in DB; $d_{\max}$: maximum number of deletion supported; N: the (current) size of the (possibly dynamic) database. For a dynamic database using lazy deletion strategy, $N = N_0 + k$ where N_0 is the size of the initial database and k is the total number of update operations so far. For keyword w: $a_w = \#$ updates, $i_w = \#$ insertions, $\ell_w = |DB(w)|$ the volume of keyword w in DB. **BP** stands for backward privacy type. **VH** stands for volume-hiding. *am.* stands for amortized efficiency. The correctness guarantee of VLH assumes DB follows Zipf-distribution. Also see [37, Corollary 4.3] for concrete parameters. We do not consider the dynamic variants in [37] as they do not support atomic updates considered in this work.

schemes		Storage Complexity		Computation Complexity		Correctness	**BP**	**VH**
		Server	Client	Search	Update			
[37]	VLH	$\Theta(\|\mathcal{W}\|\ell_{\max})$	$\Theta(1)$	$\Theta(\ell_{\max})$	N.A.	$(1 - \mathsf{negl}(\lambda), 1 - o(1))$	N.A.	Yes
	aVLH	$\Theta(N)$	$\Theta(1)$	$\Theta(\ell_{\max} \cdot \log N)$	N.A.	perfect	N.A.	Yes
[55]	dprfMM	$\Theta(N)$	$\Theta(1)$	$\Theta(\ell_{\max})$	N.A.	perfect	N.A.	Yes
[17]	SD$_a$	$\Theta(N)$	$\Theta(1)$	$\Theta(a_w + \log N)$	$\Theta(\log N)$ (*am.*)	perfect	BP-II	No
	SD$_d$	$\Theta(N)$	$\Theta(1)$	$\Theta(a_w + \log N)$	$\Theta(\log^3 N)$	perfect	BP-II	No
	QOS	$\Theta(N)$	$\Theta(1)$	$\Theta(\ell_w \log i_w + \log^2 \|\mathcal{W}\|)$	$\Theta(\log^3 N)$	perfect	BP-III	No
[67]	FB-DSSE	$\Theta(\|\mathcal{W}\|\|\mathcal{ID}\|)$	$\Theta(\|\mathcal{W}\|)$	$\Theta(a_w\|\mathcal{ID}\|)$	$\Theta(\|\mathcal{ID}\|)$	perfect	BP-I*	No
[62]	Aura	$\Theta(N)$	$\Theta(\|\mathcal{W}\|d_{\max})$	$\Theta(\ell_w)$	$\Theta(1)$	$(1, 1 - o(1))$	BP-II	No
VH-DSSE		$\Theta(N)$	$\Theta(1)$	$\Theta(\ell_{\max} + \log N)$	$\Theta(\log N)$ (*am.*)	$(1, \beta = \ell_{\max}/N)$ $(1 - \mathsf{negl}(\lambda), 1 - o(1))$	BP-I & I*	Yes
VH-DSSE^k		$\omega(\log \lambda)\Theta(N)$	$\Theta(1)$	$\omega(\log \lambda)\Theta(\ell_{\max} + \log N)$	$\omega(\log \lambda)\Theta(\log N)$ (*am.*)	$(1 - \mathsf{negl}(\lambda), 1 - \mathsf{negl}(\lambda))$	BP-I & I*	Yes

- As the maximum response length is hidden from the untrusted server, VH-DSSE becomes lossy, *i.e.*, some matching documents may not be returned by the search protocol. However, we prove that the loss is bounded under some assumptions on the database distribution with high probability. Our detailed probability analysis turned out to be more difficult than expected when computing the lower bound of the summation of *dependent* random variables following hypergeometric distribution. See Section 6.2.3 for more detail. Our probability analysis may be of independent interest. VH-DSSE achieves asymptotically optimal storage complexity. For search and update complexity, VH-DSSE is comparable with recent forward and backward private SSE schemes [17]. Our final construction VH-DSSE^k achieves negligible loss by amplifying the correctness of VH-DSSE using parallel repetition.

- As a side product, we show that VH-DSSE and VH-DSSE^k satisfy the two strongest notions of backward-privacy (BP-I and BP-I*). To the best of our knowledge, VH-DSSE and VH-DSSE^k are the first ones in the literature.

A detailed comparison with prior arts in terms of various efficiency metrics and security guarantees is given in Table 1. We can see that the asymptotic efficiency of our schemes is comparable with recent works, such as the SD$_a$ construction in [17]. The most recent forward&backward private SSE scheme Aura [62] suffers from asymptotically large client-side storage and it only achieves a weaker notion of backward privacy (BP-II).

1.2 Related Work

The most common data structure used by SSE constructions is multi-map. The design of SSE boils down to encrypting the multi-map in an efficiently queryable manner. The notion of SSE has been extended to structured encryption [14] that encrypts more general data structures like graphs [14, 48, 50] while maintaining the ability to privately query the data structure. Search queries beyond (single-)keyword search are also considered in the literature, including conjunctive/disjunctive/Boolean query [11, 35, 56], range query [20, 22, 40, 58], skyline query [49, 63, 65], SQL query [36], *etc.*

Towards the goal of better understanding the concrete security of SSE schemes, many attacks leveraging different leakage profiles have been proposed. This line of research was initiated by Islam *et al.* [34], who use access pattern leakage to recover search queries. Subsequent attacks reconstruct the plaintext values [29, 39, 45, 47] from an encrypted database supporting expressive queries, *e.g.*, range query/k-NN. While most of these works leverage access pattern and search pattern leakage only, volumetric leakage attacks [5,28,31,39,64] are receiving increasing attention recently, possibly because they are much harder to avoid and their consequences are much less understood.

The idea of forward and backward privacy, first appeared in [13] and [61] respectively, is to restrict the information leakage during updates. Subsequent works [8, 12, 21, 42, 48, 62, 67] formalized the notions and also made significant improvements in terms of efficiency. Kamara, Moataz, and Ohrimenko [38] proposed a general transform that suppresses search pattern leakage. Kamara and Moataz [37] and Patel *et al.* [55] study the problem of hiding volumetric leakage.

Two recent works [19, 32] also consider reducing leakages in searchable symmetric encryption. Demertzis *et al.* [19] reduce the search/access/volumetric patterns in *static* SSE via adjustable Oblivious RAM and adjustable padding. Their *static* SSE scheme, SEAL(α, x), is parameterized by α (which controls the amount of search leakage) and x (which controls the amount of volumetric leakage). They left the problem of handling dynamic databases as an open question explicitly and discussed difficulties in making their scheme dynamic. Gui *et al.* [32] introduce bucketization to hide volumetric leakage and delayed, pseudorandom write-backs to hide access patterns. The latter technique also makes search and update operations indistinguishable, which leads to a strong notion of forward and backward security. Like [37], their dynamic SSE scheme only supports document-level addition/deletion. Atomic update operations, which is the focus of our work, can be simulated (inefficiently) by a deletion on an old document followed by an insertion on the modified document.

2 Preliminary

We use $\lambda \in \mathbb{N}$ to denote the security parameter, which is an implicit input to all algorithms/protocols. PPT stands for probabilistic polynomial-time. $\mathsf{negl}(\lambda)$ denotes the set of functions that grow slower than any inverse polynomial in λ. For a finite set or a vector S, $|S|$ denotes its size and $s \leftarrow S$ denotes picking an element uniformly at random from S. We denote $[i, j] = \{i, i+1, \ldots, j\}$. We consider a collection of D documents with identifiers $\mathsf{ID}_1, \ldots, \mathsf{ID}_D$, each of which contains keywords from a given alphabet Λ. We follow the canonical database representation in the SSE literature: let DB consists of pairs of keyword-identifiers, such that $(w, \mathsf{ID}) \in DB$ if and only if the file ID contains the keyword w. For a dynamic database that adopts lazy deletion strategy, the database entries (w, ID, op) will have an extra "operation" field $op \in \{add, del\}$. For each w, let $DB(w)$ denote the set of database entries (w, ID) (or (w, ID, op)) associated with the keyword w. $|DB|$ denotes the size of the database, namely $|DB| = \sum_{w \in \mathcal{W}} |DB(w)|$. $DB[i, j]$ denotes the subset of DB consisting of the i-th entry to the j-th entry (both inclusive) in DB, where $1 \leq i \leq j \leq |DB|$. Let $\mathcal{Q}$ denote the query space. We focus on single-keyword search queries, so $\mathcal{W} = \mathcal{Q}$. We denote by $(\mathsf{out}_A, \mathsf{out}_B) \leftarrow \Pi_{A,B}(X; Y)$ the execution of a two-party protocol Π between parties A and B, where X and Y are the inputs provided by A and B, respectively; and out_A and out_B are the outputs to A and B, respectively. Our protocols are executed between two parties, a client and a server. We omit the subscripts when the identities of the participants are clear in the context.

2.1 (Dynamic) Symmetric Searchable Encryption

A *symmetric searchable encryption scheme* (SSE) $\Sigma = (\mathsf{Setup}, \mathsf{Search})$ consists of a setup algorithm Setup and a protocol Search that is executed between a client and a server:

- $\mathsf{Setup}(1^\lambda, DB)$ outputs (K, τ, EDB) where K is a secret key for the client, τ is the client's local state, and EDB is an (initially empty) encrypted database (if DB is empty) that is sent to the server. The algorithm may additionally take a parameter N (*i.e.*, $\mathsf{Setup}(1^\lambda, DB, N)$) denoting the maximum supported number of entries.
- $\mathsf{Search}(K, q, \tau; EDB)$ is a protocol for searching the encrypted database. We consider single-keyword search query, *i.e.*, $q = w \in \Lambda$. The client's output is $DB(w)$. The protocol may also modify K, τ and EDB.

If the SSE is *dynamic* (DSSE), it has an additional Update protocol:

- $\mathsf{Update}(K, op, (w, \mathsf{ID}), \tau; EDB)$ inserts an entry to or removes an entry from DB. Input consists of $op = add/del$, file identifier ID, and keyword w. The protocol may modify K, τ and EDB.

Usually, SSE schemes are lossless, *i.e.*, all the matching document identifiers are returned by the search protocol. Recently, many lossy SSE schemes [15, 15, 37, 59, 62] have been proposed for better security or efficiency. For the convenience of analysis, we need to introduce the notion of *correctness* for SSE schemes.

Definition 2.1 (Correctness of SSE)**.** An SSE scheme Σ is called (p, ϵ)-correct for a keyword w if, with probability at least p, it returns at least ϵ fraction of $DB(w)$.[5] We say that Σ is perfectly correct if $\epsilon = p = 100\%$ for all keywords.

Looking forward, our keyword-level correctness (Definition 2.1) captures the correctness guarantee of our constructions in a fine-grained manner accurately (Theorem 6.1, 6.2 and 7.2). It is easy to define a macro-level correctness definition on top of Definition 2.1. We leave the flexibility to readers.

The security of Σ is parametrized by a (stateful) leakage function $\mathcal{L} = (\mathcal{L}^{Stp}, \mathcal{L}^{Srch}, \mathcal{L}^{Updt})$ describing the information revealed to the server in the protocol execution. $\mathcal{L}^{Stp}$ refers to leakage during setup, $\mathcal{L}^{Srch}$ during a search operation, and $\mathcal{L}^{Updt}$ during updates. Informally, a secure DSSE scheme with leakage $\mathcal{L}$ should reveal nothing about the database DB beyond this leakage. This is formally captured by a standard real/ideal experiment with two games $\mathsf{Real}_{\mathcal{A}}^{DSSE}$, $\mathsf{Ideal}_{\mathsf{Sim},\mathcal{L}}^{DSSE}$. We present these two games (adapted from [17]) in Figure 1.

Definition 2.2. A DSSE scheme Σ is adaptively-secure with respect to leakage function $\mathcal{L}$, iff for any PPT-adversary $\mathcal{A}$, there exists a stateful PPT simulator $\mathsf{Sim} = (\mathsf{SimSetup}, \mathsf{SimSearch}, \mathsf{SimUpdate})$ such that $|\Pr[\mathsf{Real}_{\mathcal{A}}^{DSSE}(1^\lambda) = 1] - \Pr[\mathsf{Ideal}_{\mathsf{Sim},\mathcal{L}}^{DSSE}(1^\lambda)]| \leq \mathsf{negl}(\lambda)$.

2.1.1 Typical Leakage Functions.

We describe typical leakage functions that have been considered in the literature. We follow existing notions [37, 38, 55].

[5] Note that false negatives are inherent by definition in all lossy (D)SSE schemes. If a lossy DSSE scheme uses lazy deletion, tuples for marking deletion may not be returned, leading to false positives too.

```
$\mathsf{Real}^{DSSE}_{\mathcal{A}}(1^\lambda)$
1: $DB_0 \leftarrow \mathcal{A}(1^\lambda), (K, \tau_0, EDB_0) \leftarrow \mathsf{Setup}(1^\lambda, DB_0)$
     // Let $t_0$ be an empty string
2: for $k = 1$ to $\mathsf{poly}(\lambda)$ do
3:   $(type_k, \mathsf{ID}_k, w_k, \ell_k) \leftarrow \mathcal{A}(EDB_k, t_0, \ldots, t_{k-1})$
4:   if $type_k == \mathsf{Search}$ then
5:     $(\tau_k, DB(w_k); EDB_k) \leftarrow \mathsf{Search}(K, w_k, \tau_{k-1}, \ell_k; EDB_{k-1}, \ell_k)$
6:   else if $type_k == \mathsf{Update}$ then
7:     $(K, \tau_k; EDB_k) \leftarrow \mathsf{Update}(K, add/del, (\mathsf{ID}_k, w_k), \tau_{k-1}; EDB_{k-1})$
8:   Let $t_k$ be the message from client to server in Search/Update above
9: return $\mathcal{A}(EDB_0, t_0, t_1, \ldots, t_{\mathsf{poly}(\lambda)})$
$\mathsf{Ideal}^{DSSE}_{\mathsf{Sim},\mathcal{L}}(1^\lambda)$
1: $DB_0 \leftarrow \mathcal{A}(1^\lambda), (st_{\mathcal{S}}, EDB_0) \leftarrow \mathsf{SimSetup}(1^\lambda, \mathcal{L}^{Stp}(DB_0))$
     // Let $t_0$ be an empty string
2: for $k = 1$ to $\mathsf{poly}(\lambda)$ do
3:   $(type_k, \mathsf{ID}_k, w_k, \ell_k) \leftarrow \mathcal{A}(EDB_0, t_0, \ldots, t_{k-1})$
4:   if $type_k == \mathsf{Search}$ then
5:     $(st_{\mathcal{S}}; t_k, EDB_k) \leftarrow \mathsf{SimSearch}(st_{\mathcal{S}}, \mathcal{L}^{Srch}(EDB_{k-1}, w_k), \ell_k)$
6:   else if $type_k == \mathsf{Update}$ then
7:     $(st_{\mathcal{S}}; t_k, EDB_k) \leftarrow \mathsf{SimUpdate}(st_{\mathcal{S}}, \mathcal{L}^{Updt}(EDB_{k-1}, w_k))$
8: return $\mathcal{A}(EDB_0, t_0, t_1, \ldots, t_{\mathsf{poly}(\lambda)})$
```

Figure 1: Real and ideal experiments for the DSSE scheme. The highlighted part is tailored for our modified dprfMM scheme only.

- **Search Pattern**: $\mathcal{L}^{Eq}$ reports whether two search queries are to the same keyword or not. Formally, for a sequence of search queries: $\vec{q} = (q_1, \ldots, q_t)$, $\mathcal{L}^{Eq}(\vec{q}) = M$ consists of a $t \times t$ binary matrix such that $M[i][j] = 1$ iff $q_i = q_j$

- **Response Length**: $\mathcal{L}^{RL}$ reports the volume associated with the queries. Formally, for a sequence of search queries $q_1, , \ldots, q_t$ and a database DB, that is $\mathcal{L}^{RL}(DB, q_1, \ldots, q_t) = (\ell_{q_1}, \ldots, \ell_{q_t}) = (|DB(q_1)|, \ldots, |DB(q_t)|)$.

- **Maximum Response Length**: $\mathcal{L}^{MRL}$ reports the maximum number of matching documents associated with any keyword in the database. Formally, for any database DB, $\mathcal{L}^{MRL}(DB) = \ell_{\max} = \max_{q \in \mathcal{W}} |DB(q)|$.

- **Database Size**: $\mathcal{L}^{DSize}$ reports the total number of key-value pairs in the database, namely $\mathcal{L}^{DSize}(DB) = |DB| = \sum_{w \in \mathcal{W}} |DB(w)|$.

2.2 Forward and Backward Privacy

Definition 2.3 (Forward Privacy [6, 8]). An $\mathcal{L}$-adaptively-secure DSSE scheme that supports single-keyword addition is forward private iff the update leakage function $\mathcal{L}^{Updt}$ can be written as $\mathcal{L}^{Updt}(op, w, \mathsf{ID}) = \mathcal{L}'(op, \mathsf{ID})$ where $\mathcal{L}'$ is a stateless function, $op \in \{add, del\}$, and ID is a file identifier., In the literature, four types of backward privacy with different leakage patterns have been considered. BP-I, BP-II, and BP-III were first formalized in [8], with BP-I leaking the least information and BP-III leaking the most. Recently, Zuo *et al.* [67] proposed BP-I*, a variant of BP-I backward privacy that is strictly stronger than BP-II but it is *incomparable* to BP-I. In the following, we present the necessary notation for BP-I and BP-I*. Readers may refer to [8] for the remaining two weaker notions. Let Q be a list with one entry for each operation. For searches, the entry is (u, w) where u is the timestamp and w is the searched keyword. For updates, it is $(u, op, (w, \mathsf{ID}))$ where $op = add/del$ and ID is the index of the modified file. $\mathbf{TimeDB}(w) = \{(u, \mathsf{ID}) | (u, add, (w, \mathsf{ID})) \in Q \wedge \forall u', (u', del, (w, \mathsf{ID})) \notin Q\}$ is the function that returns all timestamp file-identifier pairs of keyword w that have been added to DB and have not been deleted. $\mathbf{Updates}(w) = \{u | (u, add, (w, \mathsf{ID})) \in Q \vee (u, del, (w, \mathsf{ID})) \in Q\}$ is the function that returns the timestamp of each insertion/deletion operation for w. Using the notions above, backward privacy is defined as follows.

Definition 2.4 (Backward Privacy [8,67])**.** An $\mathcal{L}$-adaptively-secure SSE scheme is backward private:

- BP-I: iff $\mathcal{L}^{Updt}(op, w, \mathsf{ID}) = \mathcal{L}'(op)$ and $\mathcal{L}^{Srch}(w) = \mathcal{L}''(\mathbf{TimeDB}(w), a_w)$,
- BP-I*: iff $\mathcal{L}^{Updt}(op, w, \mathsf{ID}) = \mathcal{L}'(op)$ and $\mathcal{L}^{Srch}(w) = \mathcal{L}''(\mathbf{Updates}(w))$,

where $\mathcal{L}'$ and $\mathcal{L}''$ are stateless functions, a_w is the number of updates for the keyword w.

If a DSSE scheme satisfies both BP-I and BP-I*, then its update leakage is at most the *intersection* of the two leakage profiles defined in BP-I and BP-I*. Such schemes provide stronger security guarantee than those that only satisfy only one backward privacy notion.

3 Our Definitions

3.1 Security Definition for Volume-Hiding DSSE

We define security notions for volume-hiding dynamic symmetric searchable encryption below. Patel *et al.* [55] defined volume-hiding leakage profiles for *static* SSE schemes. Our definition naturally extends theirs. Intuitively, a *volume-hiding* leakage function should ensure that the number of values associated with any keyword, namely the *volume* of the keyword, is not revealed. We formalize this intuition as follows.

We start with the description of volume-hiding game $\mathsf{VolH}_{\mathcal{A},\mathcal{L}}(1^\lambda)$ parameterized by a (stateful) leakage function $\mathcal{L} = (\mathcal{L}^{Stp}, \mathcal{L}^{Srch}, \mathcal{L}^{Updt})$ and an adversary $\mathcal{A}$. We then explain why we have formulated the game in this way and compare it with the existing one for static databases in [55]. The $\mathsf{VolH}_{\mathcal{A},\mathcal{L}}(1^\lambda)$ game proceeds in three phases:

- Setup phase: The adversary $\mathcal{A}$ chooses two databases DB_0 and DB_1, subject to the following constraints:
  - $\sum_{w \in \mathcal{W}} |DB_0(w)| = \sum_{w \in \mathcal{W}} |DB_1(w)|$;
  - $\max_{w \in \mathcal{W}} |DB_0(w)| = \max_{w \in \mathcal{W}} |DB_1(w)|$.

  The challenger C flips a coin $b \leftarrow \{0, 1\}$ internally and returns $\mathcal{L}^{Stp}(DB)$ to the adversary $\mathcal{A}$.
- Query phase: The challenger C and the adversary $\mathcal{A}$ repeat the following steps for $\mathsf{poly}(\lambda)$ times:

1. $\mathcal{A}$ adaptively chooses a query q and sends it to the challenger C;
2. If q is a Search query, C parses $q = w$ and returns $\mathcal{L}^{Srch}(DB, w)$ to $\mathcal{A}$;
3. If q is an Update query, then C parses $q = (q_0, q_1)$, where $q_b = (op_b, (w_b, \mathsf{ID}_b))$ for $b \in \{0, 1\}$. C updates DB with q_b and returns $\mathcal{L}^{Updt}(DB, q_b)$ to $\mathcal{A}$.

- Output phase: $\mathcal{A}$ outputs a bit b'.

We say that $\mathcal{A}$ wins the $\mathsf{VolH}_{\mathcal{A},\mathcal{L}}(1^\lambda)$ game if $b' = b$. The advantage of $\mathcal{A}$ in the above game is defined as $\mathsf{Adv}^{\mathsf{VolH}}_{\mathcal{A},\mathcal{L}} = |2\Pr[b' = b] - 1|$.

Definition 3.1 (Volume-Hiding Leakage Functions)**.** The leakage function $\mathcal{L} = (\mathcal{L}^{Stp}, \mathcal{L}^{Srch}, \mathcal{L}^{Updt})$ is volume-hiding if $\mathsf{Adv}^{\mathsf{VolH}}_{\mathcal{A},\mathcal{L}}$ is negligible for all (possibly computationally unbounded) adversaries.

With the definition of volume-hiding leakage function, we introduce the security definition for dynamic volume-hiding SSE schemes as follows.

Definition 3.2 (Volume-Hiding DSSE)**.** A DSSE scheme $\Sigma = (\mathsf{Setup}, \mathsf{Search}, \mathsf{Update})$ is *volume-hiding* if there exists a leakage profile $\mathcal{L} = (\mathcal{L}^{Stp}, \mathcal{L}^{Srch}, \mathcal{L}^{Updt})$ such that

1. Σ is adaptively-secure with respect to leakage $\mathcal{L}$ according to Definition 2.2;

2. $\mathcal{L}$ is a volume-hiding leakage profile according to Definition 3.1.

3.2 Intuition and Explanation of Our Definitions

In the $\mathsf{VolH}_{\mathcal{A},\mathcal{L}}$ game, the adversary $\mathcal{A}$ (implicitly) defines two dynamically evolving databases DB_0 and DB_1 (hence the number of matching documents, *i.e.*, the response length/volumes of all keywords) using the Update queries. Definition 3.2 requires that $\mathcal{A}$ cannot tell whether C has chosen DB_0 or DB_1. Therefore, it guarantees that the response lengths do not give $\mathcal{A}$ any advantage in distinguishing two databases.

Observe that Definition 3.1 prevents $\mathcal{L}^{Srch}$ and $\mathcal{L}^{Updt}$ from containing the maximum response length $\mathcal{L}^{MRL}$. Otherwise, there is a trivial attack: the adversary updates the two databases DB_0, DB_1 so that they have different maximum response lengths and uses $\mathcal{L}^{MRL}$ to win. This attack is inherent, as long as the adversary is allowed to choose one pair of update queries (q_0, q_1) to be of different types. Recent attacks on DSSE schemes using volumetric leakage [5, 64] indeed assume the adversary has the ability to inject files containing maliciously chosen keywords. Our formulation thus faithfully captures the adversary's ability in these attacks. Moreover, if a DSSE scheme is forward-private (*cf.*, Definition 2.3), then its update leakage $\mathcal{L}^{Updt}$ cannot contain $\mathcal{L}^{MRL}$. Otherwise, the adversary can differentiate an update on the keyword with maximum response length from an update on other keywords. This violates the requirement that the update leakage being independent of w (*i.e.*, $\mathcal{L}^{Updt}(op, w, \mathsf{ID}) = \mathcal{L}'(op, \mathsf{ID})$). Our definition reflects the natural security requirements of volume-hiding in the dynamic setting. Nevertheless, it is easy to obtain weaker notions by adding restrictions (say, requiring $\mathcal{L}^{MRL}(DB_0) = \mathcal{L}^{MRL}(DB_1)$ throughout the game). But we believe such definitions will be too weak. Looking ahead, removing $\mathcal{L}^{MRL}$ will be the main design principle of our volume-hiding DSSE schemes.

3.3 Comparisons with Previous Definitions.

3.3.1 Compare with KM19 [37].

Kamara and Moataz did not provide any formal definition for volume-hiding. In their security analysis, they only informally argued that "their (static) construction is volume-hiding" by observing that "its query leakage does not include the response length."[6]

3.3.2 Compare with PPYY19 [55].

Patel *et al.* [55] gave the first formal definition of volume-hiding leakage function for *static* databases, partially closing the gap left by Kamara and Moataz. Their volume-hiding game, which we renamed as sVolH in the rest of the paper for easier differentiation, begins by having *the challenger* generate two databases according to the volumes chosen (under certain constraints) by the adversary. Then the challenger randomly chooses one of the databases and sends the setup and search leakages of the chosen database to the adversary. Informally, a leakage profile is said to be volume-hiding if no adversary can tell apart which database the challenger has chosen. For completeness, we recall their formal definitions in Appendix D.

Our Definitions 3.1 and 3.2 extend the ones given in [55] in several aspects. First, we define volume-hiding leakage in the context of DSSE instead of static SSE. The sVolH game in [55] imposes two restrictions on DB_0 and DB_1: (1) the total number of keyword-document pairs are the same (*i.e.*, $\mathcal{L}^{DSize}(DB_0) = \mathcal{L}^{DSize}(DB_1)$); (2) the maximum response lengths are the same (*i.e.*, $\mathcal{L}^{MRL}(DB_0) = \mathcal{L}^{MRL}(DB_1)$). In VolH, these two restrictions are only imposed in the *setup* phase. The only other restriction in VolH is that the number of *updates* is the same throughout the whole game, *e.g.*, in the VolH game, it is possible that q_0 is an *add* query while q_1 is a *del* query. Second, in our VolH game, it is the adversary who fully decides the two challenge database, further giving the adversary extra distinguishing power. Lastly, in Definition 3.1, we allow the adversary to have a small-but-negligible advantage in winning the VolH game, while in [55], the advantage must be 0, which seems to be unnecessarily restrictive. For these reasons, we believe our definitions are more general.

Although the last difference between our definitions and those in [55] mentioned above prevents us from asserting that our definitions are (strictly) stronger, it is not hard to see that if there exists an adversary $\mathcal{A}$ who has an advantage p in the sVolH game, there must exist an adversary $\mathcal{B}$ who can win our VolH game with advantage at least p. In Appendix D, we formally prove this statement to quantitatively compare our definitions with those in [55].

4 Technical Roadmap

Our starting point is the *static* volume-hiding SSE scheme dprfMM proposed by Patel *et al.* [55], which is the only one achieving asymptotically optimal storage complexity. For ease of exposition, we first review dprfMM and discuss the difficulties in making it dynamic. We describe the high-level ideas of our approach and the challenges that we encountered and how we circumvent them before we finally introduce our construction.

4.1 Review of dprfMM@CCS'19 [55]

At a high-level, dprfMM still follows the common design footprint of many inverted-index based (non-volume-hiding) SSE schemes in the literature [10, 16]. To avoid leaking ℓ_w (*i.e.*, the volume of the search query $q = w$), one can pad DB with dummy entries so that all keywords have the same maximum volume $\ell_{\max}$, or all keywords have (pseudo)randomly distributed volumes [37]. However, these two padding methods sig-

[6]They did not argue similarly for their dynamic variant.

nificantly increase the server-side storage overhead. Ideally, we want to keep the size of the encrypted database to be linear in the plaintext one, *i.e.*, $|EDB| \in O(|DB|)$.[7]

The main idea of Patel *et al.* [55] is to use Cuckoo hash table [53] instead of general dictionaries/hash tables to store the encrypted inverted-index entries $(\mathsf{PRF}(K_w, i), \mathsf{Enc}(K, (w, \mathsf{ID}_{w,i})))$, where $i \in [1, \ell_w]$. In detail, to store n entries, a Cuckoo hash table consists of two arrays T_0, T_1 of size $O(n)$, an $\omega(1)$-size stash, and two independent hash functions H_0, H_1. An entry x will be stored in either $T_0[H_0(x)]$ or $T_1[H_1(x)]$ or in the stash.[8] In dprfMM, the hash functions H_0, H_1 are instantiated by a (delegatable) PRF in the following way: the entry $\mathsf{Enc}(K, (w, \mathsf{ID}_{w,i}))$ will be stored in either $T_0[\mathsf{PRF}(K_w, (i||0))]$ or $T_1[\mathsf{PRF}(K_w, (i||1))]$ or in the local stash. Empty slots in T_0, T_1 are padded with dummy entries. In the search protocol, the server obliviously returns entries in $T_0[\mathsf{PRF}(K_w, (i||0))]$ and $T_1[\mathsf{PRF}(K_w, (i||1))]$ for all $i \in [1, \ell_{\max}]$. Note that only the first ℓ_w entries may be correct results. Some correct results may reside in the stash. The rest could be encryptions of either incorrect key-value pairs $(w', \mathsf{ID}_{w'})$ or dummy entries. The server cannot differentiate them from correct results but the client can locally filter them out. In this way, the server only knows the maximum volume $\ell_{\max}$ but not keyword-specific volume ℓ_w.

4.2 Difficulties in Making dprfMM@CCS'19 Dynamic

Making dprfMM dynamic is not straightforward. Let's consider the basic update operation: adding a new (w, ID^*) tuple to an inverted-index. Suppose originally the keyword w is contained in $\ell_w < \ell_{\max}$ documents. Following the insertion algorithm of *plaintext* Cuckoo hash tables [1], naturally, the new key-value pair should be stored in exactly one of the three possible locations: $T_0[\mathsf{PRF}(K_w, (\ell_w + 1||0))]$, $T_1[\mathsf{PRF}(K_w, (\ell_w + 1||1))]$, or the stash, in order to maintain the invariance in dprfMM. However, the first two cases violate forward-privacy (Definition 2.3): If the client has searched the keyword w before, then the server must have accessed both $T_0[\mathsf{PRF}(K_w, (i||0))]$, $T_1[\mathsf{PRF}(K_w, (i||1))]$ for $i \in [\ell_w + 1, \ell_{\max}]$. Hence, the server can correlate this update with a previous search query. If the new key-value pair goes to the local stash, then the size of the stash will grow indefinitely. Moreover, if both $T_0[\mathsf{PRF}(K_w, (i||0))]$ and $T_1[\mathsf{PRF}(K_w, (i||1))]$ are already occupied, an eviction process will be triggered. It is not clear how to prevent potential leakage in eviction.

4.3 Our Approach

4.3.1 Another way to Make dprfMM [55] Dynamic.

Given the difficulties discussed above, we resort to a different approach [4] that (semi-generically) transforms a static data structure into a dynamic one).[9] The price to pay is increasing search complexity. In detail, the technique builds a DSSE scheme that holds N item using $\lfloor \log N \rfloor + 1$ instances of *static* SSE schemes of sizes $2^0, 2^1, \ldots, 2^{\lfloor \log N \rfloor}$ respectively. Denote these encrypted indexes as $\hat{EDB}_0, \ldots, \hat{EDB}_{\lfloor \log N \rfloor}$. These encrypted indexes are either empty or full, subject to the constraint that the total size of all non-empty ones is N. A new insertion will be written to $\hat{EDB}_0$ if $\hat{EDB}_0$ is empty. Otherwise, the server identifies the first empty index, say $\hat{EDB}_j$. Then, the server sends

[7] Kamara and Moataz [37] proposed alternative constructions that achieve optimal server storage size. But those constructions not only introduce a much larger search overhead, but also rely on non-standard computational assumptions.

[8] It has been shown [43] that with high probability, the n entries can be successfully placed in the Cuckoo hash table subject to the constraints above.

[9] This technique has been used to construct verifiable DSSE [7] and forward private DSSE with small client-side storage [17], hierechical ORAMs [2, 23, 54], and dynamic proofs-of-retrievability [60]. To the best of our knowledge, it has not been used to construct volume-hiding DSSE before.

all $\hat{EDB}_0, \ldots, \hat{EDB}_{j-1}$ (they must be all full) to the client, who merges them together with the entry to be inserted into a new encrypted index of size 2^j. $\hat{EDB}_0, \ldots, \hat{EDB}_{j-1}$ will be deleted and the originally empty $\hat{EDB}_j$ will be replaced by the one just created. Deletions are treated as inserting entries of the form $(w, (\mathsf{ID}, del))$. The client can locally filter out deleted entries during search. The complexity of the update protocol is $\Theta(\log N)$ amortized.

4.3.2 Handling Stashes.

Although we have made the data structure in the server dynamic, complications arise because of the existence of local stashes of Cuckoo hash tables. It turns out we need to carefully handle the failure probability of the underlying Cuckoo hash table due to stash overflow, for both correctness and efficiency. Recall that a Cuckoo hash table with a stash of size s fails if more than s items need to be inserted into the stash. The failure probability is $O(n^{-s})$ where n is the capacity of the Cuckoo hash table [43]. In the original dprfMM [55], the stash is set to be $O(1)$-size and stored locally as part of the secret key. Directly employing this approach in the VH-DSSE construction sketched above causes problems since we are using $O(\log N)$ instances of dprfMM on databases of sizes $2^0, \ldots, 2^{\lfloor \log N \rfloor}$. Firstly, the failure probability of a small-size Cuckoo hash table is non-negligible. Secondly, even if none of the $O(\log N)$ instances fails, the client still needs to locally store the stashes of size $O(\log N)$ in total.

For the first problem, we borrow the idea of "shared stash" in the ORAM literature [26,27]. Instead of allocating an $O(1)$-size stash for each Cuckoo hash table, we use a single $O(\log N)$-size stash. Although some of the Cuckoo hash tables may have $\omega(1)$ overflowed items, it has been shown [27, Theorem 2.1] that the total number of items in the shared stash is unlikely to exceed $O(\log N)$. To implement this idea, we modify dprfMM from [55] so that dprfMM.Setup does not abort even if the size of the stash is $\omega(1)$. For the sake of self-containedness, we present the modified dprfMM scheme in detail in Appendix C.

For the second problem, we can encrypt and upload this $O(\log N)$-size shared stash to the server. In the search protocol, the server always sends the $O(\log N)$-size encrypted stash to the client. This does not increase the search complexity, as the server needs to search $O(\log N)$ instances of encrypted indexes anyway. Meanwhile, when the update protocol rebuilds a particular instance of an encrypted index using dprfMM.Setup of size 2^j, the stashed elements corresponding to $\hat{EDB}_1, \ldots, \hat{EDB}_{j-1}$ should be replaced. A simple way is to always download the whole stash and then upload an updated version back to the server. The complexity of such stash handling in the update protocol is $\Theta(\log N)$. The overall asymptotic complexity of the update protocol remains $\Theta(\log N)$ amortized.

In VH-DSSE, we use a more efficient approach to handle the stash. At a high level, we introduce an amortization technique that can outsource this $O(\log N)$-size stash to the server while reducing the complexity of stash update to $O(1)$ (amortized). As a result, the constant behind the $O(\log N)$ cost of the update protocol is reduced. We make two observations here. First, we do not need to frequently update part of the $O(\log N)$-size stash that corresponds to a large-size Cuckoo hash table. In fact, they need to be updated only when a rebuild occurs. Second, rebuild of a large encrypted index is relatively infrequent: rebuilding an encrypted index of size $O(\log N)$ only happens after at least $O(\log N)$ updates. These observations give us room to amortize the cost of stash update to $O(1)$ by delaying rebuild of small-sized encrypted indices.

In more detail, we replace all encrypted indexes whose size is smaller than $\log N$ with a *single* list of size at most $2\lfloor \log N \rfloor - 1$, denoted by buf, which temporarily holds all recent updates until it is full. The update protocol is changed to simply sending the (randomized) encrypted keyword-document pair to the server, who appends the ciphertext to buf. Once buf is full after $O(\log N)$ updates, the server runs the original update protocol described before: (1) identifying the first empty encrypted index $\hat{EDB}_j$; (2) sending $\hat{EDB}_{j-1}$, $\hat{EDB}_{j-2}, \ldots$, together with buf and stash (both of size $O(\log N)$) to the client. The client builds an encrypted index $\hat{EDB}_j$ of size 2^j using $\mathsf{dprfMM.Setup}$, updates stash, and sends $\hat{EDB}_j, \mathsf{stash}$ and an empty buf to the server. It is not hard to see that the amortized stash update complexity is reduced to $O(1)$.

Remark Recently, Falk *et al.* [23] pointed out that a security flaw in some hierechical ORAMs that use a combined stash. Their attack applies to hierarchical ORAMs whose physical access pattern depends on whether an item is found in the combined stash or not. Our constructions do not behave differently according to the content in the combined stash so we are immune to the attack.

4.3.3 Achieving Volume-Hiding.

We have described how to make dprfMM dynamic but we have so far ignored the issue of volume-hiding. It turns out keeping the above design volume-hiding is quite tricky. If we were in the static setting, then we could make the response length equal to $\ell_{\max} = \max_{q \in \mathcal{W}} |DB(q)|$, just like in dprfMM. However, we are considering *dynamic* SSE, which means $\ell_{\max}$ may change over time. Naturally, $\ell_{\max}$ should be calculated for the *current* database DB, which requires the client to locally record the number of matching documents for *each* keyword, which is quite a large local storage overhead. More unfortunately, as we have discussed in Section 4.3, if the search leakage $\mathcal{L}^{Srch}(DB, q)$ contains $\mathcal{L}^{MRL}(DB) = \ell_{\max}$, it cannot satisfy our volume-hiding definition (*cf.*, Definition 3.1).

To deal with the stringent security requirement, we propose to make the response length of every search query independent of $\max_{w \in \mathcal{W}} |DB(w)|$ and w. There are multiple ways to instantiate this seemingly straightforward idea, but one should be careful when choosing the appropriate response length. For example, setting the response length to be some constant c will satisfy the above criteria, but as the encrypted database grows larger, the number of matching documents for many keywords may exceed this constant eventually. On the other hand, if we set it to be equal to the database size $|DB| = \sum_{w \in \mathcal{W}} |DB(w)|$, then the communication cost will be too high.

Our choice is to set the response length to grow (somewhat) linearly with respect to the *total number of updates* to the database. The rationale behind this is that when database size grows, naturally the *average* response length and the *maximum* response length will grow correspondingly. The server can estimate these two numbers by itself from the current size of the encrypted database. Our main observation is that volume-hiding does not mean *preventing* the server from making such estimations but just preventing the server from *knowing* how accurate its estimation is.

4.3.4 Correctness.

Of course, now that the server does not know how accurate its estimation on $\ell_{\max}$ is, it is inevitable that for some search queries, not all matching documents are returned. Since we cannot achieve $(1, 1)$-correctness (*cf.*, Definition 2.1), the best we may hope for is $(1 - \mathsf{negl}(\lambda), 1 - \mathsf{negl}(\lambda))$-correctness. It turns out this goal is still very challenging if we do not make extra assumptions on how the database is distributed. We leave it as an open question.

That said, if we allow the server to know the normalized frequency of the most common keyword $\ell_{\max}/|DB|$, then we can ask the server to return $2^i \cdot \ell_{\max}/|DB|$ entries from the encrypted index of size 2^i. The resulting scheme achieves $(1, \ell_{\max}/|DB|)$-correctness for all keywords. If we further assume that the keyword-ID pairs are uniformly distributed across all $O(\log N)$ instances of encrypted indexes, we can show that our (single-copy) scheme achieves $(1-\mathsf{negl}(\lambda), 1-o(1))$-correctness. Somewhat surprisingly, the proof turns out to be challenging. One of the reasons is that the distributions of the encrypted indexes are correlated: once we condition on the number of matching documents for keyword w in EDB_i, the number of matching documents for this keyword in other $EDB_{j,j\neq i}$ will change. This makes the probability analysis more difficult than one may expect. See the outline of the proof in Section 6.1 for details.

We argue that leaking (the initial) $\ell_{\max}/|DB|$ to the server is a reasonable and mild assumption for the following reasons. First, it is commonly assumed in the literature that the adversary has prior knowledge (say, the distribution) about the plaintext database *before* seeing EDB and search/update queries [9,30,51,57] and $\beta = \ell_{max}/|DB|$ is a small part of database distribution. Leaking (the initial) β is not the best we can hope for but at least it does not give the adversary extra information in such scenarios. Second, prior work [37] assumes a stronger assumption that the keywords follow Zipf-distribution to derive a better efficiency guarantee.[10] Under their assumption, the server knows the normalized frequency of *all* keywords.

To further amplify correctness, we can run k copies of the (single-copy) scheme and take the union of the answers. The resulting multi-copy scheme $\mathsf{VH\text{-}DSSE}^k$ achieves $(1-\mathsf{negl}(\lambda), 1-\left(\frac{2}{3}\right)^{\Theta(k)})$-correctness. If we set $k=\omega(\log\lambda)$, then we get $(1-\mathsf{negl}(\lambda), 1-\mathsf{negl}(\lambda))$-correctness.

5 Our Single-Copy VH-DSSE Construction

5.1 Formal Description.

Figures 2 to 4 give a formal description of VH-DSSE, which is parameterized by a positive constant $\beta \leq 1$ and a function $f(n) \in O(\log n)$. dprfMM makes use of a modified version of dprfMM [55]. Apart from the difference in stash handling that we described above, we further modify the interface of dprfMM.Search (as well as the underlying delegatable PRF, see Section A) to make the response length an explicit parameter for both parties. See the complete description of our modified dprfMM in Appendix C for details.

The function $f(n)$ is the upper bound of the size of the combined stash. The parameter β controls the response length of a search query (line 3 of VH-DSSE.Search). In short, VH-DSSE returns a β-fraction of the database for each search query. $\beta = 1$ means always returning the whole database. We assume the client and the server share prior knowledge of the normalized frequency of the most common keyword in DB, (*i.e.*, $\frac{\ell_{\max}}{|DB|} = \frac{\max_{w\in\mathcal{W}}|DB(w)|}{\sum_{w\in\mathcal{W}}|DB(w)|}$), and set β to be this value. [11]

5.2 Leakage Function of VH-DSSE

We formally describe the leakage $\mathcal{L} = (\mathcal{L}^{Stp}_{\mathsf{VH\text{-}DSSE}}, \mathcal{L}^{Srch}_{\mathsf{VH\text{-}DSSE}}, \mathcal{L}^{Updt}_{\mathsf{VH\text{-}DSSE}})$ and a high-level discussion on why VH-DSSE *only* leaks this information, We then briefly argue why this $\mathcal{L}$ satisfies forward and backward privacy and our definition of volume-hiding. Formal security proofs will be given in Section 5.3.

[10] We recall the definition of Zipf-distribution in Appendix B.2.

[11] For a key-value pair database following the Zipf-distribution (*cf.* Definition B.1 in Appendix B.2), β can be computed directly as $\frac{1}{H_{|\mathcal{W}|,1}} = (1+\frac{1}{2}+\cdots+\frac{1}{|\mathcal{W}|})^{(-1)}$.

```
$(K, \tau, EDB) \leftarrow \mathsf{VH\text{-}DSSE.Setup}(1^\lambda, DB)$
1 :  Set $EDB, K, \tau$ to be empty vectors; Set $\mathsf{buf}, \mathsf{stash}$ to be empty lists
2 :  Let $N = |DB|, kv_1, \ldots, kv_N$ denote the keyword-document pairs in $DB$
3 :  Randomly select two encryption keys $K_{\mathsf{stash}}, K_{\mathsf{buf}} \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$
4 :  for $i = 1$ to $N$ do :
5 :    Parse $kv_i$ as $(w', \mathsf{ID}')$
6 :    Set $kv_i$ as $(w', (\mathsf{ID}', add))$
7 :  Randomly permute the order of $kv_1, \ldots, kv_N$
8 :  Let $\mathsf{idx} = 1$, and $b_{\lfloor \log N \rfloor} \cdots b_0$ be the binary representation of $N$
9 :  Let $\mathsf{min}$ be the integer such that $2^{\mathsf{min}} \leq \log N < 2^{\mathsf{min}+1}$
10 : for $i = \lfloor \log N \rfloor$ to $\mathsf{min}$ do :
11 :   if $b_i = 1$ then
12 :     Let $DB_i = \{kv_{\mathsf{idx}}, \ldots, kv_{\mathsf{idx}+2^i-1}\}$
13 :     $(K[i], \tau[i], \hat{EDB}_i) \leftarrow \mathsf{dprfMM.Setup}(1^\lambda, DB_i)$
14 :     Parse $K[i]$ as $(K_{\mathsf{PRF}}[i], K_{\mathsf{SKE}}[i], \mathsf{stash}[i])$
15 :     Append $\mathsf{stash}[i]$ to $\mathsf{stash}$
16 :     $K[i] := (K_{\mathsf{PRF}}[i], K_{\mathsf{SKE}}[i]), \mathsf{idx} \mathrel{+}= 2^i$
17 : // Handle stash and the remaining $f(N)$ items
18 : for $i = \mathsf{idx}$ to $N$ do :
19 :   Append $ct = \mathsf{SKE.Enc}(K_{\mathsf{buf}}, kv_i)$ to $\mathsf{buf}$
20 : foreach $kv' \in \mathsf{stash}$ :
21 :   $kv' := \mathsf{SKE.Enc}(K_{\mathsf{stash}}, kv')$
22 : Pad $\mathsf{stash}$ with dummy ciphertexts so that there are $f(N)$ elements
23 : Compute $\beta = \frac{\max_{q \in \mathcal{W}} |DB(q)|}{N}$    // Normalized freq. of the most common keyword
24 : Set $EDB = (\beta, \mathsf{buf}, \mathsf{stash}, (\hat{EDB}_{\mathsf{min}}, \ldots, \hat{EDB}_{\lfloor \log N \rfloor}))$
25 : Set $K = (K_{\mathsf{stash}}, K_{\mathsf{buf}}, (K[\mathsf{min}], \ldots, K[\lfloor \log N \rfloor])), \tau = N$
26 : return $(K, \tau, EDB)$
```

Figure 2: VH-DSSE.Setup Algorithm

5.2.1 Setup leakage:

$\mathcal{L}^{Stp}_{\mathsf{VH\text{-}DSSE}} = (\mathcal{L}^{DSize}, \beta) = (|DB|, \frac{\max_{w \in \mathcal{W}} |DB(w)|}{|DB|})$.

First, β is explicitly revealed to the server to control how many results should be returned in VH-DSSE.Search. Second, when VH-DSSE.Setup is executed on a database of size $|DB| = N$, it invokes dprfMM.Setup for $O(\log N)$ times to build $(\hat{EDB}_{\min}, \ldots, \hat{EDB}_{\lfloor \log N \rfloor})$ and SKE.Enc to encrypt $O(\log N)$ key-value pairs into stash and buf. The size of stash and buf is determined by N. As SKE is RoR-secure, stash and buf do not leak information to the server and the leakage of VH-DSSE.Setup is the union of the leakage from dprfMM.Setup. Note that the only leakage of dprfMM.Setup is the database size. Moreover, the exact number of invocations of dprfMM.Setup as well as the size of the input database to dprfMM.Setup are both uniquely determined by $|DB| = N$. Therefore, the setup leakage of $\mathcal{L}^{Stp}_{\mathsf{VH\text{-}DSSE}}$ can be uniquely described by $(\mathcal{L}^{DSize}, \beta)$.

```
$\mathsf{VH\text{-}DSSE.Search}(K, q, \tau; EDB)$
    Client ⟷ Server
1 :  Set $\{\mathcal{X}_0, \mathcal{X}_1, \ldots\}$ as a list of empty sets
2 :  for all $i$ such that $\hat{EDB}_i \neq \phi$ do :
3 :      $\ell_i = \lceil \beta 2^i \rceil$
4 :      $\mathcal{X}_i \leftarrow \mathsf{dprfMM.Search}(K[i], q, \tau[i], \ell_i; \hat{EDB}_i, \ell_i)$
5 :  Send $(\{\mathcal{X}_0, \mathcal{X}_1, \ldots\}, \mathsf{stash}, \mathsf{buf})$ to the client
     Client:
6 :  Let $\mathcal{X}$ be an empty set
7 :  for all $\mathcal{X}_i \in \{\mathcal{X}_0, \mathcal{X}_1, \ldots\}$ such that $\mathcal{X}_i \neq \phi$
8 :      $\mathcal{X} = \mathcal{X} \cup \mathsf{DecAll}(K[i], \mathcal{X}_i)$
9 :  $\mathcal{X} = \mathcal{X} \cup \mathsf{DecAll}(K_{\mathsf{stash}}, \mathsf{stash}) \cup \mathsf{DecAll}(K_{\mathsf{buf}}, \mathsf{buf})$
10 : Remove dummy entries in $\mathcal{X}$
11 : return $DB(q) \leftarrow \{\mathsf{ID} | (w, (\mathsf{ID}, add)) \in \mathcal{X} \wedge (w, (\mathsf{ID}, del)) \notin \mathcal{X}\}$
```

Figure 3: VH-DSSE.Search Protocol

5.2.2 Search leakage:

$\mathcal{L}^{Srch}_{\mathsf{VH\text{-}DSSE}} = (\ldots, M_{i-1}, M_i, M_{i+1}, \ldots)$ where M_i is a binary matrix defined as follow. Let c_i denotes the number of search queries when the size of the largest non-empty index in EDB is 2^i and let $Q = (\ldots, \vec{q}_{i-1}, \vec{q}_i, \vec{q}_{i+1}, \ldots)$ be the set of all search queries, where $\vec{q}_i = (q^{(i)}_1, q^{(i)}_2, \ldots, q^{(i)}_{c_i})$. M_i is a $c_i \times c_i$ binary matrix and $M_i[x][y] = 1$ iff $q^{(i)}_x = q^{(i)}_y$ for $x, y \in [1, c_i]$.

To see why $\mathcal{L}^{Srch}_{\mathsf{VH\text{-}DSSE}}$ is in this form, recall that VH-DSSE.Search invokes dprfMM.Search on all non-empty instances in EDB and $\mathcal{L}^{Srch}_{\mathsf{dprfMM}} = \mathcal{L}^{Eq}$ only. Suppose $q^{(i)}_{t_1}, q^{(j)}_{t_2}$ are two search queries ($i \leq j$) and $q^{(i)}_{t_1}$ (resp. $q^{(j)}_{t_2}$) was issued when the index of the largest non-empty encrypted index is i (resp. j). If $i = j$, the server can learn if $q^{(i)}_{t_1} = q^{(j)}_{t_2}$ from $\mathcal{L}^{Srch}_{\mathsf{dprfMM}}$. If $i < j$, then all the non-empty encrypted indexes at time t_1 must have been rebuilt using freshly chosen keys from dprfMM.Setup. The search tokens for $q^{(i)}_{t_1}$ and $q^{(j)}_{t_2}$ will therefore be independent, making it impossible to learn $q^{(i)}_{t_1} \overset{?}{=} q^{(j)}_{t_2}$. Hence, the search pattern leakage in VH-DSSE is not a full $|Q| \times |Q|$ binary matrix as the standard search pattern leakage $\mathcal{L}^{Eq}$ in the literature. Effectively, $\mathcal{L}^{Srch}_{\mathsf{VH\text{-}DSSE}}$ is only the list of small square sub-matrices at the diagonal of $\mathcal{L}^{Eq}$. In the extreme case where $c_j = 1$ for all j, $\mathcal{L}^{Srch}_{\mathsf{VH\text{-}DSSE}}$ is reduced to *only* the diagonal entries in the $|Q| \times |Q|$ search pattern binary matrix $\mathcal{L}^{Eq}$, namely the search pattern is completely hidden in such cases. We also stress that $\mathcal{L}^{Srch}_{\mathsf{VH\text{-}DSSE}}$ does not reveal the timestamps of the insertions/deletions related to w.

5.2.3 Update leakage:

$\mathcal{L}^{Updt}_{\mathsf{VH\text{-}DSSE}} = \perp$.

To see, when an update query $(op, (w, \mathsf{ID}))$ is issued on the encrypted database EDB, the server always receives a new ciphertext ct and appends it to buf. If buf is not full ($|\mathsf{buf}| \neq 2\lfloor \log N \rfloor$) at this point, then VH-DSSE.Update completes. As SKE is RoR-secure, ct does not reveal any information to the server. If buf is full, and let $\hat{EDB}_i$ be the first empty index in the current $EDB = (\hat{EDB}_{\min}, \ldots, \hat{EDB}_{\lfloor \log N \rfloor})$. The server

```
$\mathsf{VH\text{-}DSSE.Update}(K, op, (w, \mathsf{ID}), \tau; EDB)$
    Client:
1 :  Set $kv' = (w, (\mathsf{ID}, op))$
2 :  Send $ct = \mathsf{SKE.Enc}(K_{\mathsf{buf}}, kv')$ to the server, $\tau$ += 1
    Server:
3 :  Parse $EDB$ as $(\beta, \mathsf{buf}, \mathsf{stash}, (\hat{EDB}_{\mathsf{min}}, \ldots, \hat{EDB}_{\lfloor \log N \rfloor}))$
4 :  Append $ct$ to $\mathsf{buf}$
5 :  Let $\mathsf{min}$ be the integer such that $2^{\mathsf{min}} \leq \log N < 2^{\mathsf{min}+1}$
6 :  if $|\mathsf{buf}| = 2^{\mathsf{min}+1}$ then      // buf is full
7 :    Find the minimum $j$ such that $\hat{EDB}_j = \phi$
8 :    Send $(\mathsf{buf}, \mathsf{stash}, (\hat{EDB}_{\mathsf{min}}, \ldots, \hat{EDB}_{j-1}))$ to the client
    Client:
9 :  Set $\mathcal{X} \leftarrow \phi$
10 : for $i = \mathsf{min}$ to $j - 1$ do :
11 :   Parse $\hat{EDB}_i$ as $(T_0, T_1)$
12 :   $\mathcal{X} \leftarrow \mathcal{X} \cup \mathsf{SKE.DecAll}(K[i], T_0) \cup \mathsf{SKE.DecAll}(K[i], T_1)$
13 :   $K[i] \leftarrow \perp, \mathsf{stash}[i] = \perp$
14 : Let $DB_j = \mathcal{X} \cup \mathsf{SKE.DecAll}(K_{\mathsf{buf}}, \mathsf{buf}) \cup \mathsf{SKE.DecAll}(K_{\mathsf{stash}}, \mathsf{stash})$
15 : for all $kv = (w, (\mathsf{ID}, add)) \in DB_j$
16 :   if $\exists kv' = (w, (\mathsf{ID}, del)) \in DB_j$
17 :     Replace both of them with dummy entries
18 : $(K[j], \tau[j], \hat{EDB}_j) \leftarrow \mathsf{dprfMM.Setup}(1^\lambda, DB_j)$
19 : Parse $K[j]$ as $(K_{\mathsf{PRF}}[j], K_{\mathsf{SKE}}[j], \mathsf{stash}[j])$
20 : foreach $kv' \in \mathsf{stash}[j]$ :
21 :   $kv' := \mathsf{SKE.Enc}(K_{\mathsf{stash}}, kv')$
22 : Update the $j$-th entries in $K, \mathsf{stash}$ with $((K_{\mathsf{PRF}}[j], K_{\mathsf{SKE}}[j]), \mathsf{stash}[j])$
23 : Re-encrypt all entries in $\mathsf{stash}[i]$ $\forall\ i \neq j$ and pad $\mathsf{stash}$ to $f(N)$ elements
24 : Send $(\hat{EDB}_j, \mathsf{stash})$ to the server
    Server:
25 : Store $\hat{EDB}_j$ and $\mathsf{stash}$; set $\mathsf{buf} \leftarrow \phi$
26 : for $i = \mathsf{min}$ to $j - 1$ do
27 :   Set $\hat{EDB}_i \leftarrow \phi$
```

Figure 4: VH-DSSE.Update Protocol

sends back $(\mathsf{stash}, \mathsf{buf}, (\hat{EDB}_{\min}, \ldots, \hat{EDB}_{i-1}))$ to the client, who runs dprfMM.Setup on a database of size 2^i. The client will send the resulting encrypted index $\hat{EDB}_i$ and updated stash, buf back to the server. The only leakage $\mathcal{L}^{Stp}_{\mathsf{dprfMM}} = |DB| = 2^i$ is due to dprfMM.Setup. But this information is known by the server before seeing the update query anyway, as it knows the first empty index is $\hat{EDB}_i$. Therefore, the update leakage is $\perp$.

5.2.4 Forward and Backward Privacy:

Given the search and update leakage described above, it is not hard to see that our scheme satisfies forward privacy (Definition 2.3), as the update leakage $\mathcal{L}^{Updt}_{\mathsf{VH\text{-}DSSE}} = \perp$ is independent of the updated keyword w and the update operation op. Similarly, our scheme is both BP-I and BP-I* backward-private (Definition 2.4), which is the first one (without relying on ORAM) in the literature, to the best of our knowledge.

5.2.5 Volume-Hiding:

Informally, VH-DSSE is volume-hiding (Definition 3.2) as the leakage profile $\mathcal{L}_{\mathsf{VH\text{-}DSSE}} = (\mathcal{L}^{Stp}_{\mathsf{VH\text{-}DSSE}}, \mathcal{L}^{Srch}_{\mathsf{VH\text{-}DSSE}}, \mathcal{L}^{Updt}_{\mathsf{VH\text{-}DSSE}})$ does not include the response length leakage.

5.3 Security Proofs

Theorem 5.1. *Our* VH-DSSE *construction in Figures 2 to 4 is adaptively-secure with respect to the leakage function* $\mathcal{L}_{\mathsf{VH\text{-}DSSE}} = (\mathcal{L}^{Stp}_{\mathsf{VH\text{-}DSSE}}, \mathcal{L}^{Srch}_{\mathsf{VH\text{-}DSSE}}, \mathcal{L}^{Updt}_{\mathsf{VH\text{-}DSSE}})$*, assuming* SKE *is* RoR*-secure and* dprfMM *is adaptively-secure with respect to the leakage function* $\mathcal{L}'_{\mathsf{dprfMM}} = (\mathcal{L}^{DSize}, \mathcal{L}^{Eq})$.[12]

The proof of Theorem 5.1 is given in Appendix E.

Theorem 5.2. VH-DSSE*, whose leakage profile being* $\mathcal{L}_{\mathsf{VH\text{-}DSSE}} = (\mathcal{L}^{Stp}_{\mathsf{VH\text{-}DSSE}}, \mathcal{L}^{Srch}_{\mathsf{VH\text{-}DSSE}}, \mathcal{L}^{Updt}_{\mathsf{VH\text{-}DSSE}})$*, is volume-hiding.*

Proof. We have shown that VH-DSSE is adaptively-secure with respect to the leakage function $\mathcal{L}_{\mathsf{VH\text{-}DSSE}} = (\mathcal{L}^{Stp}_{\mathsf{VH\text{-}DSSE}}, \mathcal{L}^{Srch}_{\mathsf{VH\text{-}DSSE}}, \mathcal{L}^{Updt}_{\mathsf{VH\text{-}DSSE}})$. It remains to show that $\mathcal{L}_{\mathsf{VH\text{-}DSSE}}$ is a volume-hiding leakage profile according to Definition 3.1.

To do so, we argue that $\mathcal{L}_{\mathsf{VH\text{-}DSSE}}$ is identically distributed regardless of the value b in the $\mathsf{VolH}_{\mathcal{A},\mathcal{L}}(1^\lambda)$ game. To see, in the setup phase of the $\mathsf{VolH}_{\mathcal{A},\mathcal{L}}(1^\lambda)$ game, the adversary $\mathcal{A}$ chooses two databases DB_0, DB_1 with the same size and the same maximum volume. Therefore, it sees the same $\mathcal{L}^{Stp}$ leakage in the setup phase regardless of b. The update leakage $\mathcal{L}^{Updt}$ is $\perp$, which is also identical regardless of the bit b. The search leakage $\mathcal{L}^{Srch}$ only depends on the search pattern. Note that in the query phase, the challenger always returns $\mathcal{L}^{Srch}(DB, w_0)$ to $\mathcal{A}$ (it ignores $q_1 = w_1$). So the search pattern will also be the same regardless of b, thus completing the proof. □

Theorem 5.3. VH-DSSE *is forward-private, and both BP-I and BP-I* backward-private.*

Proof. Forward-privacy is straightforward as $\mathcal{L}^{Stp}_{\mathsf{VH\text{-}DSSE}} = \perp$. For backward-privacy, note that if the search leakage $\mathcal{L}^{Srch}$ of a DSSE scheme can be derived from $(\mathbf{TimeDB}(w), a_w)$ (resp. $\mathbf{Updates}(w)$), then it is BP-I (resp. BP-I*) backward-private. Both $(\mathbf{TimeDB}(w), a_w)$ and $\mathbf{Updates}(w)$ imply $\mathcal{L}^{Eq}$ while $\mathcal{L}^{Srch}_{\mathsf{VH\text{-}DSSE}}$ only contains a subset of $\mathcal{L}^{Eq}$ as discussed in Section 5.2. Hence, BP-I and BP-I* backward-privacy follow immediately. □

6 Efficiency and Correctness of VH-DSSE

6.1 Efficiency

VH-DSSE outputs an encrypted index $EDB = (\beta, \mathsf{buf}, \mathsf{stash}, (\hat{EDB}_{\mathsf{min}}, \ldots, \hat{EDB}_{\lfloor \log N \rfloor}))$, where $\hat{EDB}_i$ is the output of dprfMM.Setup on a database of size 2^i. The size of buf and stash are both $\Theta(\log N)$. All entries in DB either resides in one of $\hat{EDB}_i$ or in buf or in stash. Because the server storage of dprfMM is linear, VH-DSSE has server storage

[12] See Appendix C for a discussion on the leakage function $\mathcal{L}'_{\mathsf{dprfMM}}$.

$\Theta(N)$. Notice that because we use lazy deletion strategy, the size of the (current) database $N = N_0 + k$ where N_0 is the size of the initial database and k is total number of update operations so far.

In terms of client-side storage, VH-DSSE requires the client to locally store one key per $E\hat{D}B_i$ instance. It is straightforward to reduce the local storage to $\Theta(1)$ by generating the key for each instance using a PRF. The security proof can be adjusted easily by introducing an additional hybrid that replaces this PRF with a random function.

VH-DSSE.Search returns $(\mathsf{buf}, \mathsf{stash})$ to the client and invokes dprfMM.Search on each $E\hat{D}B_i$ instance with length parameter $\ell_i = \lceil \beta 2^i \rceil$. The search complexity (both computation and communication) of dprfMM.Search is $\Theta(\ell_i)$. Assuming $\beta = \frac{\max_{w \in \mathcal{W}} |DB(w)|}{|DB|} = \frac{\ell_{\max}}{|DB|}$, then the search complexity of VH-DSSE is $\Theta(\ell_{\max} + \log N)$.

As for update complexity, after 2^j updates on an initially empty database, the client has run dprfMM.Setup once for a database of size 2^j, twice for 2^{j-1}, *etc.*, all the way down to $2^{j-\mathsf{min}}$ times for size 2^{min}, where $\mathsf{min} \in \Theta(\log \log N)$. The computational complexity of dprfMM.Setup is linear, so the amortized cost of dprfMM.Setup-related operations per update after N update is $\Theta(\log N)$. In the meantime, the client rebuilds $\Theta(\log N)$-size stash after every $\Theta(\log N)$ updates. As discussed in Section 4.3, the amortized complexity of stash handling is $\Theta(1)$. Overall, the update complexity (both computation and communication) of VH-DSSE.Update is $\Theta(\log N)$ amortized.

A summary of the efficiency of VH-DSSE is given in Table 1.

6.2 Correctness

As discussed in Section 4.3, VH-DSSE does not achieve perfect correctness. However, we can still prove that VH-DSSE achieves certain correctness guarantees.

6.2.1 Worst-case Correctness

Theorem 6.1 (Worst-Case Correctness)**.** VH-DSSE *achieves the following worst-case correctness guarantees:*

1. $(1, \beta)$*-correctness for all keywords;*
2. $(1, 1)$*-correctness for keyword w if its volume $\ell_w \leq \lceil \beta \log |DB| \rceil$.*

Proof. The proof is straightforward. For all search queries, VH-DSSE.Search always sends $\mathsf{buf}, \mathsf{stash}$ to the client and returns $\lceil \beta 2^i \rceil$ entries from $E\hat{D}B_i$ of size 2^i from dprfMM.Search. The correctness of dprfMM guarantees that if there are $\ell_{w,i}$ matching documents for keyword w in $E\hat{D}B_i$, dprfMM.Search returns $\min(\ell_{w,i}, \lceil \beta 2^i \rceil)$ ones from $E\hat{D}B_i$ (and the smallest EDB_i is of size $\log N$).

Let $\ell_{w,\mathsf{buf}}$ and $\ell_{w,\mathsf{stash}}$ be the number of matching documents for keyword w in buf and stash, respectively. The total number of matching documents returned by VH-DSSE is $\tilde{\ell}_w = \ell_{w,\mathsf{buf}} + \ell_{w,\mathsf{stash}} + \sum \min(\ell_{w,i}, \lceil \beta 2^i \rceil)$. If $\ell_w \leq \lceil \beta \log N \rceil$, then we have $\min(\ell_{w,i}, \lceil \beta 2^i \rceil) \geq \ell_{w,i}$ for all i. Hence,

$$\begin{aligned} \tilde{\ell}_w &= \ell_{w,\mathsf{buf}} + \ell_{w,\mathsf{stash}} + \sum \min(\ell_{w,i}, \lceil \beta 2^i \rceil) \\ &\geq \ell_{w,\mathsf{buf}} + \ell_{w,\mathsf{stash}} + \sum \ell_{w,i} = \ell_w. \end{aligned}$$

Therefore, VH-DSSE achieves $(1, 1)$-correctness for keyword w if $\ell_w \leq \lceil \beta \log N \rceil$.

Let $\mathcal{I}'$ denote the set of indices such that $\ell_{w,i} \leq \lceil \beta 2^i \rceil, i \in \mathcal{I}'$. We can bound the ratio $\frac{\tilde{\ell}_w}{\ell_w}$ as follows:

$$\begin{aligned}\frac{\tilde{\ell}_w}{\ell_w} &= \frac{\ell_{w,\mathsf{buf}} + \ell_{w,\mathsf{stash}} + \sum_{i\in\mathcal{I}'} \min(\ell_{w,i}, \lceil\beta 2^i\rceil) + \sum_{i\notin\mathcal{I}'} \min(\ell_{w,i}, \lceil\beta 2^i\rceil)}{\ell_{w,\mathsf{buf}} + \ell_{w,\mathsf{stash}} + \sum_{i\in\mathcal{I}'} \ell_{w,i} + \sum_{i\notin\mathcal{I}'} \ell_{w,i}} \\ &= \frac{(\ell_{w,\mathsf{buf}} + \ell_{w,\mathsf{stash}} + \sum_{i\in\mathcal{I}'} \ell_{w,i}) + \sum_{i\notin\mathcal{I}'} \lceil\beta 2^i\rceil}{(\ell_{w,\mathsf{buf}} + \ell_{w,\mathsf{stash}} + \sum_{i\in\mathcal{I}'} \ell_{w,i}) + \sum_{i\notin\mathcal{I}'} \ell_{w,i}} \geq \frac{\sum_{i\notin\mathcal{I}'} \lceil\beta 2^i\rceil}{\sum_{i\notin\mathcal{I}'} 2^i} \geq \beta\end{aligned}$$

The last but one inequality holds because $(\ell_{w,\mathsf{buf}}+\ell_{w,\mathsf{stash}}+\sum_{i\in\mathcal{I}'}\ell_{w,i}) \geq 0$ and $\ell_{w,i} \leq 2^i$ for all i. The lower-bound is reached when all ℓ_w matching documents for w fully occupy one or more $\hat{EDB}_i$ instances. Hence, VH-DSSE guarantees that at least β-fraction of matching documents will be returned for all keywords. □

6.2.2 Average-case Correctness

Theorem 6.1 analyzes the correctness guarantee of VH-DSSE in the worst-case scenario, in which the matching documents of a particular keyword are concentrated in some $\hat{EDB}_i$ instances. In practice, we expect that the matching documents are evenly distributed across all $\hat{EDB}_i$ because in Step (7) of VH-DSSE.Setup, a random permutation is applied before the entries are assigned into $\hat{EDB}_i$. Therefore, VH-DSSE achieves a much better correctness guarantee in the average sense. We formalize it in Theorem 6.2 below.

Theorem 6.2 (Average-Case Correctness)**.** *Let DB be a database such that $\ell_w = |DB(w)| \in O(|DB|)$ for all $w \in \mathcal{W}$, where ℓ_w is the number of matching documents for keyword w. When* VH-DSSE *is applied to DB, it achieves* $(1-\mathsf{negl}(\lambda), 1-o(1))$*-correctness.*

The proof of Theorem 6.2 turns out to be surprisingly complicated. The extra assumption that $\ell_w \in O(|DB|)$ is relevant to some technical difficulties in the proof.[13] For ease of exposition, we first outline a natural and simple proof strategy. We discuss the difficulties we encounter when instantiating the proof strategy into a complete proof, before presenting the actual one.

6.2.3 Outline of Proof of Theorem 6.2 and Technical Challenges.

We will use the following notations. Let DB denote a database of $N = |DB| = \sum_{w\in\mathcal{W}} |DB(w)|$ keyword-document pairs, and $EDB = (\hat{EDB}_{\min}, \hat{EDB}_{\min+1}, \ldots)$ denote the output encrypted database from VH-DSSE.Setup$(1^\lambda, DB)$. Let $\mathcal{I} = \{i_{m_1}, i_{m_2}, \ldots\}$ be the set of indices such that $i \in \mathcal{I}$ iff $\hat{EDB}_i \neq \phi$. Without loss of generality, let's assume $i_{m_1} > i_{m_2} > \ldots$ in decreasing order. Let $X_{w,i}$ denote the random variable counting the number of matching documents for keyword w in $\hat{EDB}_i$, where the randomness comes from VH-DSSE.Setup.

Recall that in VH-DSSE.Search, the server always returns $\lceil\beta 2^i\rceil = \lceil\ell_{\max}\cdot 2^i/N\rceil$ entries from $\hat{EDB}_i$. By the correctness of the underlying dprfMM scheme, the number of matching documents returned by the server from $\hat{EDB}_i$ for the search keyword w is $\min(X_{w,i}, \lceil\ell_{\max}\cdot 2^i/N\rceil)$. To prove that VH-DSSE achieves $(1-\mathsf{negl}(\lambda), 1-o(1))$ correctness, it suffices to show that $X_{w,i} \geq E[X_{w,i}](1-o(1))$ for all $i \in \mathcal{I}$ with probability $1-\mathsf{negl}(\lambda)$. Further notice that $X_{w,i}$ follows hypergeometric distribution $H(2^i, N, \ell_w)$ for $i \in \mathcal{I}$. We can then use the well-known tail-bound of hypergeometric distribution (*cf.*, Lemma B.1 in Appendix B.1) to compute $\Pr[X_{w,i} \geq E[X_{w,i}]-\delta_i]$ for some carefully selected value $\delta_i \in o(E[X_{w,i}])$. Let $E_{w,i}$ denote the event that $X_{w,i} \geq E[X_{w,i}]-\delta_i$. The proof of Theorem 6.2 boils down to showing $\Pr[\bigwedge_{i\in\mathcal{I}} E_{w,i}] > 1-\mathsf{negl}(\lambda)$.

[13]This assumption holds for many database distributions (*e.g.*, Zipf-distribution) in practice.

While the intuition above is simple, instantiating it into a complete proof is non-trivial as we face at least three challenges below. First, although we know the largest index $i_{m_1} \in \mathcal{I}$ satisfies $N/2 < 2^{i_{m_1}} \leq N$, the second largest index $i_{m_2} \in \mathcal{I}$ can range from 1 to $i_{m_1} - 1$. Thus, we are not able to give a useful bound on $2^{i_{m_2}}$ (*i.e.*, the size of $\hat{EDB}_{i_{m_2}}$) and $E[X_{w,i_{m_2}}] = \ell_w \cdot 2^{i_{m_2}}/N$. It is not clear how to use the tail-bound of hypergeometric distribution to estimate $\Pr[E_{w,i_{m_2}}]$ (and in general $\Pr[E_{w,i}]_{i\in\mathcal{I},i\neq i_{m_1}}$) without knowing the concrete value of i_{m_2}. Second, by careful computation, one can verify that $\Pr[E_{w,i}] = \Pr[X_{w,i} \geq E[X_{w,i}] - \delta_i] > 1 - \mathsf{negl}(\lambda)$ *does not* hold for small i (say, $2^i \in o(N)$), if $\delta_i \in o(E[X_{w,i}])$. Third, even if we can compute $\Pr[E_{w,i}]$ for all $i \in \mathcal{I}$ and make sure that they are all larger than $1 - \mathsf{negl}(\lambda)$, we should notice that $\Pr[\bigwedge_{i\in\mathcal{I}} E_{w,i}] \neq \prod_{i\in\mathcal{I}} \Pr[E_{w,i}]$ because the events $E_{w,i}$ are dependent.[14]

These difficulties complicate the probability analysis in our actual proof. First, instead of computing $\Pr[\bigwedge_{i\in\mathcal{I}} E_{w,i}]$ directly (the third challenge), we will prove by induction that if $\Pr[\bigwedge_{j=1}^{k-1} E_{w,i_{m_j}}] > 1 - \mathsf{negl}(\lambda)$, then $\Pr[E_{w,i_{m_k}} | \bigwedge_{j=1}^{k-1} E_{w,i_{m_j}}] > 1 - \mathsf{negl}(\lambda)$. This is sufficient to show that $\Pr[\bigwedge_{j=1}^{k} E_{w,i_{m_j}}] = \Pr[E_{w,i_{m_k}} | \bigwedge_{j=1}^{k-1} E_{w,i_{m_j}}] \cdot \Pr[\bigwedge_{j=1}^{k-1} E_{w,i_{m_j}}] > 1 - \mathsf{negl}(\lambda)$. Second, our induction stops as long as $N - \sum_{j=1}^{k} 2^{i_{m_j}} \in o(N)$ instead of exhausting all indices $i \in \mathcal{I}$. The rationale behind is the following observation: $1/2 \cdot (N - \sum_{j=1}^{k-1} 2^{i_{m_j}}) < 2^{i_{m_k}} \leq N - \sum_{j=1}^{k-1} 2^{i_{m_j}}$. This guarantees that during induction, we always have $2^{i_{m_k}} \in O(N)$. Hence, we can estimate the size of $\hat{EDB}_{i_{m_k}}$ (the first challenge) and at the same time avoid the issue that $\Pr[E_{w,i}] < 1 - \mathsf{negl}(\lambda)$ for small i (the second challenge). Finally, the revised proof strategy above requires us to give not only a lower-bound of $X_{w,i}$ but also an upper-bound. Therefore, we need to replace $E_{w,i}$ with $\hat{E}_{w,i}$ denoting the event that $E[X_{w,i}] - \delta_i \leq X_{w,i} \leq E[X_{w,i}] + \delta_i$.

6.2.4 Proof of Theorem 6.2.

Proof. In VH-DSSE.Setup, a random permutation is applied to DB before they are partitioned into sets of sizes of power of 2 (Line 3, Fig. 2). This is equivalent to selecting the key-value pairs into $\hat{EDB}_{i_{m_1}}, \hat{EDB}_{i_{m_2}}, \ldots$ without replacement. Hence, $X_{w,i}$ follows hypergeometric distribution $H(2^i, N, \ell_w)$ for $i \in \mathcal{I}$. Let $EX_{w,i} = E[X_{w,i}] = \ell_w \cdot 2^i/N$ be the expected value of $X_{w,i}$. The core of our proof is the following proposition regarding the random variables $\{X_{w,i}\}_{i\in\mathcal{I}}$.

Proposition 6.3. *For all k such that $N - \sum_{j=1}^{k} 2^{i_{m_j}} \in O(N)$ and $i_{m_k} \in \mathcal{I}$,*

$$\Pr[\bigwedge_{j=1}^{k} \hat{E}_{w,i_{m_j}}] > 1 - \mathsf{negl}(\lambda), \qquad (*)$$

where $\hat{E}_{w,i_{m_j}}$ denote the event that $-\delta_j \leq X_{w,i_{m_j}} - EX_{w,i_{m_j}} \leq \delta_j$ for some $\delta_j \in o(EX_{w,i_{m_j}})$.

Given Proposition 6.3 (which will be proven in Appendix F), it is easy to prove Theorem 6.2 as follow. Notice that in VH-DSSE.Search, the server always returns $\lceil \ell_{\max} \cdot 2^i/N \rceil$ key-value pairs from $\hat{EDB}_i$, and the total number of matching documents for keyword w in $\hat{EDB}_i$ is $X_{w,i}$. Therefore, the total number of matching documents returned by the server from EDB for the search keyword w is $\hat{\ell}_w = \sum_{i\in\mathcal{I}} \min(X_{w,i}, \lceil \ell_{\max} \cdot 2^i/N \rceil)$, which can be bounded by:

$$\hat{\ell}_w \geq \sum_{i\in\mathcal{I}'} \min(X_{w,i}, \lceil \ell_{\max} \cdot 2^i/N \rceil) \geq \sum_{i\in\mathcal{I}'} (EX_{w,i} - o(EX_{w,i})) = \left(\sum_{i\in\mathcal{I}'} EX_{w,i}\right)(1 - o(1)),$$

[14]This can be seen easily by observing that $\sum_{i\in\mathcal{I}} X_{w,i} = \ell_w = |DB(w)|$.

where $\mathcal{I}' \subseteq \mathcal{I}$ denote the set of indices in eq. (*). The second inequality follows from Proposition 6.3 and the fact that $\forall i \in \mathcal{I}, EX_{w,i} = \ell_w \cdot 2^i/N \leq \lceil \ell_{\max} \cdot 2^i/N \rceil$.

Lastly, note that $\sum_{i\in\mathcal{I}'} EX_{w,i} = \sum_{i\in\mathcal{I}'} \ell_w \cdot 2^i/N = \ell_w/N(N - \sum_{i\in\mathcal{I}\setminus\mathcal{I}'} 2^i) = \ell_w/N(N - o(N)) = \ell_w(1 - o(1))$, which concludes the proof of Theorem 6.2. □

7 Final Construction: Multi-Copy Scheme $\mathsf{VH\text{-}DSSE}^k$

The main limitation of VH-DSSE is that it suffers from noticeable correctness error. We propose a simple-yet-effective solution: run k parallel instances of VH-DSSE with independent randomness. If each individual copy of VH-DSSE returns a constant random fraction of matching documents, we are guaranteed that with high probability, all the matching documents will be returned by at least one of these copies. The security of the resulting multi-copy scheme VH-DSSE in Figure 5 follows directly from the privacy of each single-copy scheme.

Multi-Copy Scheme $\mathsf{VH\text{-}DSSE}^k$

Run k parallel copies of the single-copy scheme VH-DSSE = (VH-DSSE.Setup, VH-DSSE.Search, VH-DSSE.Update) in Figures 2 to 4 with the following modification: the client's output in VH-DSSE.Search is the union of the outputs from the k parallel instances.

Figure 5: $\mathsf{VH\text{-}DSSE}^k$ Protocol

Theorem 7.1. *Suppose that the underlying single-copy scheme is adaptively-secure with respect to the leakage function $\mathcal{L}_{\mathsf{VH\text{-}DSSE}} = (\mathcal{L}^{Stp}_{\mathsf{VH\text{-}DSSE}}, \mathcal{L}^{Srch}_{\mathsf{VH\text{-}DSSE}}, \mathcal{L}^{Updt}_{\mathsf{VH\text{-}DSSE}})$ (Theorem 5.1), forward private, BP-I and BP-I* backward-private (Theorem 5.3). Then, $\mathsf{VH\text{-}DSSE}^k$ is also adaptively-secure with respect to the same leakage function, forward-private, and both BP-I and BP-I* backward-private.*

Proof. The proof follows directly from the security theorems of the single-copy scheme, *i.e.* (Theorem 5.1 and Theorem 5.3). □

Theorem 7.2. $\mathsf{VH\text{-}DSSE}^k$ *achieves the following correctness guarantees, if $k \in \omega(\log \lambda)$:*

1. *$(1, \beta)$-correctness for all keywords;*
2. *$(1, 1)$-correctness for keyword w if its volume $\ell_w \leq \lceil \beta \log |DB| \rceil$;*
3. *$(1 - \mathsf{negl}(\lambda), 1 - \mathsf{negl}(\lambda))$-correctness for keyword w if its volume $\ell_w \in \omega(\log \lambda)$.*

The proof of Theorem 7.2 is given in **??**.

8 Conclusions and Future Works

In this work, we further the research of volume-hiding structured encryption. We present formal definitions for volume-hiding *dynamic* SSE and present the first efficient constructions VH-DSSE and $\mathsf{VH\text{-}DSSE}^k$. Our result fills the gap between existing defensive mechanisms (volume-hiding primitives) and recent volumetric attacks. Meanwhile, both VH-DSSE and $\mathsf{VH\text{-}DSSE}^k$ satisfy the strongest notions of backward-privacy, which are also the first ones in the literature.

One future direction is to efficiently de-amortize $\mathsf{VH\text{-}DSSE}(^k)$ and to propose new volume-hiding DSSE schemes with better efficiency. In addition, designing volume-hiding structured encryption schemes for other dynamic data structures (other than multi-maps) and more expressive queries are also challenging.

Acknowledgement

This research/project is supported by the National Research Foundation, Singapore under its Strategic Capability Research Centres Funding Initiative. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

References

[1] Yuriy Arbitman, Moni Naor, and Gil Segev. De-amortized cuckoo hashing: Provable worst-case performance and experimental results. In *ICALP*, 2009.

[2] Gilad Asharov, Ilan Komargodski, Wei-Kai Lin, Kartik Nayak, Enoch Peserico, and Elaine Shi. Optorama: Optimal oblivious RAM. In *EUROCRYPT*, 2020.

[3] Gilad Asharov, Gil Segev, and Ido Shahaf. Tight tradeoffs in searchable symmetric encryption. In *CRYPTO*, 2018.

[4] Jon Louis Bentley and James B. Saxe. Decomposable searching problems I: static-to-dynamic transformation. *J. Algorithms*, 1(4):301–358, 1980.

[5] Laura Blackstone, Seny Kamara, and Tarik Moataz. Revisiting leakage abuse attacks. In *NDSS*, 2020.

[6] Raphael Bost. $\sum o\varphi o\varsigma$: Forward secure searchable encryption. In *ACM SIGSAC CCS*, 2016.

[7] Raphael Bost, Pierre-Alain Fouque, and David Pointcheval. Verifiable dynamic symmetric searchable encryption: Optimality and forward security. *IACR Cryptol. ePrint Arch.*, 2016:62, 2016.

[8] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. In *ACM SIGSAC CCS*, 2017.

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

[10] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *NDSS*, 2014.

[11] David Cash, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *CRYPTO*, 2013.

[12] Javad Ghareh Chamani, Dimitrios Papadopoulos, Charalampos Papamanthou, and Rasool Jalili. New constructions for forward and backward private symmetric searchable encryption. In *ACM SIGSAC CCS*, 2018.

[13] Yan-Cheng Chang and Michael Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *ACNS*, 2005.

[14] Melissa Chase and Seny Kamara. Structured encryption and controlled disclosure. In *ASIACRYPT*, 2010.

[15] Guoxing Chen, Ten-Hwang Lai, Michael K. Reiter, and Yinqian Zhang. Differentially private access patterns for searchable symmetric encryption. In *IEEE INFOCOM*, 2018.

[16] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *ACM SIGSAC CCS*, 2006.

[17] Ioannis Demertzis, Javad Ghareh Chamani, Dimitrios Papadopoulos, and Charalampos Papamanthou. Dynamic searchable encryption with small client storage. In *NDSS*, 2020.

[18] Ioannis Demertzis, Dimitrios Papadopoulos, and Charalampos Papamanthou. Searchable encryption with optimal locality: Achieving sublogarithmic read efficiency. In *CRYPTO*, 2018.

[19] Ioannis Demertzis, Dimitrios Papadopoulos, Charalampos Papamanthou, and Saurabh Shintre. SEAL: attack mitigation for encrypted databases via adjustable leakage. In *USENIX Security Symposium*, 2020.

[20] Ioannis Demertzis, Stavros Papadopoulos, Odysseas Papapetrou, Antonios Deligiannakis, and Minos N. Garofalakis. Practical private range search revisited. In *SIGMOD*. ACM, 2016.

[21] Mohammad Etemad, Alptekin Küpçü, Charalampos Papamanthou, and David Evans. Efficient dynamic searchable encryption with forward privacy. *Proc. Priv. Enhancing Technol.*, 2018.

[22] Sky Faber, Stanislaw Jarecki, Hugo Krawczyk, Quan Nguyen, Marcel-Catalin Rosu, and Michael Steiner. Rich queries on encrypted data: Beyond exact matches. In *ESORICS*, 2015.

[23] Brett Hemenway Falk, Daniel Noble, and Rafail Ostrovsky. Alibi: A flaw in cuckoo-hashing based hierarchical ORAM schemes and a solution. *IACR Cryptol. ePrint Arch.*, page 997, 2020.

[24] Francesca Falzon, Evangelia Anna Markatou, Akshima, David Cash, Adam Rivkin, Jesse Stern, and Roberto Tamassia. Full database reconstruction in two dimensions. In *ACM CCS*, 2020.

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

[26] Michael T. Goodrich and Michael Mitzenmacher. Privacy-preserving access of outsourced data via oblivious RAM simulation. In *ICALP*, 2011.

[27] Michael T. Goodrich, Michael Mitzenmacher, Olga Ohrimenko, and Roberto Tamassia. Privacy-preserving group data access via stateless oblivious RAM simulation. In *SODA*. SIAM, 2012.

[28] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. Pump up the volume: Practical database reconstruction from volume leakage on range queries. In *ACM SIGSAC CCS*, 2018.

[29] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. Learning to reconstruct: Statistical learning theory and encrypted database attacks. In *IEEE S&P*, 2019.

[30] Paul Grubbs, Kevin Sekniqi, Vincent Bindschaedler, Muhammad Naveed, and Thomas Ristenpart. Leakage-abuse attacks against order-revealing encryption. In *IEEE SP*. IEEE Computer Society, 2017.

[31] Zichen Gui, Oliver Johnson, and Bogdan Warinschi. Encrypted databases: New volume attacks against range queries. In *ACM SIGSAC CCS*, 2019.

[32] Zichen Gui, Kenneth G. Paterson, Sikhar Patranabis, and Bogdan Warinschi. Swissse: System-wide security for searchable symmetric encryption. *IACR Cryptol. ePrint Arch.*, 2020:1328, 2020.

[33] Don Hush and Clint Scovel. Concentration of the hypergeometric distribution. *Statistics & probability letters*, 75(2):127–132, 2005.

[34] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *NDSS*, 2012.

[35] Seny Kamara and Tarik Moataz. Boolean searchable symmetric encryption with worst-case sub-linear complexity. In *EUROCRYPT*, 2017.

[36] Seny Kamara and Tarik Moataz. SQL on structurally-encrypted databases. In *ASIACRYPT*, 2018.

[37] Seny Kamara and Tarik Moataz. Computationally volume-hiding structured encryption. In *EUROCRYPT*, 2019.

[38] Seny Kamara, Tarik Moataz, and Olga Ohrimenko. Structured encryption and leakage suppression. In *CRYPTO*, 2018.

[39] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. Generic attacks on secure outsourced databases. In *ACM SIGSAC CCS*, 2016.

[40] Florian Kerschbaum and Anselme Tueno. An efficiently searchable encrypted data structure for range queries. In *ESORICS*, 2019.

[41] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *ACM SIGSAC CCS*, 2013.

[42] Kee Sung Kim, Minkyu Kim, Dongsoo Lee, Je Hong Park, and Woo-Hwan Kim. Forward secure dynamic searchable symmetric encryption with efficient updates. In *ACM SIGSACCCS*, 2017.

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

[44] Evgenios M. Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. Data recovery on encrypted databases with k-nearest neighbor query leakage. In *IEEE S&P*, 2019.

[45] Evgenios M. Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. The state of the uniform: Attacks on encrypted databases beyond the uniform query distribution. In *IEEE S&P*, 2020.

[46] Evgenios M. Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. Response-hiding encrypted ranges: Revisiting security via parametrized leakage-abuse attacks. *IACR Cryptol. ePrint Arch.*, 2021:93, 2021.

[47] Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. Improved reconstruction attacks on encrypted data using range query leakage. In *IEEE S&P*, 2018.

[48] Russell W. F. Lai and Sherman S. M. Chow. Forward-secure searchable encryption on labeled bipartite graphs. In *ACNS*, 2017.

[49] Jinfei Liu, Juncheng Yang, Li Xiong, and Jian Pei. Secure skyline queries on cloud platform. In *IEEE ICDE*, 2017.

[50] Xianrui Meng, Seny Kamara, Kobbi Nissim, and George Kollios. GRECS: graph encryption for approximate shortest distance queries. In *ACM SIGSAC CCS*, 2015.

[51] Muhammad Naveed, Seny Kamara, and Charles V. Wright. Inference attacks on property-preserving encrypted databases. In *ACM SIGSAC CCS*. ACM, 2015.

[52] Simon Oya and Florian Kerschbaum. Hiding the access pattern is not enough: Exploiting search pattern leakage in searchable encryption. *CoRR*, abs/2010.03465, 2020.

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

[54] Sarvar Patel, Giuseppe Persiano, Mariana Raykova, and Kevin Yeo. Panorama: Oblivious RAM with logarithmic overhead. In *IEEE FOCS*, 2018.

[55] Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing. In *ACM SIGSAC CCS*, 2019.

[56] Sikhar Patranabis and Debdeep Mukhopadhyay. Forward and backward private conjunctive searchable symmetric encryption. In *NDSS*. The Internet Society, 2021.

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

[58] Panagiotis Rizomiliotis, Eirini Molla, and Stefanos Gritzalis. REX: A searchable symmetric encryption scheme supporting range queries. In *CCSWCCS*. ACM, 2017.

[59] Zhiwei Shang, Simon Oya, Andreas Peter, and Florian Kerschbaum. Obfuscated access and search patterns in searchable encryption. *CoRR*, abs/2102.09651, 2021.

[60] Elaine Shi, Emil Stefanov, and Charalampos Papamanthou. Practical dynamic proofs of retrievability. In *ACM CCS*, 2013.

[61] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. Practical dynamic searchable encryption with small leakage. In *NDSS*. The Internet Society, 2014.

[62] Shi-Feng Sun, Ron Steinfeld, Shangqi Lai, Xingliang Yuan, Amin Sakzad, Joseph K. Liu, Surya Nepal, and Dawu Gu. Practical non-interactive searchable encryption with forward and backward privacy. In *NDSS*. The Internet Society, 2021.

[63] Jiafan Wang, Minxin Du, and Sherman S. M. Chow. Stargazing in the dark: Secure skyline queries with SGX. In *DASFAA*, 2020.

[64] Stephanie Wang, Rishabh Poddar, Jianan Lu, and Raluca Ada Popa. Practical volume-based attacks on encrypted databases. *IACR Cryptol. ePrint Arch.*, 2019:1224, 2019.

[65] Weiguo Wang, Hui Li, Yanguo Peng, Sourav S. Bhowmick, Peng Chen, Xiaofeng Chen, and Jiangtao Cui. SCALE: an efficient framework for secure dynamic skyline query processing in the cloud. In *DASFAA*, 2020.

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

[67] Cong Zuo, Shifeng Sun, Joseph K. Liu, Jun Shao, and Josef Pieprzyk. Dynamic searchable symmetric encryption with forward and stronger backward privacy. *IACR Cryptol. ePrint Arch.*, page 1055, 2019.

[68] Cong Zuo, Shifeng Sun, Joseph K. Liu, Jun Shao, and Josef Pieprzyk. Dynamic searchable symmetric encryption with forward and stronger backward privacy. In *ESORICS*, 2019.

A (Delegatable) Pseudorandom Function and RoR-Secure Symmetric-key Encryption Scheme

Let $\mathsf{KeyGen}(1^\lambda) \in \{0,1\}^\lambda$ be a key generation function, and $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^\ell \to \{0,1\}^{\ell'}$ be a *pseudorandom function* (PRF) family. PRF is a *secure* PRF family if for all PPT adversaries $\mathcal{A}$, the following holds:

$$|\Pr[K \leftarrow \mathsf{KeyGen}(1^\lambda); \mathcal{A}^{\mathsf{PRF}(K,\cdot)}(1^\lambda) = 1] - \Pr[\mathcal{A}^{R(\cdot)}(1^\lambda) = 1]| \leq \mathsf{negl}(\lambda),$$

where $R(\cdot)$ denotes a truly random function.

A family of *delegatable* PRF enables the owner of the secret key K to delegate an untrusted party to compute $\mathsf{PRF}(K,x)$ for all values $x \in S$ where S is a subset of the input space $\mathcal{X}$. The delegation is done by deriving a token token_S by $\mathsf{PRF.Delegate}(K,S)$. With token_S one can compute $\mathsf{PRF.Eval}(\mathsf{token}_S, x) = \mathsf{PRF}(K,x)$ for all $x \in S$, without accessing K. The security requirement is that given token_S, all values $\mathsf{PRF}(K,x), x \notin S$ remains indistinguishable from truly random ones. We consider delegatable PRF for (fixed-size) subsets of strings with matching prefixes. Let prefix denote a string and ℓ be a positive integer. The token generation algorithm outputs $\mathsf{token}_{\mathsf{prefix},\ell} = \mathsf{PRF.Delegate}(K, (\mathsf{prefix}, \ell))$ with which one can compute $\mathsf{PRF}(K,x) = \mathsf{PRF.Eval}(\mathsf{token}_{\mathsf{prefix}}, i)$ for all $x = \mathsf{prefix}||i||0$ and $x = \mathsf{prefix}||i||1$ where $i \in [1,\ell]$ that share the same common prefix. The classical GGM construction [25] of PRF can be used to implement such delegatable PRFs [41]. It is also easy to see that when ℓ is fixed, the resulting tokens for fixed-length prefixes are computationally indistinguishable from random values.

A symmetric-key encryption scheme SKE consists of three algorithms (KeyGen, Enc, Dec). Let $\mathsf{KeyGen}(1^\lambda) \in \{0,1\}^\lambda$ is a probabilistic key generation function that takes a security parameter λ as input and returns a secret key K; Enc is a probabilistic algorithm that takes as input a key K and a message m and generates a ciphertext ct; Dec is a deterministic algorithm that takes as input a key K and a ciphertext ct and recovers m if K was used to encrypt m into ct. For notation simplicity, we additionally introduce a helper function DecAll that decrypts a set of ciphertexts. We require SKE to satisfy real-or-random security (RoR). Informally, a symmetric-key encryption scheme is RoR-secure if the ciphertexts are indistinguishable from random strings of the same length, even to an adversary that can adaptively query an encryption oracle.

B Probability Distributions

B.1 Hypergeometric Distribution

A hypergeometric distribution describes the process of sampling without replacement. A random variable X following the hypergeometric distribution $H(n_1, n, m)$ describes the process of counting how many defectives are sampled when n_1 items are randomly selected without replacement from a population of n items of which m are defective. The probability mass functions of X is $\Pr[X = k] = \frac{\binom{n-m}{n_1-k}\binom{m}{k}}{\binom{n}{n_1}}$. The expected value $E[X]$ is $n_1 \frac{m}{n}$.

Lemma B.1 (Tail Bound of Hypergeometric Distribution [33])**.** *Let X be a random variable following the hypergeometric distribution $H(n_1, n, m)$. Let $\gamma \geq 2$. Then*

$$\Pr[X - E[X] > \gamma] < e^{-2(\gamma^2-1)\alpha_{n_1,n,m}} \tag{1}$$

and

$$\Pr[X - E[X] < -\gamma] < e^{-2(\gamma^2-1)\alpha_{n_1,n,m}} \tag{2}$$

where

$$\alpha_{n_1,n,m} = \max\left(\left(\frac{1}{n_1+1} + \frac{1}{n-n_1+1}\right), \left(\frac{1}{m+1} + \frac{1}{n-m+1}\right)\right).$$

The proof of this Lemma can be found in [33, Theorem 1].

B.2 Zipf-Distribution

Originally, Zipf's law is an empirical law that models the rank-frequency distribution of words in human languages such as English. It is observed that similar relationships also occur in other rankings of human-created systems. Following [37], we present its definition in the context of key-value database below.

Definition B.1 (Zipf-Distributed Database)**.** We say that the response lengths of a keyword-document database DB is $\mathcal{Z}_{a,b}$-distrubted, if the normalized frequency of the r-th most frequent keyword is $\frac{r^{-b}}{H_{a,b}}$, where $N = \sum |DB(w)|_{w\in\mathcal{W}}$ and $H_{a,b}$ is the harmonic number $\sum_{i=1}^{a} i^{-b}$.

For instance, if we assume that DB is $\mathcal{Z}_{|\mathcal{W}|,1}$-distributed, then the normalized frequency of the i-th most frequent keyword is $\frac{1/i}{H_{|\mathcal{W}|,1}} = \frac{1/i}{(1+\frac{1}{2}+\cdots+\frac{1}{|\mathcal{W}|})}$ where $|\mathcal{W}|$ is the size of the keyword space.

C Modified Static Volume-hiding SSE scheme dprfMM [55]

Let $\mathsf{SKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be an RoR-secure symmetric encryption scheme and PRF be a family of secure delegatable pseudorandom function. This construction is parameterized by a constant $\alpha > 0$ and a function $g(n) \in \omega(1)$.

- $(K, \tau, EDB) \leftarrow \mathsf{dprfMM.Setup}(1^\lambda, DB)$:
  1. Randomly select a PRF key $K_{\mathsf{PRF}} \leftarrow \mathsf{PRF.KeyGen}(1^\lambda)$;
  2. Randomly select an encryption key $K_{\mathsf{SKE}} \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$;
  3. Create two empty arrays T_0, T_1 of length $t = (1+\alpha)n$ where $n = |DB|$;
  4. Initialize $\mathsf{stash} \leftarrow \phi$;
  5. For each $w \in \mathcal{W}$ and each $i \in [1, |DB(w)|]$:
     (a) Let (w, ID) be the i-th entry in $DB(w)$, namely $DB(w)[i] = (w, \mathsf{ID})$;
     (b) Insert (w, ID) using the Cuckooo hashing with a stash insertion algorithm where (w, ID) is assigned to one of $T_0[\mathsf{PRF}(K_{\mathsf{PRF}}, (w||i||0))]$, $T_1[\mathsf{PRF}(K_{\mathsf{PRF}}, (w||i||1))]$ or stash; ~~If stash contains more than $f(n) \in \omega(1)$ items, abort.~~
  6. For each empty location in T_0 and T_1, insert $(\perp, \perp)$;
  7. For each $i \in [t], b \in \{0,1\}$:
     – Set $T_b[i] \leftarrow \mathsf{SKE.Enc}(K_{\mathsf{SKE}}, T_b[i])$;
  8. Set the private key as $K \leftarrow (K_{\mathsf{PRF}}, K_{\mathsf{SKE}}, \mathsf{stash})$;
  9. Set $EDB \leftarrow (T_0, T_1)$, set $\tau = \phi$;
  10. Return (K, τ, EDB).

- $\mathsf{dprfMM.Search}(K, q, \tau, \ell; EDB, \ell)$:
  Client:
  1. Parse K as $(K_{\mathsf{PRF}}, K_{\mathsf{SKE}}, \mathsf{stash})$;
  2. Compute $\mathsf{token}_{q,\ell} \leftarrow \mathsf{PRF.Delegate}(K_{\mathsf{PRF}}, (q, \ell))$;
  3. Send $\mathsf{token}_{q,\ell}$ to the server.

  Server:
  1. For each $i \in [1, \ell], b \in \{0,1\}$:
     – Compute $a_{b,i} = \mathsf{PRF}(K_{\mathsf{PRF}}, (q||i||b)) = \mathsf{PRF.Eval}(\mathsf{token}_{q,\ell}, (i||b))$;
  2. Send $\mathsf{response} = \{T_0[a_{0,i}], T_1[a_{1,i}]\}_{i\in[1,\ell]}$ to the client.

  Client:
  1. Parse K as $(K_{\mathsf{PRF}}, K_{\mathsf{SKE}}, \mathsf{stash})$, parse $\mathsf{response}$ as $\{ct_{0,i}, ct_{1,i}\}_{i\in[1,\ell]}$;
  2. Set $\mathsf{result} = \phi$;
  3. For each $i \in [1, \ell], b \in \{0,1\}$:
     (a) Set $(w', \mathsf{ID}') \leftarrow \mathsf{SKE.Dec}(K_{\mathsf{SKE}}, ct_{b,i})$;
     (b) If $w' = q$, then $\mathsf{result} = \mathsf{result} \cup \mathsf{ID}'$;
  4. For each $(w', \mathsf{ID}') \in \mathsf{stash}$:
     – If $w' = q$, $\mathsf{result} = \mathsf{result} \cup \mathsf{ID}'$;
  5. Return result.

As discussed in Section 4.3, we make two modifications to the original dprfMM scheme in [55]. First, the setup algorithm does not abort even if the size of stash is $\omega(1)$. Second, we make the response length as an explicit input of the search protocol. These differences are highlighted in blue above.

The leakage profile of dprfMM presented above is $\mathcal{L}'_{\mathsf{dprfMM}} = (\mathcal{L}^{Stp}_{\mathsf{dprfMM}}, \mathcal{L}^{Srch}_{\mathsf{dprfMM}}) = (\mathcal{L}^{DSize}, \mathcal{L}^{Eq})$. In contrast, the leakage profile in the original paper [55] further includes $\mathcal{L}^{MRL}$ because the response length is fixed to be $\ell_{\max} = \max_{q \in \mathcal{W}} |DB(q)|$ in the original scheme. Because the interface of the search protocol is changed to take the response length as an explicit input, the $\mathsf{Real}^{DSSE}_{\mathcal{A}}$ and $\mathsf{Ideal}^{DSSE}_{\mathsf{Sim},\mathcal{L}}$ game need to be modified accordingly. The modifications are highlighted in Figure 1. To be specific, the adversary $\mathcal{A}$ additionally chooses a length parameter ℓ_k when it adaptively chooses search queries q_k in Line 3, Figure 1. This length parameter ℓ_k is also an explicit input to $\mathsf{SimSearch}$. With such modifications, $\mathsf{SimSearch}$ is required to simulate the real search protocol for *any* response length because ℓ_k is adaptively chosen by $\mathcal{A}$. We stress that ℓ_k should not be considered as part of $\mathcal{L}^{Srch}_{\mathsf{dprfMM}}$ because it is directly chosen by the adversary in the security games. That said, we do not claim that the modified dprfMM scheme remains volume-hiding as per the original definition in [55], since the input interface has changed.

D Volume-Hiding Leakage Functions for Static SSE [55]

We want to quantitatively compare Definition 3.1 with the one for static databases in [55]. For this purpose, we present the definition of volume-hiding leakage functions in [55] below. We slightly rephrased their syntax for easier comparison.

Definition D.1 ([55]). The *signature* of a (single-keyword) encrypted database is the sequence of pairs $(w, \mathsf{len}(DB, w))_{w \in \mathcal{W}}$ where $\mathsf{len}(DB, w) = |DB(w)|$ is the number of matching documents in DB associated with the keyword w.

Now we are ready to define the $\mathsf{sVolH}^{\eta}_{\mathcal{A},\mathcal{L}}(1^\lambda, N, \ell_{\max})$ game for leakage functions $\mathcal{L} = (\mathcal{L}^{Stp}, \mathcal{L}^{Srch})$, adversary $\mathcal{A}$, and $\eta \in \{0, 1\}$, which proceeds in the following steps:

- Setup phase: $\mathcal{A}$ generates two pairs of keyword-volume pairs $S_0 = \{(w, \ell_{(0,w)})\}_{w \in \mathcal{W}}$ and $S_1 = \{(w, \ell_{(1,w)})\}_{w \in \mathcal{W}}$ with the following constraints:
  - $\sum_{w \in \mathcal{W}} \ell_{(0,w)} = \sum_{w \in \mathcal{W}} \ell_{(1,w)} = N$;
  - $\max_{w \in \mathcal{W}} \ell_{(0,w)} = \max_{w \in \mathcal{W}} \ell_{(1,w)} = \ell_{\max}$.

  The challenger C receives S_0, S_1 from the adversary $\mathcal{A}$ and generates a database DB with the signature S_η. Specifically, the challenger C generates $\mathsf{len}(DB, w) = \ell_{(\eta,w)}$ arbitrary values for each keyword $w \in \mathcal{W}$. The challenger C then sends $\mathcal{L}^{Stp}(DB)$ to the adversary.

- Query phase: The challenger C and the adversary $\mathcal{A}$ repeat the following steps for $\mathsf{poly}(\lambda)$ times:
  - $\mathcal{A}$ adaptively chooses keywords $w_1, \ldots, w_t$ for search operations. For each w_i, the challenger will compute the (stateful) leakage function $\mathcal{L}^{Srch}(DB, w_i)$ which is returned to the adversary $\mathcal{A}$.

- Output phase: $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$.

We denote by $p^{\mathcal{A},\mathcal{L}}_{\eta}(N, \ell_{\max})$ the probability that $\mathcal{A}$ outputs 1 when playing the game $\mathsf{sVolH}^{\eta}_{\mathcal{A},\mathcal{L}}(1^\lambda, N, \ell_{\max})$.

The advantage of $\mathcal{A}$ in the game $\mathsf{sVolH}_{\mathcal{A},\mathcal{L}}$ is defined as $\mathsf{Adv}^{\mathsf{sVolH}}_{\mathcal{A},\mathcal{L}} = |p^{\mathcal{A},\mathcal{L}}_0(N, \ell_{\max}) - p^{\mathcal{A},\mathcal{L}}_1(N, \ell_{\max})|$.

Definition D.2 (Volume-Hiding Leakage Functions for Static SSE [55]). A leakage function $\mathcal{L} = (\mathcal{L}^{Stp}, \mathcal{L}^{Srch})$ is volume-hiding if the advantage of the adversary in the above game is 0.

Now, we are ready to formally prove the statement given in Section 3.3.

Theorem D.1. *Let Σ be a DSSE scheme with leakage profile $\mathcal{L} = (\mathcal{L}^{Stp}, \mathcal{L}^{Srch}, \mathcal{L}^{Updt})$ and $\mathcal{A}$ be an adversary. Suppose the advantage of $\mathcal{A}$ in the $\mathsf{sVolH}_{\mathcal{A},\mathcal{L}'}$ game is p, where $\mathcal{L}' = (\mathcal{L}^{Stp}, \mathcal{L}^{Srch})$. There must exist an adversary $\mathcal{B}$ whose advantage in the $\mathsf{VolH}_{\mathcal{A},\mathcal{L}}$ game is at least p.*

Proof. We describe how to construct the adversary $\mathcal{B}$ from $\mathcal{A}$. $\mathcal{B}$ invokes $\mathcal{A}$ and plays the role of the challenger in the sVolH game. It simulates the three phases of the sVolH game as follows:

- Setup Phase: $\mathcal{B}$ receives two sets of keyword-volume pairs S_0, S_1 from $\mathcal{A}$ and generates two databases DB_0 and DB_1 with the signatures S_0 and S_1, respectively. Specifically, the challenger C generates $\mathsf{len}(DB, w) = \ell_{(\eta,w)}$ arbitrary values for each keyword $w \in \mathcal{W}$. $\mathcal{B}$ forwards DB_0, DB_1 to its own challenger, who randomly selects a bit b and returns $\mathcal{L}^{Stp}(DB_b)$ to $\mathcal{B}$. $\mathcal{B}$ directly forwards $\mathcal{L}^{Stp}(DB_b)$ to $\mathcal{A}$.
- Query Phase: $\mathcal{B}$ receives search query w_i from $\mathcal{A}$. $\mathcal{B}$ constructs its own query $((\mathsf{type}_0 = \mathsf{Search}, w_i), (\mathsf{type}_1 = \mathsf{Search}, w_i))$ and forwards it to its own challenger. $\mathcal{B}$ forwards the received search leakage $\mathcal{L}^{Srch}(DB_b, w_i)$ back to $\mathcal{A}$.
- Output Phase: $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

It is easy to verify that $\mathcal{B}$ has simulated the sVolH game for $\mathcal{A}$ perfectly. Now it remains to analyze the advantage of $\mathcal{B}$ in the VolH game. According to our assumption, we have $\mathsf{Adv}^{\mathsf{sVolH}}_{\mathcal{A},\mathcal{L}'} = |p_0^{\mathcal{A},\mathcal{L}'} - p_1^{\mathcal{A},\mathcal{L}'}| = p$.

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{VolH}}_{\mathcal{A},\mathcal{L}} &= |2\Pr[b' = b] - 1| \\
&= |2\Pr[b' = 1|b = 1] \cdot \Pr[b = 1] + 2\Pr[b' = 0|b = 0] \cdot \Pr[b = 0] - 1| \\
&= |\Pr[b' = 1|b = 1] + \Pr[b' = 0|b = 0] - 1| \\
&= |\Pr[b' = 1|b = 1] + (1 - \Pr[b' = 1|b = 0]) - 1| \\
&= |p_1^{\mathcal{A},\mathcal{L}'} - p_0^{\mathcal{A},\mathcal{L}'}| = p
\end{aligned}$$

□

E Missing Proof of Theorem 5.1.

sketch. Based on our assumption, there exists a simulator $\mathsf{Sim}_{\mathsf{dprfMM}} = (\mathsf{SimSetup}_{\mathsf{dprfMM}}, \mathsf{SimSearch}_{\mathsf{dprfMM}})$. We construct $\mathsf{Sim} = (\mathsf{SimSetup}, \mathsf{SimSearch}, \mathsf{SimUpdate})$ using $\mathsf{Sim}_{\mathsf{dprfMM}}$ and then show that for any PPT adversary $\mathcal{A}$, the random variables $\mathsf{Real}^{\mathsf{VH\text{-}DSSE}}_{\mathcal{A}}$ and $\mathsf{Ideal}^{\mathsf{VH\text{-}DSSE}}_{\mathsf{Sim},\mathcal{L}_{\mathsf{VH\text{-}DSSE}}}$ are negligibly-distinguishable.

- $\mathsf{SimSetup}(\mathcal{L}^{Stp}_{\mathsf{VH\text{-}DSSE}}(DB_0))$

```
1 :  Parse $\mathcal{L}^{Stp}_{\mathsf{VH\text{-}DSSE}}$ as $(N, \beta)$
2 :  Set $\mathsf{ctr} = N$, Set $EDB$ as an empty set of vectors
3 :  Let $b_{\lfloor \log N \rfloor} \cdots b_0$ be the binary representation of $N$
4 :  Let $\mathsf{min}$ be the integer such that $2^{\mathsf{min}} \leq \log N < 2^{\mathsf{min}+1}$
5 :  // Run a fresh instance of $\mathsf{Sim}_{\mathsf{dprfMM}} = (\mathsf{SimSetup}_{\mathsf{dprfMM}}, \mathsf{SimSearch}_{\mathsf{dprfMM}})$ for each $b_i = 1$
6 :  for $i = \lfloor \log N \rfloor$ to $\mathsf{min}$ do :
7 :    if $b_i = 1$ then
8 :      $(st_i, \hat{EDB}_i) \leftarrow \mathsf{SimSetup}^{(i)}_{\mathsf{dprfMM}}(1^{\lambda}, 2^i)$
9 :      $\mathsf{ctr} = \mathsf{ctr} - 2^i$
10 : Fill $\mathsf{stash}$ with $f(N)$ random strings
11 : Fill $\mathsf{buf}$ with $\mathsf{ctr}$ random strings
12 : Set $EDB = (\beta, \mathsf{buf}, \mathsf{stash}, (\hat{EDB}_{\mathsf{min}}, \ldots, \hat{EDB}_{\lfloor \log N \rfloor}))$
13 : Set $st_{\mathcal{S}} = (st_{\mathsf{min}}, \ldots, st_{\lfloor \log N \rfloor}, N, \beta, \mathsf{buf}, \mathsf{stash})$
14 : return $(st_{\mathcal{S}}, EDB)$
```

- $\mathsf{SimUpdate}(st_{\mathcal{S}}, \mathcal{L}^{Updt}_{\mathsf{VH\text{-}DSSE}}(EDB_{k-1}, w_k))$

```
1 :  $ct^* \leftarrow \{0,1\}^{|ct|}$, extract $N$ from $st_{\mathcal{S}}$, $N = N + 1$
2 :  Compute $size_{\mathsf{buf}}$, $\mathsf{min}$ from $N$ as in the real algorithm
3 :  if $size_{\mathsf{buf}} = 2^{\mathsf{min}+1}$
4 :    Identify the first index $i$ that does not have a running instance of $\mathsf{Sim}_{\mathsf{dprfMM}}$
5 :    for $j = \mathsf{min}$ to $i - 1$ do
6 :      Stop the simulator $\mathsf{Sim}^{(j)}_{\mathsf{dprfMM}}$ for the $j$-th instance
7 :    $(st_i, \hat{EDB}_i) \leftarrow \mathsf{SimSetup}^{(i)}_{\mathsf{dprfMM}}(1^{\lambda}, 2^i)$
8 :    Replace $\mathsf{stash}$ with $f(N)$ fresh random strings, set $\mathsf{buf} \leftarrow \phi$
9 :    $t^* = (ct^*, \hat{EDB}_i)$
10 : else
11 :   $t^* = ct^*$
12 : Update $N, \mathsf{buf}, \mathsf{stash}, st_{\mathsf{min}}, \ldots, st_i$ in $st_{\mathcal{S}}$ and return $t^*$
```

Before we present the construction of $\mathsf{SimSearch}$, we first establish the following notations. Let $q_k = w_k$ be the latest search query and $Q_k = \{q_1, \ldots, q_k\}$ be the sequence of *all search queries* issued so far. Q_k can be partitioned into $\{\ldots, Q^{(j-1)}, Q^{(j)}\}$, where $j = \lfloor \log N \rfloor$ and $Q^{(i)}$ denote the sequence of search queries when the largest non-empty encrypted index in EDB was i. Therefore, q_k is the last query in $Q^{(j)}$. Let M_j be a $|Q^{(j)}| \times |Q^{(j)}|$ binary matrix such that $M_j[x][y] = 1$ if and only if $q_{x-|Q^{(j)}|+k} = q_{y-|Q^{(j)}|+k}$. Looking ahead, $\mathsf{SimSearch}$ will use M_j, which is part of $\mathcal{L}^{Srch}_{\mathsf{VH\text{-}DSSE}}$, to simulate the search leakage of q_k.

- $\mathsf{SimSearch}(st_{\mathcal{S}}, \mathcal{L}^{Srch}_{\mathsf{VH\text{-}DSSE}}(EDB_{k-1}, w_k))$

```
1 : Extract $N, \beta$ from $st_{\mathcal{S}}$, set $t^* = \phi$
2 : Set $j = \lfloor \log N \rfloor$; compute $\mathsf{min}$ from $N$ as in the real algorithm
3 : Let $b_j \cdots b_0$ be the binary representation of $N$
4 : Extract $M_j$ from $\mathcal{L}^{Srch}_{\mathsf{VH\text{-}DSSE}}(EDB_{k-1}, w_k)$
5 : for $i = j$ to min do :
6 :   if $b_i = 1$   // There must be an active running instance of $\mathsf{Sim}^{(i)}_{\mathsf{dprfMM}}$
7 :     Extract (sub)search pattern $M^{(i)}_j$ in the lifespan of $\mathsf{Sim}^{(i)}_{\mathsf{dprfMM}}$ from $M_j$
8 :     Append $\mathsf{SimSearch}^{(i)}_{\mathsf{dprfMM}}(st_i, M^{(i)}_j)$ to $t^*$ and update $st_i$
9 : return $t^*$
```

We now argue for all PPT adversary $\mathcal{A}$,

$$|\Pr[\mathsf{Real}^{DSSE}_{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathsf{Ideal}^{DSSE}_{\mathsf{Sim},\mathcal{L}}(1^\lambda)]| \leq \mathsf{negl}(\lambda).$$

To do so, we use the following sequence of games:

- **Game**$_0$ is identical to $\mathsf{Real}^{DSSE}_{\mathcal{A}}(1^\lambda)$.
- **Game**$_1$ replaces the RoR-secure SKE encryption scheme with a random function.
- **Game**$_2$ replaces all dprfMM algorithms with $\mathsf{Sim}_{\mathsf{dprfMM}}$.

Game$_0$ and **Game**$_1$ are computationally indistinguishable due to the RoR-security of SKE. **Game**$_1$ and **Game**$_2$ are computationally indistinguishable due to the security of dprfMM. It is easy to verify that **Game**$_2$ is the same as $\mathsf{Ideal}^{DSSE}_{\mathsf{Sim},\mathcal{L}}(1^\lambda)$. □

F Missing Proof of Proposition 6.3.

Before we proceed, we first give a concrete expression of δ_j as follows:

$$\delta_j = \begin{cases} \gamma_{i_{m_1}} & \text{if } j = 1 \\ \sum_{k=1}^{j-1} \delta_k + \gamma_{i_{m_j}} & \text{if } j \geq 2, \end{cases}$$

where $\gamma_{i_{m_j}} = \sqrt{EX_{w,i_{m_j}} \cdot \mathsf{polylog}(\lambda)} = \sqrt{(\ell_w \cdot 2^{i_{m_j}}/N) \cdot \mathsf{polylog}(\lambda)} \in o(EX_{w,i_{m_j}})$. We can upper bound the value of δ_j as follows, which will be used in our proof.

Claim F.1. *For $j \geq 1$, we have $\delta_j \leq 2^{j-1} \cdot \delta_1$.*

Proof. Note that $\forall j, \gamma_{i_{m_j}} > \gamma_{i_{m_{j+1}}}$. For $j = 1, 2$, it is easy to see $\delta_1 \leq 2^0\delta_1$, $\delta_2 = \delta_1 + \gamma_{i_{m_2}} < 2^1\delta_1$. For $j > 2$, we have $\delta_j = \sum_{k=1}^{j-1} \delta_k + \gamma_{i_{m_j}} = \delta_{j-1} + \sum_{k=1}^{j-2} \delta_k + \gamma_{i_{m_j}} = \delta_{j-1} + (\delta_{j-1} - \gamma_{i_{m_{j-1}}}) + \gamma_{i_{m_j}} < 2\delta_{j-1} < 2^{j-1} \cdot \delta_1$. □

Now we are ready to prove Proposition 6.3.

Proof. **Base Case:** The base case, $\Pr[\hat{E}_{w,i_{m_1}}] > 1 - \mathsf{negl}(\lambda)$, concerns the largest index i_{m_1} in $\mathcal{I}$. By plugging $\gamma_{i_{m_1}} = \sqrt{(\ell_w \cdot 2^{i_{m_1}}/N) \cdot \mathsf{polylog}(\lambda)} \in o(EX_{w,i_{m_1}})$, $n_1 = 2^{i_{m_1}}$, $n = N$, $m = \ell_w$ into Equations (1) and (2) in Lemma B.1, we get

$$\Pr[X_{w,i_{m_1}} - EX_{w,i_{m_1}} \in [-\gamma_{i_{m_1}}, \gamma_{i_{m_1}}]] \geq 1 - 2e^{-2\alpha_{2^{i_{m_1}},N,\ell_w}((\ell_w \cdot 2^{i_{m_1}}/N)\cdot\mathsf{polylog}(\lambda) - 1)}$$

Notice that $\alpha_{2^{i_{m_1}},N,\ell_w} = \max\left(\left(\frac{1}{2^{i_{m_1}}+1} + \frac{1}{N-2^{i_{m_1}}+1}\right), \left(\frac{1}{\ell_w+1} + \frac{1}{N-\ell_w+1}\right)\right) > \frac{1}{\ell_w+1}$ and $N/2 < 2^{i_{m_1}} \leq N$. Therefore,

$$\begin{aligned}
\Pr[X_{w,i_{m_1}} - EX_{w,i_{m_1}} \in [-\gamma_{i_{m_1}}, \gamma_{i_{m_1}}]] &\geq 1 - 2e^{-2\alpha_{2^{i_{m_1}},N,\ell_w}((\ell_w \cdot 2^{i_{m_1}}/N)\cdot \mathsf{polylog}(\lambda)-1)} \\
\Pr[\hat{E}_{w,i_{m_1}}] &> 1 - 2e^{-2\frac{1}{\ell_w+1}\cdot((\ell_w \cdot 2^{i_{m_1}}/N)\cdot \mathsf{polylog}(\lambda)-1)} \\
&= 1 - 2e^{-O(\mathsf{polylog}(\lambda))} \\
&= 1 - \mathsf{negl}(\lambda) \qquad (**)
\end{aligned}$$

With Equation (**), we have concluded the proof of the base case.

Inductive Step: We want to show that if $\Pr[\bigwedge_{j=1}^{k} \hat{E}_{w,i_{m_j}}] > 1 - \mathsf{negl}(\lambda)$ and $N - \sum_{j=1}^{k} 2^{i_{m_j}} \in O(N)$ for some $k \geq 1$, then $\Pr[\bigwedge_{j=1}^{k+1} \hat{E}_{w,i_{m_j}}] > 1 - \mathsf{negl}(\lambda)$. The base case above corresponds to $k = 1$.

Let $N^{(i_{m_{k+1}})} = N - \sum_{j=1}^{k} 2^{i_{m_j}}$ denote the total number of key-value pairs remaining in $\{E\hat{D}B_{i_{m_{k+1}}}, E\hat{D}B_{i_{m_{k+2}}}, \ldots\}$. Let $\ell_w^{(i_{m_{k+1}})} = \ell_w - \sum_{j=1}^{k} X_{w,i_{m_j}}$ denote the total number of matching results for w remaining in $\{E\hat{D}B_{i_{m_{k+1}}}, E\hat{D}B_{i_{m_{k+2}}}, \ldots\}$. According to our inductive hypothesis, $N^{(i_{m_{k+1}})} \in O(N)$.

The conditional random variable $X_{w,i_{m_{k+1}}} | \bigwedge_{j=1}^{k} \hat{E}_{w,i_{m_j}}$ follows hypergeometric distribution $H(2^{i_{m_{k+1}}}, N^{(i_{m_{k+1}})}, \ell_w^{(i_{m_{k+1}})})$ and $E[X_{w,i_{m_{k+1}}} | \bigwedge_{j=1}^{k} \hat{E}_{w,i_{m_j}}] = \frac{\ell_w^{(i_{m_{k+1}})} \cdot 2^{i_{m_{k+1}}}}{N^{(i_{m_{k+1}})}}$. Hence, we can apply Lemma B.1 again:

$$\begin{aligned}
&\Pr\left[X_{w,i_{m_{k+1}}} - E[X_{w,i_{m_{k+1}}} | \textstyle\bigwedge_{j=1}^{k} \hat{E}_{w,i_{m_j}}] \in [-\gamma_{i_{m_{k+1}}}, \gamma_{i_{m_{k+1}}}] | \bigwedge_{j=1}^{k} \hat{E}_{w,i_{m_j}}\right] \\
\geq& 1 - 2e^{-2\alpha_{2^{i_{m_{k+1}}}, N^{(i_{m_{k+1}})}, \ell_w^{(i_{m_{k+1}})}} (\gamma^2_{i_{m_{k+1}}} - 1)} \\
>& 1 - 2e^{-2\frac{1}{\ell_w^{(i_{m_{k+1}})}+1} \cdot (\gamma^2_{i_{m_{k+1}}} - 1)} \qquad (***)
\end{aligned}$$

Conditioned on the event $\bigwedge_{j=1}^{k} \hat{E}_{w,i_{m_j}}$, then $\ell_w^{(i_{m_{k+1}})}$ falls in range $[\ell_w - \sum_{j=1}^{k}(EX_{w,i_{m_j}} + \delta_j), \ell_w - \sum_{j=1}^{k}(EX_{w,i_{m_j}} - \delta_j)]$, namely:

$$\ell_w - \sum_{j=1}^{k}(EX_{w,i_{m_j}} + \delta_j) \leq \ell_w^{(i_{m_{k+1}})} \leq \ell_w - \sum_{j=1}^{k}(EX_{w,i_{m_j}} - \delta_j)$$

$$\ell_w \left(\frac{N - \sum_{j=1}^{k} 2^{i_{m_j}}}{N}\right) - \sum_{j=1}^{k} \delta_j \leq \ell_w^{(i_{m_{k+1}})} \leq \ell_w \left(\frac{N - \sum_{j=1}^{k} 2^{i_{m_j}}}{N}\right) + \sum_{j=1}^{k} \delta_j \qquad (****)$$

According to our hypothesis that $N - \sum_{j=1}^{k} 2^{i_{m_j}} \in O(N)$, there exists a constant c such that $N^{(i_{m_{k+1}})} = N - \sum_{j=1}^{k} 2^{i_{m_j}} = cN$. The inequality above simplifies to $c\ell_w - \sum_{j=1}^{k} \delta_j \leq \ell_w^{(i_{m_{k+1}})} \leq c\ell_w + \sum_{j=1}^{k} \delta_j$. Given the upper bound of δ_j that we have proven earlier, we have $\sum_{j=1}^{k} \delta_j \leq \sum_{j=1}^{k}(2^{j-1} \cdot \delta_1) = (2^k - 1)\delta_1$. Observe that $k \leq |\mathcal{I}| \in o(\log N)$ and our assumption that $\ell_w \in O(N)$. So $2^k = 2^{o(\log N)} = N^{o(1)}$ and $\sum_{j=1}^{k} \delta_j < N^{o(1)} \cdot \delta_1 \in o(\ell_w)$. Hence $\ell_w^{(i_{m_{k+1}})} \in O(\ell_w)$.

Recall $\gamma_{i_{m_{k+1}}} = \sqrt{(\ell_w \cdot 2^{i_{m_{k+1}}}/N) \cdot \mathsf{polylog}(\lambda)}$. Then $\gamma^2_{i_{m_{k+1}}} = (\ell_w \cdot 2^{i_{m_{k+1}}}/N) \cdot \mathsf{polylog}(\lambda)$. Recall our observation that $1/2 \cdot N^{(i_{m_{k+1}})} < 2^{i_{m_{k+1}}} \leq N^{(i_{m_{k+1}})}$, and $N^{(i_{m_{k+1}})} = cN$. Therefore the R.H.S. of Equation (***) simplifies to $1 - \mathsf{negl}(\lambda)$. Equation (***) is equivalent to saying that condition on the event $\bigwedge_{j=1}^{k} \hat{E}_{w,i_{m_j}}$, with probability $1 - \mathsf{negl}(\lambda)$, $X_{w,i_{m_{k+1}}} - E[X_{w,i_{m_{k+1}}} | \bigwedge_{j=1}^{k} \hat{E}_{w,i_{m_j}}]$ is at least $-\gamma_{i_{m_{k+1}}}$ (and at most

$\gamma_{i_{m_{k+1}}}$). We can bound the range of $E[X_{w,i_{m_{k+1}}} \,|\, \bigwedge_{j=1}^{k} \hat{E}_{w,i_{m_j}}]$ using Equation (****).

$$
\begin{aligned}
E[X_{w,i_{m_{k+1}}} \,|\, \bigwedge_{j=1}^{k} \hat{E}_{w,i_{m_j}}] &= \frac{\ell_w^{(i_{m_{k+1}})} \cdot 2^{i_{m_{k+1}}}}{N^{(i_{m_{k+1}})}} \\
\in & \left[\frac{(c\ell_w - \sum_{j=1}^{k} \delta_{i_{m_j}}) \cdot 2^{i_{m_{k+1}}}}{cN}, \frac{(c\ell_w + \sum_{j=1}^{k} \delta_{i_{m_j}}) \cdot 2^{i_{m_{k+1}}}}{cN} \right] \\
\in & \left[EX_{w,i_{m_{k+1}}} - \frac{(\sum_{j=1}^{k} \delta_{i_{m_j}}) \cdot 2^{i_{m_{k+1}}}}{cN}, EX_{w,i_{m_{k+1}}} + \frac{(\sum_{j=1}^{k} \delta_{i_{m_j}}) \cdot 2^{i_{m_{k+1}}}}{cN} \right] \\
\in & \left[EX_{w,i_{m_{k+1}}} - \sum_{j=1}^{k} \delta_{i_{m_j}}, EX_{w,i_{m_{k+1}}} + \sum_{j=1}^{k} \delta_{i_{m_j}} \right]
\end{aligned}
$$

The last step follows from the fact that $2^{i_{m_{k+1}}} \leq N^{(i_{m_{k+1}})} = cN$.

This means condition on the event $\bigwedge_{j=1}^{k} \hat{E}_{w,i_{m_j}}$, the random variable $X_{w,i_{m_{k+1}}}$ (*i.e.*, the number of matching entries in $E\hat{D}B_{i_{m_{k+1}}}$) is at least $EX_{w,i_{m_{k+1}}} - \sum_{j=1}^{k+1} \delta_j$ (and at most $EX_{w,i_{m_{k+1}}} + \sum_{j=1}^{k+1} \delta_j$) with probability $1 - \mathsf{negl}(\lambda)$.

Finally, $EX_{w,i_{m_{k+1}}} = \ell_w \cdot 2^{i_{m_{k+1}}}/N \in O(\ell_w)$ and $\sum_{j=1}^{k+1} \gamma_{i_{m_j}} \in o(\ell_w)$. So we have proved that $\Pr[\hat{E}_{w,i_{m_{k+1}}} \,|\, \bigwedge_{j=1}^{k} \hat{E}_{w,i_{m_j}}] > 1 - \mathsf{negl}(\lambda)$ and thus $\Pr[\bigwedge_{j=1}^{k+1} \hat{E}_{w,i_{m_j}}] = \Pr[\hat{E}_{w,i_{m_{k+1}}} \,|\, \bigwedge_{j=1}^{k} \hat{E}_{w,i_{m_j}}] \cdot \Pr[\bigwedge_{j=1}^{k} \hat{E}_{w,i_{m_j}}] > 1 - \mathsf{negl}(\lambda)$.

$\square$