DDR-SSE: Duplicated Retrieval of Documents for System-wide Secure Searchable Symmetric Encryption

Zichen Gui
University of Georgia

Simon-Philipp Merz
ETH Zürich

Kenneth G. Paterson
ETH Zürich

Sikhar Patranabis
IBM Research India

Abstract

Searchable Symmetric Encryption (SSE) schemes enable efficient keyword searches over encrypted documents at the cost of some leakage. An SSE scheme is said to be *system-wide secure* if it resists cryptanalysis by an adversary with access to leakage from retrieval of *both* encrypted indices *and* encrypted documents. The vast majority of state-of-the-art SSE schemes are, in fact, not system-wide secure (Gui *et al.*, IEEE S&P 2023). Currently, the only efficient system-wide secure SSE scheme is SWiSSSE (Gui *et al.*, PoPETS 2024). However, SWiSSSE requires a client state that is updated per query (which hinders adoption in various practical settings), and its leakage is hard to characterize precisely (thus making security analysis harder).

In this paper, we present DDR-SSE – a practically efficient, system-wide secure SSE scheme that only requires a static client state, and has a simple leakage profile. Technically, we introduce a novel encrypted document retrieval scheme that uses duplicated document storage and randomized document retrieval to suppress access pattern leakage without compromising on practical efficiency. A remarkable feature of our scheme is its conceptual simplicity (unlike SWiSSSE, which uses an extremely involved document retrieval mechanism).

We present a simulation-based security proof for DDR-SSE with respect to a rigorously formal system-wide leakage profile. Through extensive leakage cryptanalysis, we establish that DDR-SSE is resilient to query reconstruction attacks (even under "unrealistically" strong attack assumptions). Finally, we benchmark a prototype implementation of DDR-SSE and show that it scales smoothly to large databases of the size seen in real-world applications.

1 Introduction

Searchable Symmetric Encryption (SSE). Searchable Symmetric Encryption (SSE) [1–4] allows a client to execute queries over a symmetrically encrypted database stored at a third-party server. Most of the SSE literature focuses on the following simple single keyword query functionality: given an encrypted document collection in which each document is tagged with keywords, return the set of all documents tagged with a given keyword w. SSE schemes for keyword search can be broadly divided into two groups: *static* SSE schemes that do not allow updates to the document collection, and *dynamic* SSE schemes that allow the document collection to be updated on-the-fly.

Leakage vs Efficiency. The main security goal of SSE is to ensure data and query privacy of the client by minimizing information "leakage" to the (untrusted) server. An ideal SSE scheme would be provably leakage-free; however, achieving this typically requires heavy cryptographic machinery such as Oblivious RAM (ORAM) [5–11] or Private Information Retrieval (PIR) [12–16] that are not practically efficient at the required scale (see Appendix A for a more detailed discussion). It is common in the SSE literature to design schemes that trade-off some leakage for gains in efficiency. The leakage can be formalised, and it can be proven that a given scheme leaks no more than some specified leakage. However, this leaves open the possibility of cryptanalytic attacks exploiting the leakage [17–25]. It is challenging to design practically efficient SSE schemes that also resist leakage cryptanalysis.

Index-only SSE and Index-retrieval Leakage. The vast majority of existing SSE constructions [3, 4, 9, 26–35] consider only *index retrieval* where, by the end of a query, the client only obtains the relevant *document identifiers* instead of the *actual* documents. We refer to these schemes as *index-only* SSE schemes and the corresponding leakage as *index-retrieval leakage*. There exist many index-only SSE schemes that effectively suppress index-retrieval leakage [10, 36–39]. But these schemes do not consider the leakage from the actual retrieval of documents. Many of these index-only SSE scheme can be used as encrypted multi-maps where the payload can be any (fixed-size) short data instead of document identifiers.

End-to-End SSE and System-wide Leakage. A recent work [40] formally introduced the notion of *system-wide leakage* for SSE. In this setting, one considers *end-to-end* SSE

schemes that combine *both* index retrieval *and* document retrieval. In particular, an end-to-end SSE scheme is said to be system-wide secure if it resists leakage cryptanalysis by an adversary with access to leakage from the entirety of the SSE system, including the retrieval of *both* the document identifiers *and* the actual documents. We describe the contributions of [40] using the following running example with three documents:

- d_0 = "Apple and banana are both fruits but they are not the same fruit.",
- d_1 = "Apple is not banana.", and
- d_2 = "Banana is a tasty fruit.".

Suppose we want to index the documents with the following keywords (we use lower-case and stemmed keywords for illustration purposes): "apple", "banana", and "fruit", so that we can use one of the keywords as a query and retrieve the *documents* corresponding to the keyword. For example, if we make a query on "apple", we should obtain d_0 and d_1; for a query on "fruit", we should obtain d_0 and d_2.

Duplication in Index-only Schemes. Following the example above, careful readers may already spot a problem. If we store the documents naively (i.e., one copy each and retrieve them as they are), then the query on "apple" and the query on "fruit" reveal that d_0 is shared between the two queries. This is exactly the co-occurrence leakage exploited by numerous attacks in the literature [17–25].

It turns out that this is not a big problem for index-only SSE schemes (where the client only receives the document identifiers matching the query, not the documents themselves). The idea is that the client constructs an inverted search index (as shown in Figure 1a) where each document identifier is a fresh instance even if there are repetitions (as opposed to an inverted search index where the values are pointers). As a result, after encryption, the client will be touching distinct parts of the encrypted database and the access-pattern is suppressed. We refer to this as the duplication technique.

In the index-only setting, the duplication technique scales very well as the document identifiers (usually 8 bytes) have a similar size as pointers (8 bytes). However, duplication does not scale well on the documents as they are typically much larger (on the order of kilobytes or more). The authors of [40] considered extensions of index-only SSE to the system-wide setting and showed that there is no extension using existing techniques and primitives yielding an SSE scheme that is simultaneously efficient and system-wide secure.

Natural Extensions are Insecure. A natural extension of an index-only SSE scheme to system-wide secure SSE is to use an index-only SSE scheme for index retrieval and store and retrieve the (encrypted) documents naively. The authors of [40] show that the natural extensions are *not* system-wide secure. In particular, the system-wide leakage incurred by these schemes (in the document retrieval phase) can be exploited for efficient and highly scalable query recovery attacks, despite the index-only component having low leakage.

Known Techniques and Secure Primitives do not Scale. The authors of [40] considered realizing the document retrieval phase with different techniques and primitives that are secure such as applying using an index-only SSE scheme on the documents directly (see Figure 1b), storing the documents in an ORAM, and storing the documents using a private information retrieval scheme. These techniques do not scale in practice. In particular, for the Enron email database, the techniques and primitives incur storage overheads ranging from $4\times$ to $800\times$ and bandwidth overheads ranging from $20\times$ to $110,000\times$, as compared to the natural end-to-end SSE systems considered above (with no leakage mitigation for document retrieval). It should be obvious that these approaches are impractical for the large databases of the size seen in real-world applications.

System-wide Secure SSE. The above observations motivate efficient end-to-end SSE schemes that are *system-wide secure*. This is important as any deployment of encrypted document search needs to be secure in the system-wide sense. Currently, there are only two system-wide SSE schemes in the literature, namely CLRZ [41] and SWiSSSE [42]. CLRZ was shown to be insecure by many attacks [22, 25, 40]. And SWiSSSE is the only system-wide SSE scheme that has not been broken yet. SWiSSSE has small query and storage overheads, and resists highly refined system-wide leakage cryptanalysis. A key technique used in SWiSSSE is delayed, pseudorandom write-backs. Here, for each query, the corresponding responses (both the encrypted document identifiers and the encrypted documents) are removed from the encrypted database, stored temporarily in a client stash, and eventually written back to the encrypted database (in freshly encrypted form) after a (pseudo)random number of queries.

While pseudorandom write-backs efficiently suppress document-retrieval leakage, it incurs two penalties. First, it requires the client stash to be updated per query. This can be a hindrance for practical scenarios (e.g., for web-based email clients). Note that in the index-only SSE literature, it is usual for the client to have some small *static* storage (for cryptographic keys and some small metadata), which is considered reasonable in practice. The requirement to maintain a frequently updatable client stash presents a significant departure from this well-accepted design paradigm. Secondly, due to the delayed nature of the write-backs, it is complex to precisely characterize the leakage of SWiSSSE. This makes it hard to gain full confidence in its security, and may make it difficult to determine its resilience against any new system-wide cryptanalytic techniques that may emerge in the future.

In this paper, we ask the following question: *can we design a practically efficient, system-wide secure end-to-end SSE scheme with static client state such that its leakage can be precisely characterized and analyzed?*

Key	Value
apple‖0	0
apple‖1	1
banana‖0	0
banana‖1	1
banana‖2	2
fruit‖0	0
fruit‖1	2

(a) The inverted search index after duplication. An inverted search index of this form is typically used as an input to index-only SSE schemes. Repeated document identifiers are duplicated in the inverted search index, so queries on different keywords touch different parts of the inverted search index (after encryption).

Key	Value
apple‖0	Apple and banana are both fruits but they are not the same fruit.
apple‖1	Apple is not banana.
banana‖0	Apple and banana are both fruits but they are not the same fruit.
banana‖1	Apple is not banana.
banana‖2	Banana is a tasty fruit.
fruit‖0	Apple and banana are both fruits but they are not the same fruit.
fruit‖1	Banana is a tasty fruit.

(b) An index-only SSE scheme with duplication is applied to the documents directly. This gives a secure end-to-end SSE (if the documents are padded to a fixed length), but the storage overhead is impractical.

Figure 1: The duplication technique is typically used in index-only SSE schemes (left). It does not scale to the document retrieval problem (right).

1.1 Our Contributions

We answer the above question in the affirmative. We introduce DDR-SSE – a practically efficient, system-wide secure SSE scheme. DDR-SSE requires a small, *static* client state for cryptographic keys and small metadata, as in traditional index-only SSE schemes. Additionally, the leakage of DDR-SSE can be characterized precisely using a simple leakage function. As a result, it admits a simpler security analysis as compared to SWiSSSE. Finally, DDR-SSE matches SWiSSSE asymptotically in terms of encrypted storage costs and computational/communication overheads for keyword searches, while incurring a smaller volume of (static) client-side storage.

We present a simulation-based security proof for DDR-SSE with respect to a rigorously formal system-wide leakage profile. Through extensive leakage cryptanalysis, we establish that DDR-SSE is resilient to query reconstruction attacks (even for "unrealistically" strong attack assumptions). As part of this, we prove that the co-occurrence leakage counts of DDR-SSE are close to being uniform in a certain sense that is made more precise in the sequel. Finally, we benchmark a prototype implementation of DDR-SSE and show that it scales smoothly to large databases of the size typically found in real-world applications.

In the rest of this subsection, we present a more detailed overview of our contributions. For simplicity of exposition, we focus on a basic version of DDR-SSE here; details are presented in Section 3.

Modular Realization of System-Wide Secure SSE. A natural and modular approach to design a system-wide secure end-to-end SSE scheme is to combine a leakage suppressing index retrieval mechanism (such as those based on volume-hiding encrypted multi-maps [36–39]) with a leakage suppressing document retrieval mechanism. This is the approach we take here. However, [40] observed that there did not exist (prior to this work) a candidate document retrieval mechanism that is sufficiently leakage suppressing while also being efficient.

New Low-leakage Document Retrieval. As a core contribution, we address this open question by introducing a novel encrypted document retrieval scheme that uses duplicated document storage and randomized document retrieval to suppress access pattern leakage without compromising on practical efficiency. In particular, we show that by accessing the documents in a carefully designed, pseudorandom manner, surprisingly, we only need to store each document *twice* (as opposed to once per keyword as in index-only SSE schemes [36–39]) in order to suppress the co-occurrence leakage (i.e., the number of common documents accessed by two queries) that has been so heavily exploited in prior attacks against SSE schemes.

Uniformization of Co-occurrence Counts. More specifically, we are able to show that, if the original co-occurrence count for a pair of queried keywords w_i, w_j is $\mathtt{C}[i,j]$, then our design produces co-occurrence counts for document retrieval that are close to uniformly distributed over the interval $[0, \mathtt{C}[i,j]]$. The distribution still depends on $\mathtt{C}[i,j]$, but in a much weaker way than prior to the application of our technique. We note that these observed co-occurrence counts are fixed by the cryptographic keys used by our scheme, hence there are no attacks based on averaging out noise. We show that the uniformization of co-occurrence counts is sufficient to thwart known techniques for leakage cryptanalysis [17,18,20–23,25,40,43].

Building DDR-SSE. To build DDR-SSE, we combine our new document retrieval scheme with a generic, low-leakage index retrieval mechanism. Concretely, the index retrieval mechanism of DDR-SSE can be instantiated with any volume-hiding index-only SSE scheme, as long as its setup leaks no more than the total number of keyword-document pairs, and its query leakage is no larger than query equality leakage (i.e., if two queries are the same) and the maximum query response volume (i.e., the maximum number of documents matching any query; this is, in fact, the leakage for several existing volume-hiding index-only SSE schemes [36–39]). A remarkable feature of DDR-SSE is its conceptual simplicity (unlike SWiSSSE, which uses an extremely involved mechanism for document retrieval due to its use of dynamic addresses and a stash). We begin with a simplified construction called DDR-SSE-FP in Sections 3.1 to 3.3. We present our final construction DDR-SSE in Section 3.4.

System-Wide Leakage Analysis. We formally prove the security of DDR-SSE with respect to the system-wide security

definition of SSE introduced in [42] and the leakage of DDR-SSE, which we characterize precisely in Section 4.1. In order to establish the resistance of DDR-SSE to system-wide leakage cryptanalysis, we modify the powerful system-wide query reconstruction attack in [40] to work against DDR-SSE. Our experiments using the Enron email corpus show that DDR-SSE is resilient against such attacks even for strong (and unrealistically hostile) attack assumptions.

Instantiation and Implementation. We consider a concrete instance of DDR-SSE where index retrieval is based on a modification of XorMM [39] – a state-of-the-art, volume-hiding index-only SSE scheme with small leakage. We implement this instance of DDR-SSE in Java and benchmark its performance using the Enron email corpus. As a summary of our results, for a database of 400K emails (which we split into 4KB chunks, or 1.6GB of plaintext emails in total) with 32K unique keywords and 29M keyword-document pairs, the encrypted database consists of 1.3GB of encrypted metadata and encrypted indices, and 3.1GB of encrypted emails (4.4GB in total); the throughput of our scheme is 900 documents per second for setup and 39,000 documents per second for search.

Multi-server and Dynamic Extensions. We discuss a multi-server extension of basic DDR-SSE in Section 6 that can be instantiated with just two servers. Under the assumption of non-colluding servers, multi-server DDR-SSE has even less leakage than DDR-SSE and maintains the same efficiency. We leave the development of a dynamic version of DDR-SSE to future work, but sketch an initial approach in Section 6.

2 Notations and Definitions

Notations. We write $x \leftarrow\$ X$ to mean sample an element x from the set X. We use the notation $(\mathtt{out}_A;\mathtt{out}_B) \leftarrow \mathtt{func}(\mathtt{in}_A;\mathtt{in}_B)$ to denote an interactive protocol $\mathtt{func}()$ between parties A and B.

Databases. We define a database to be a set of documents, i.e., $\mathtt{DB} = \{d_i\}$. Each document is associated with a set of keywords which we denote as $W(d_i)$. We write $w\{\mathtt{DB}\}$ to mean the multiset of keywords in all documents. For a keyword w, we write $\mathtt{DB}(w)$ to mean the set of documents containing w, i.e., $\mathtt{DB}(w) = \{d_i \mid w \in W(d_i)\}$.

Inverted Search Index. Given a database $\mathtt{DB} = \{d_i\}$, an inverted search index $\mathtt{I}$ is defined to be a map between the keywords in the database and the document identifiers that contain the keywords. Specifically, $\mathtt{I}(w) = \{i \mid w \in W(d_i)\}$. We write $|\mathtt{I}|$ to mean the total number of keyword-identifier pairs in $\mathtt{I}$, i.e., $|\mathtt{I}| = \sum_w |\mathtt{I}(w)|$. Most of the literature on SSE studies the encrypted index retrieval problem from the inverted search index (or equivalently, encrypted retrieval from multi-maps with short values).

Key-Value Stores. The basic data structure used by DDR-SSE is a "key-value store". This data structure implements a dictionary that maps (non-cryptographic) keys to values. We require the following operations from a key-value store:

- Initialization: Create an empty key-value store.
- Put: $S[x] = y$ means create key x in the key-value store S and set the value associated to the key as y. If x is already in S, then the old value is overwritten by y.
- Get: $y \leftarrow S[x]$ means get the value associate with key x in the key-value store S.

2.1 System-Wide Static SSE

A system-wide static SSE scheme Π consists of protocols **Setup** and **Query** between a client and a server:

- **Setup**$(1^\lambda, \mathtt{param}, \mathtt{DB};)$ is a client-server protocol. The client takes as input a security parameter 1^λ, an additional parameter $\mathtt{param}$ and a database $\mathtt{DB}$, and the server takes no input. At the end of the **Setup** protocol, the client obtains a secret key sk and a state $\mathtt{st}_{\mathtt{Clt}}$ and the server obtains an encrypted database $\mathtt{EDB}$.
- **Query**$(sk, q, \mathtt{st}_{\mathtt{Clt}}; \mathtt{EDB})$ is a client-server protocol to support single-keyword queries. The query q contains a keyword w and potentially some additional arguments. The client takes as input a secret key sk, a search query q, and its internal state $\mathtt{st}_{\mathtt{Clt}}$. The server takes as input the encrypted database $\mathtt{EDB}$. The client and server interact with each other and by the end of the interaction, the client gets $(\mathtt{rspn}, \mathtt{st}_{\mathtt{Clt}})$ where $\mathtt{rspn}$ is the result of the query (i.e., the actual documents matching the query) and $\mathtt{st}_{\mathtt{Clt}}$ is the new state of the client; the server state $\mathtt{EDB}$ may also be modified following the execution.

We say that Π is correct if given a security parameter λ, for all databases $\mathtt{DB}$, for all sequences of queries $q_0, \ldots, q_\ell$,

$$(\mathtt{st}_{\mathtt{Clt}}; \mathtt{EDB}) \leftarrow \Pi.\textbf{Setup}(1^\lambda, \mathtt{DB};)$$
$$(\mathtt{rspn}_i, \mathtt{st}_{\mathtt{Clt}}; \mathtt{EDB}) \leftarrow \Pi.\textbf{Query}(sk, q_i, \mathtt{st}_{\mathtt{Clt}}; \mathtt{EDB})$$

and $\mathtt{rspn}_i = \mathtt{DB}(w(q_i))$, where $w(q_i)$ is the queried keyword.[1]

2.2 System-Wide Security for Static SSE

We say that a static SSE scheme is system-wide secure with respect to a leakage function $\mathcal{L} = (\mathcal{L}_{\textbf{Setup}}, \mathcal{L}_{\textbf{Query}})$ if no (honest-but-curious) adversary can distinguish between the real execution and an execution simulated given only the leakage, as outlined below.

Definition 1 (System-Wide Security of Static SSE)**.** Let $\Pi = (\textbf{Setup}, \textbf{Query})$ be a static SSE scheme and consider the following probabilistic experiment where $\mathcal{A}$ is a stateful, probabilistic polynomial time (PPT) honest-but-curious adversary, $\mathcal{S}$ is a stateful simulator and $\mathcal{L}$ is the leakage function.

$\underline{\textbf{Real}_{\Pi,\mathcal{A}}(1^\lambda):}$

[1] One may relax the correctness requirement to $\mathtt{rspn}_i \subseteq \mathtt{DB}(w(q_i))$ for schemes that include noise in the query response.

1. The adversary $\mathcal{A}$ selects a database $\mathtt{DB}$ and gives it to the challenger $\mathcal{C}$.
2. The challenger $\mathcal{C}$ generates a key sk, and encrypts the database as $\mathtt{EDB} \leftarrow \mathbf{Setup}(1^\lambda, sk, \mathtt{DB})$. The challenger $\mathcal{C}$ sends the encrypted database $\mathtt{EDB}$ to $\mathcal{A}$.
3. The adversary picks polynomially many keyword search queries $q_1, \ldots, q_{\mathtt{poly}(\lambda)}$. For each query, the challenger $\mathcal{C}$ interacts with the adversary $\mathcal{A}$ to execute the search query protocol (including the final document retrieval phase), where the challenger plays the role of the server and the adversary plays the role of the client.
4. Finally, the adversary $\mathcal{A}$ outputs a bit $b \in \{0,1\}$.

$\mathbf{Ideal}_{\Pi,\mathcal{A},\mathcal{S}}(1^\lambda, \mathcal{L} = (\mathcal{L}_{\mathbf{Setup}}, \mathcal{L}_{\mathbf{Query}}))$:

1. The adversary $\mathcal{A}$ selects a database $\mathtt{DB}$ and gives the leakage $\mathcal{L}_{\mathbf{Setup}}(\mathtt{DB})$ to the simulator $\mathcal{S}$.
2. Using $\mathcal{L}_{\mathbf{Setup}}(\mathtt{DB})$, the simulator $\mathcal{S}$ generates $\mathtt{EDB}$ and returns it to $\mathcal{A}$.
3. The adversary picks a polynomial number of search queries $q_1, \ldots, q_{\mathtt{poly}(\lambda)}$. For each query q_i, the simulator $\mathcal{S}$ (playing the role of server) takes $\mathcal{L}_{\mathbf{Query}}(q_i)$ (including the leakage from the document retrieval phase corresponding to the query) as input and interacts with the adversary $\mathcal{A}$ (playing the role of client) to execute the search query protocol.
4. Finally, the adversary $\mathcal{A}$ outputs a bit $b \in \{0,1\}$.

We say that Π is $\mathcal{L}$-system-wide-secure if there exists a PPT simulator $\mathcal{S}$ such that for all PPT adversaries $\mathcal{A}$,

$$\left|\Pr\left[\mathbf{Real}_{\Pi,\mathcal{A}}(1^\lambda) = 1\right] - \Pr\left[\mathbf{Ideal}_{\Pi,\mathcal{A},\mathcal{S}}(1^\lambda, \mathcal{L}) = 1\right]\right| \leq \mathtt{negl}(\lambda).$$

Index-Only SSE and Encrypted Document Retrieval. DDR-SSE splits the construction of a system-wide secure SSE scheme into two components. The first component is an index retrieval scheme which allows the client to obtain the document identifiers of the documents matching the query. The second component is an encrypted document store which allows the client to retrieve the actual documents (securely) with the document identifiers obtained in the first step. Therefore, it makes sense for us to talk about the syntax and security of index-only SSE and encrypted document retrieval. The definitions of index-only SSE and encrypted document retrieval can be easily derived by modification of the system-wide SSE definitions presented above.

Discussion. We do not use the structured encryption framework proposed in [3] because the document retrieval problem is a different problem from the index retrieval problem, and it is not a typical problem considered in the structured encryption framework, as we have demonstrated in Section 1.

3 DDR-SSE for Static Databases

In this section, we describe DDR-SSE for static databases. We begin with a general construction (DDR-SSE-FP) which uses an index-only SSE scheme in a black-box manner (Sections 3.1 to 3.3). We then show our final construction (DDR-SSE) which uses an index-only SSE scheme in a grey-box manner (Section 3.4). DDR-SSE-FP and DDR-SSE have the same leakage, but DDR-SSE has better efficiency.

3.1 Overview of DDR-SSE-FP

Two-Stage System-Wide Secure SSE scheme. DDR-SSE-FP makes use of two encrypted data structures to build a system-wide secure SSE scheme, namely an *index-only SSE scheme* for index retrieval and an *encrypted document retrieval scheme* to retrieve the actual documents.

Encrypted Index Retrieval. DDR-SSE-FP uses a generic, volume-hiding, index-only SSE scheme for index retrieval, with the restriction that this scheme should be correct, and it has the size of the inverted search index as the setup leakage, and the maximum query response volume and query equality leakage as the query leakage. We note that several existing volume-hiding index-only SSE schemes [36–39] can be used to instantiate DDR-SSE-FP. We see modularity as a strength of DDR-SSE-FP. We choose the XorMM scheme from [39] as a concrete instantiation of the index-only SSE scheme used in DDR-SSE-FP(and DDR-SSE) starting from Section 3.6. It is used for discussions on performance and resistance to system-wide leakage cryptanalysis of DDR-SSE-FP (and DDR-SSE).

Encrypted Document Retrieval. We devise a new construction for encrypted document retrieval. The documents are stored in a key-value store in our construction. The key features of the construction are that: (1) it is stateless, (2) its storage overhead with respect to naive document storage is only a factor of 2, and (3) it has an optimal communication overhead.

Duplicated Document Storage and Randomized Document Retrieval. Our encrypted document retrieval scheme suppresses access pattern leakage by using two core techniques, namely duplication and randomized document retrieval. In the setup phase, our encrypted document retrieval scheme generates two encrypted copies (freshly encrypted each time) of each document and stores them in two locations (of a key-value store).

To run a query that retrieves documents $i_0, \ldots, i_j$, we pick a pseudorandom value k between 0 and j and generate a pseudorandom bit b from $\{0,1\}$. If $b = 0$, then we retrieve the first k documents from the first copy of the documents (by accessing the documents with the addresses associated to the first copy) and the other documents from the second copy; otherwise we retrieve the first k documents from the second copy of the documents and the other documents from the first copy. The seed for the pseudorandom processes is fixed (and only depends on the queried keyword) so that repeated queries for the same keyword do not generate additional leakage.

We show in Section 4.2 that this simple randomization for document retrieval is very effective in suppressing leakage.

As a technical remark, the document identifiers matching each query need to be randomly distributed for the cryptanalysis to hold. This can be achieved by randomizing the document identifiers (by using a pseudorandom permutation) before running the setup procedure of our scheme. The randomization does not affect the correctness of our scheme. For simplicity, we thus omit this step in the presentation of our scheme.

Randomization of Document Identifiers. The main cryptanalysis result of DDR-SSE (Theorem 3) only holds when the document identifiers are randomly distributed. We achieve this by using a small-domain pseudorandom permutation on the raw document identifiers before running the setup procedure of the index-only SSE scheme and encrypted document storage in DDR-SSE.

Bucketization. Most of the volume-hiding index-only SSE schemes achieve volume-hiding by retrieving $\max_w |\mathtt{DB}(w)|$ number of document identifiers for any search query. The query response is then filtered on the client to extract the document identifiers matching the query. Using the Enron email corpus (400K emails) as an example, $\max_w |\mathtt{DB}(w)|$ is approximately 20K. Assume that we have a volume-hiding index-only SSE scheme with optimal bandwidth cost, the document identifiers are 4-byte long, and encrypting these document identifiers does not introduce any storage overhead. Then, each search query requires 80KB communication from the server to the client. While the asymptotic communication overhead looks terrible for search queries with a small number of matching document identifiers, the concrete communication cost is completely acceptable.

On the other hand, retrieving 20K (encrypted) documents for each search query has a huge concrete communication cost. If we assume that each email has length 4KB, retrieving 20K emails costs 80MB of bandwidth even though the user may only be interested in a handful of emails. Therefore, we adapt the bucketization technique used in SWiSSSE [42] to reduce the communication complexities of DDR-SSE-FP. More details on the bucketization technique can be found in Section 3.2 and a discussion on the parameter selection for bucketization can be found in Appendix B.3.

Fixed-length Documents. For simplicity, we present DDR-SSE for fixed-length documents. That is, for $\mathtt{DB} = \{d_i\}$, we assume that for all i, $|d_i| = L$ for some constant L. Of course, in practice, this does not hold. So in our implementation, we split documents larger than 4KB into 4KB chunks (we treat each chunk as a document; the chunks inherit all keywords of the original documents). Then, all documents are padded to size 4KB before encryption.

We present a detailed discussion of the effect of splitting the documents on the leakage of DDR-SSE in Section 4.1. The leakage cryptanalysis in Section 4.2 works with our implementation directly.

Building Blocks. DDR-SSE-FP relies on:

- A volume-hiding index-only SSE scheme $\Pi_{\mathtt{I}} = (\mathbf{Setup}, \mathbf{Query})$. $\Pi_{\mathtt{I}}$ should be correct (i.e., the client receives exactly the set of matching document identifiers after each query). The leakage of $\Pi_{\mathtt{I}}$ should be no worse than the size of the inverted search index as the setup leakage, and the maximum query response volume and query equality leakage as the query leakage.
- A pseudorandom permutation (PRP) $P : \{0,1\}^{\lambda} \times \{0,1\}^{L} \to \{0,1\}^{L}$, where 1^{λ} is the security parameter of the PRP and L is the size of the input and output.
- A pseudorandom function (PRF) $F : \{0,1\}^{\lambda} \times \{0,1\}^{*} \to \{0,1\}^{L}$, where 1^{λ} is the security parameter of the PRF and L is the size of the output.
- An IND-CPA secure symmetric encryption scheme $\Pi_{\mathbf{Enc}} = (\mathbf{KGen}, \mathbf{Enc}, \mathbf{Dec})$.

3.2 The Setup Procedure of DDR-SSE-FP

The setup procedure of DDR-SSE-FP can be broken down into five sub-procedures:

1. **Randomization of document identifiers:** The sub-process is carried out when building the plaintext index for the documents. We note while that the process uses a PRP, we do not have to keep the key of the PRP around after the setup, as the remaining procedures will work with the permuted document identifiers directly.
2. **Bucketization:** The sub-procedure takes as input a database $\mathtt{DB} = \{d_i\}$, and outputs an inverted index $\mathtt{I}$, where the frequency of the keywords are padded into "buckets".
3. **Setup of encrypted metadata:** The sub-procedure generates a PRF key and builds encrypted metadata which contains the real keyword frequencies and the padded keyword frequencies.
4. **Setup of encrypted inverted index:** The sub-procedure runs $\Pi_{\mathtt{I}}.\mathbf{Setup}()$ with the inverted index $\mathtt{I}$ produced by the bucketization sub-procedure. It produces a secret key $sk_{\mathtt{I}}$ and an encrypted inverted index $\mathtt{EI}$.
5. **Setup of encrypted documents:** The sub-procedure begins by generating a PRF key and an encryption key. Given the documents $\mathtt{DB} = \{d_i\}$ as the input, it then produces an encrypted key-value store where each document has two encrypted copies in the key-value store.

The output of the setup procedure consists of: (i) secret keys used for the encrypted inverted index and the encrypted documents (stored on the client), (ii) encrypted metadata of the keywords (i.e., real frequencies and padded frequencies; stored on the server), and (iii) encrypted inverted index and encrypted documents (stored on the server).

We give more details on the bucketization sub-procedure and the setup of the encrypted documents below. The pseudocode of **Setup** appears in Algorithm 1.

Bucketization. As we have explained in the overview, we want to avoid the communication overhead of worst-case padding in the document retrieval phase. For that rea-

son, DDR-SSE adapts the bucketization technique from SWiSSSE [42].

Concretely, let $\mathtt{I}$ be an inverted index obtained by parsing a database naively and $w_0, \ldots, w_{l-1}$ be the keywords in $\mathtt{I}$. Without loss of generality, assume that $|\mathtt{I}[w_i]| \geq |\mathtt{I}[w_j]|$ for all $i < j$. Let the bucket size be $\mathtt{bkt} \in \mathbb{Z}^+$, then the bucketization procedure adds random document identifiers to $\mathtt{I}$ such that in the end, $|\mathtt{I}[w_i]| = |\mathtt{I}[w_j]|$ for all pairs of indices i, j with $\lfloor i/\mathtt{bkt} \rfloor = \lfloor j/\mathtt{bkt} \rfloor$.

To ensure that the client can remove the random documents in the end, $\mathtt{I}$ is implemented as a list and the random document identifiers are added to its end. The true query response volume (without bucketization) is stored in the encrypted metadata and will be used to truncate the query response during queries.

Setup of the Encrypted Inverted Index. The encrypted inverted index is set up with the *bucketized* inverted index. As a result, the client will be able to retrieve the same set of documents (including the random ones) for each query in the document retrieval phase. This prevents an attacker from removing the random documents from the access pattern leakage by observing multiple queries on the same keyword.

Setup of the Encrypted Documents. Let 1^{λ_1} be the security parameter of the PRF and 1^{λ_2} be the security parameter of the encryption scheme. The sub-procedure begins by generating a PRF key $sk_F \leftarrow\$ \{0,1\}^{\lambda_1}$ and an encryption key $sk_{\mathbf{Enc}} \leftarrow \Pi_{\mathbf{Enc}}.\mathbf{KGen}(1^{\lambda_2})$. It also initializes an empty key-value store $\mathtt{EDoc}$. Then for each document d_i, the sub-procedure generates two addresses $F(sk_F, i \,\|\, 0)$ and $F(sk_F, i \,\|\, 1)$ and stores a freshly encrypted copy of the document $\mathbf{Enc}(sk_{\mathbf{Enc}}, d_i)$ at each of these two addresses in $\mathtt{EDoc}$.

3.3 The Query Procedure of DDR-SSE-FP

The query procedure consists of three sub-procedures:

Metadata retrieval. The client queries the encrypted metadata for a keyword w and retrieves the true and padded query response volumes.

Index retrieval. For a query on w, the client and the server run $\Pi_{\mathtt{I}}.\mathbf{Query}()$ on w to retrieve a list of document identifiers in DDR-SSE. See Algorithm 2 for the pseudocode of the **Query** procedure. By correctness of $\Pi_{\mathtt{I}}$, we expect to see $id_0, \ldots, id_{\mathtt{PF}_w - 1}$ as the query response, where $\mathtt{PF}_w$ is the query response volume after padding of keyword w. Metadata retrieval and index retrieval can be packed into a single round of communication.

Document retrieval. The document retrieval sub-procedure takes as input the queried keyword w, the list of document identifiers $id_0, \ldots, id_{\mathtt{PF}_w - 1}$ obtained in the index retrieval and the encrypted documents, and outputs the list of documents $d_{id_0}, \ldots, d_{id_{\mathtt{PF}_w - 1}}$ to the client. The client then uses the true query response volume obtained in the metadata retrieval procedure to filter the query response. The client proceeds

Algorithm 1 The **Setup** Procedure of DDR-SSE-FP

Input: The security parameter of the index-only SSE scheme 1^{λ_0}, the security parameter of the PRF 1^{λ_1}, the security parameter of the encryption scheme 1^{λ_2}, the security parameter of the PRP 1^{λ_3}, the bucket size $\mathtt{bkt}$, and the plaintext database $\mathtt{DB}$.
procedure $\textsc{Setup}((1^{\lambda_0}, 1^{\lambda_1}, 1^{\lambda_2}, 1^{\lambda_3}), \mathtt{bkt}, \mathtt{DB})$
  / Index Construction */*
  $sk_P \leftarrow \{0,1\}^{\lambda_3}$
  $\mathtt{I} \leftarrow \mathtt{dict}()$
  for $d_i \leftarrow \mathtt{DB}$ **do**
    for $w \leftarrow W(d_i)$ **do**
      $\mathtt{I}[w] \leftarrow \mathtt{I}[w] + [P(sk_P, i)]$ ▷ $\mathtt{I}[w] = [\,]$ if $w \notin \mathtt{I}$, the domain and co-domain of P are $|\mathtt{DB}|$
  / Bucketization */*
  $(w_0, \ldots, w_{l-1}) \leftarrow \mathtt{sorted}(\{w \in \mathtt{I}\}, \mathtt{key} = |\mathtt{I}[w]|)$ ▷ Sort the keywords from the highest frequency to the lowest frequency
  for $i \in \{0, \ldots, l-1\}$ **do**
    while $|\mathtt{I}[w_i]| < |\mathtt{I}[w_{\lfloor i/\mathtt{bkt} \rfloor \cdot \mathtt{bkt}}]|$ **do**
      $r \leftarrow\$ \{0, \ldots, |\mathtt{DB}| - 1\}$
      $\mathtt{I}[w_i] \leftarrow \mathtt{I}[w_i] + [r]$
  / Metadata on keyword frequencies */*
  $sk_{\mathtt{EMD}} \leftarrow\$ \{0,1\}^{\lambda_1}$
  $\mathtt{EMD} \leftarrow \mathtt{dict}()$
  for $i \in \{0, \ldots, l-1\}$ **do**
    $\mathtt{EMD}[F(sk_{\mathtt{EMD}}, w_i \,\|\, 0)] \leftarrow (|\mathtt{DB}(w_i)| \oplus F(sk_{\mathtt{EMD}}, w_i \,\|\, 1)) \,\|\, (|\mathtt{I}[w_i]| \oplus F(sk_{\mathtt{EMD}}, w_i \,\|\, 2))$
  / Setup of the encrypted inverted index */*
  $(sk_{\mathtt{I}}, \mathtt{EI}) \leftarrow \Pi_{\mathtt{I}}.\mathbf{Setup}(1^{\lambda_0}, \mathtt{I})$
  / Setup of the encrypted documents */*
  $sk_F \leftarrow\$ \{0,1\}^{\lambda_1}$
  $sk_{\mathbf{Enc}} \leftarrow \Pi_{\mathbf{Enc}}.\mathbf{KGen}(1^{\lambda_2})$
  $\mathtt{EDoc} \leftarrow \mathtt{dict}()$
  for $i \leftarrow 0, \ldots, |\mathtt{DB}| - 1$ **do**
    $\mathtt{EDoc}[F(sk_F, P(sk_P, i) \,\|\, 0)] \leftarrow \Pi.\mathbf{Enc}(sk_{\mathbf{Enc}}, d_i)$
    $\mathtt{EDoc}[F(sk_F, P(sk_P, i) \,\|\, 1)] \leftarrow \Pi.\mathbf{Enc}(sk_{\mathbf{Enc}}, d_i)$
  return $((sk_{\mathtt{EMD}}, sk_{\mathtt{I}}, sk_F, sk_{\mathbf{Enc}}); (\mathtt{EMD}, \mathtt{EI}, \mathtt{EDoc}))$

by sorting $(id_0, \ldots, id_{\mathtt{PF}_w - 1})$ to $(id'_0, \ldots, id'_{\mathtt{PF}_w - 1})$ in increasing order, which is a step necessary to get the desired query leakage. Then, they generate two pseudorandom integers – a bit b and an integer $r \in \{0, \ldots, \mathtt{PF}_w\}$. The bit b determines from which of the two copies of the encrypted documents the first r documents are retrieved. The remaining documents are retrieved from the other copy. Hereby, the different copies can be distinguished using an indicator bit. Given the secret key and the queried keyword, the two pseudorandom integers are deterministic which means that multiple queries on the same keyword will not leak additional information. While some of the choices made may at first appear arbitrary, we will show how they help with leakage suppression in Section 4.1.

3.4 Grey-box Modification

In this section, we show an optimization we can apply to DDR-SSE-FP if it is instantiated with an index-only SSE scheme with certain properties. The optimization introduces no additional leakage to our scheme.

Volume-hiding Index-only SSE with Adjustable Padding. While this does not apply to all volume-hiding index-only SSE schemes, there are constructions (such as [37, 39]) in which one can "adjust" the amount of padding in queries. Loosely speaking, some constructions store encrypted document identifiers of a given keyword w at ad-

Algorithm 2 The **Query** Procedure of DDR-SSE-FP

Input: Secret keys $sk_{\texttt{EMD}}, sk_{\texttt{I}}, sk_F, sk_{\textbf{Enc}}$ and keyword w from the client; encrypted metadata $\texttt{EMD}$, encrypted inverted index $\texttt{EI}$, and encrypted documents $\texttt{EDoc}$ from the server.
procedure $\textsc{Query}((sk_{\texttt{EMD}}, sk_{\texttt{I}}, sk_F, sk_{\textbf{Enc}}), w; (\texttt{EMD}, \texttt{EI}, \texttt{EDoc}))$
/* *Metadata Retrieval* */
Run by the client
  $\texttt{addr} \leftarrow F(sk_{\texttt{EMD}}, w \,\|\, 0)$
Run by the server
  $u \,\|\, v \leftarrow \texttt{EMD}[\texttt{addr}]$
Run by the client
  $\texttt{TF}_w \leftarrow u \oplus F(sk_{\texttt{EMD}}, w \,\|\, 1)$
  $\texttt{PF}_w \leftarrow v \oplus F(sk_{\texttt{EMD}}, w \,\|\, 2)$
/* *Index Retrieval* */
  $(id_0, \ldots, id_{\texttt{PF}_w - 1}) \leftarrow \Pi_{\texttt{I}}.\textbf{Query}(sk_{\texttt{I}}, w; \texttt{EI})$
/* *Document Retrieval* */
Run by the client
  $(id'_0, \ldots, id'_{\texttt{PF}_w - 1}) \leftarrow \texttt{sorted}(id_0, \ldots, id_{\texttt{PF}_w - 1})$ ▷ Sort the document identifiers in increasing order
  $b \leftarrow \texttt{RandInt}(1; (sk_F, w, 0))$ ▷ $\texttt{RandInt}(n; \texttt{seed})$ returns a pseudorandom integer uniformly distributed over $[0, n]$ for seed value $\texttt{seed}$
  $r \leftarrow \texttt{RandInt}(\texttt{PF}_w; (sk_F, w, 1))$
  $\texttt{addrs} \leftarrow \emptyset$
  for $i \leftarrow \{0, \ldots, \texttt{PF}_w - 1\}$ **do**
    if $b = 0 \wedge i < r$ **then**
      $\texttt{addrs} \leftarrow \texttt{addrs} \cup \{F(sk_F, id'_i \,\|\, 0)\}$
    if $b = 0 \wedge i \geq r$ **then**
      $\texttt{addrs} \leftarrow \texttt{addrs} \cup \{F(sk_F, id'_i \,\|\, 1)\}$
    if $b = 1 \wedge i < r$ **then**
      $\texttt{addrs} \leftarrow \texttt{addrs} \cup \{F(sk_F, id'_i \,\|\, 1)\}$
    if $b = 1 \wedge i \geq r$ **then**
      $\texttt{addrs} \leftarrow \texttt{addrs} \cup \{F(sk_F, id'_i \,\|\, 0)\}$
Run by the server
  $\texttt{rspn} \leftarrow \emptyset$
  for $\texttt{addr} \in \texttt{addrs}$ **do**
    $\texttt{rspn} \leftarrow \texttt{rspn} \cup \{\texttt{EDoc}[\texttt{addr}]\}$
Run by the client
  $\texttt{res} \leftarrow \{\textbf{Dec}(sk_{\textbf{Enc}}, ed) \mid ed \in \texttt{rspn}\}$
  Remove documents with identifier in $\{id_{\texttt{TF}_w}, \ldots, id_{\texttt{PF}_w - 1}\}$ from $\texttt{res}$
  return $(\texttt{res}; (\texttt{EMD}, \texttt{EI}, \texttt{EDoc}))$

Algorithm 3 The **Query** Procedure of DDR-SSE

Input: Secret keys $sk_{\texttt{EMD}}, sk_{\texttt{I}}, sk_F, sk_{\textbf{Enc}}$ and keyword w from the client; encrypted metadata $\texttt{EMD}$, encrypted inverted index $\texttt{EI}$, and encrypted documents $\texttt{EDoc}$ from the server.
procedure $\textsc{Query}((sk_{\texttt{EMD}}, sk_{\texttt{I}}, sk_F, sk_{\textbf{Enc}}), w; (\texttt{EMD}, \texttt{EI}, \texttt{EDoc}))$
/* *Metadata Retrieval* */
Run by the client
  $\texttt{addr} \leftarrow F(sk_{\texttt{EMD}}, w \,\|\, 0)$
  $x \leftarrow F(sk_{\texttt{EMD}}, w \,\|\, 2)$
Run by the server
  $u \,\|\, v \leftarrow \texttt{EMD}[\texttt{addr}]$
  $\texttt{PF}_w \leftarrow v \oplus x$
Run by the client
  $\texttt{TF}_w \leftarrow u \oplus F(sk_{\texttt{EMD}}, w \,\|\, 1)$
/* *Index Retrieval* */
  $(id_0, \ldots, id_{\texttt{PF}_w - 1}) \leftarrow \Pi_{\texttt{I}}.\textbf{Query}(sk_{\texttt{I}}, w, \texttt{PF}_w; \texttt{EI})$
(Same as Line 8 of Algorithm 2 from this point.)

dresses $F(sk_F, w \,\|\, 0), \ldots, F(sk_F, w \,\|\, \texttt{PF}_w - 1)$ in some data structure, where F is a PRF, sk_F is a PRF key, and $\texttt{PF}_w$ is the query response volume of w. During a query on w, instead of just retrieving the encrypted document identifiers from $F(sk_F, w \,\|\, 0), \ldots, F(sk_F, w \,\|\, \texttt{PF}_w - 1)$, the client also retrieves encrypted document identifiers from $F(sk_F, w \,\|\, \texttt{PF}_w), \ldots, F(sk_F, w \,\|\, \texttt{PF}_{\max} - 1)$, where $\texttt{PF}_{\max}$ is the maximum query response volume. With a careful choice of the data structure, the latter addresses are also valid addresses in the data structure, so the server is not able to differentiate these dummy retrievals from the real ones, leading to the desired volume-hiding property.

A direct consequence of these constructions is that it is possible to adjust the amount of padding (at the cost of worse leakage) in a grey-box manner. We call our scheme with the grey-box modification DDR-SSE. Pseudocode for the query procedure of DDR-SSE can be found in Algorithm 3; the modification does not change the setup procedure from DDR-SSE-FP.

A More Storage-Efficient Variant of DDR-SSE. Careful readers may wonder why DDR-SSE needs to use a volume-hiding index-only SSE scheme for index retrieval. Indeed, any index-only SSE scheme suffices for the job in the way the scheme is constructed, since padding is done explicitly during the setup. However, there is a more storage-efficient variant of DDR-SSE which requires a volume-hiding index-only SSE scheme. In this variant, we run the setup algorithm of DDR-SSE without explicit bucketization (i.e., remove lines 9 to 13 of Algorithm 1 while still record the padded query response volume on line 17 of Algorithm 1). Then, during a query on keyword w, the client begins by querying $\texttt{EMD}$ and learning the true query response volume $\texttt{TF}_w$ and the padded query response volume $\texttt{PF}_w$. The client will then query the volume-hiding index-only SSE scheme (with the query response volume padded to $\texttt{PF}_w$) and learn the $\texttt{TF}_w$ real document identifiers matching the query. After that, the client can pad the number of document identifiers to use for the document retrieval pseudorandomly and locally to $\texttt{PF}_w$, before running the rest of the query procedure. This variant is more storage-efficient than DDR-SSE, as the padded document identifiers do not need to be stored explicitly. However, we opt to not present this as our main construction as its security proof would be more involved.

Comparing DDR-SSE to Other Leakage Suppression Techniques. The padding strategy used in DDR-SSE is the same as that in [42]. There were two padding strategies prior to this. In [18], the authors propose to pad the number of documents matching each keyword to a multiple of x (for some predetermined x). In [10], the authors propose to pad the number of documents matching each keyword to a power of x. These padding strategies are different from that of DDR-SSE as they do not ensure that multiple keywords will have the same query response volume after padding. Thus, [18] and [10] are more susceptible to attacks that exploit volume leakage (e.g., [22]).

PANCAKE [44] is an encrypted key-value store which is designed to suppress query equality leakage. While PANCAKE also uses some form of duplication and randomized accesses, these techniques and their goals are very different from the ones used in DDR-SSE. Therefore, we consider PANCAKE as an orthogonal work.

Correctness of DDR-SSE. We state the following theorem

for the correctness of DDR-SSE.[2] Due to space constraints, we defer a detailed proof of Theorem 1 to the full version of the paper.

Theorem 1 (Correctness of DDR-SSE)**.** Let $\mathtt{DB}$ be a database and $|\mathtt{DB}|$ be the number of documents in the database. Let $\mathtt{bkt}$ be the bucket size of the bucketization procedure. Let 1^{λ_0} be the security parameter of the index-only SSE scheme used in DDR-SSE and 1^{λ_1} be the security parameter of F used for the addresses of the encrypted documents. Let ℓ denote the output length of F. Let $Adv^{\mathtt{Corr}}_{\Pi_{\mathtt{I}}}(1^{\lambda_0},\mathtt{DB},\mathtt{bkt})$ be the failure probability of the index-only SSE scheme on database $\mathtt{DB}$ after bucketization with the bucket size $\mathtt{bkt}$ and $Adv^{\mathtt{PRF},2\cdot|\mathtt{DB}|}_{F,\mathcal{B}}(1^{\lambda_1})$ be the advantage of PPT adversary $\mathcal{B}$ in distinguishing F from a true random function in $2\cdot|\mathtt{DB}|$ queries. The advantage of any PPT adversary $\mathcal{A}$ in breaking the correctness of DDR-SSE over the database $\mathtt{DB}$ is at most:

$$\frac{|\mathtt{DB}|^2}{2^{\ell+1}}+\frac{|\mathtt{DB}|^2}{2^{\ell-1}}+Adv^{\mathtt{Corr}}_{\Pi_{\mathtt{I}}}(1^{\lambda_0},\mathtt{DB},\mathtt{bkt})$$
$$+Adv^{\mathtt{PRF},|\mathtt{DB}|}_{F,\mathcal{B}}(1^{\lambda_1})+Adv^{\mathtt{PRF},2|\mathtt{DB}|}_{F,\mathcal{B}}(1^{\lambda_1}).$$

Proof. The correctness of DDR-SSE can be broken down into the correctness of the encrypted inverted index and the correctness of the encrypted metadata and encrypted documents. The failure probability of the earlier is bounded by $Adv^{\mathtt{Corr}}_{\Pi_{\mathtt{I}}}(1^{\lambda_0},\mathtt{DB},\mathtt{bkt})$. For the encrypted metadata and encrypted documents, DDR-SSE can fail if there are collisions in the addresses of the metadata entries or the encrypted documents. The collision probability is bounded by the collision probability of the underlying pseudorandom function F which is $\frac{|\mathtt{DB}|^2}{2^{l+1}}+Adv^{\mathtt{PRF},|\mathtt{DB}|}_{F,\mathcal{B}}(1^{\lambda_1})$ for the metadata and $\frac{|\mathtt{DB}|^2}{2^{l-1}}+Adv^{\mathtt{PRF},2|\mathtt{DB}|}_{F,\mathcal{B}}(1^{\lambda_1})$ for the encrypted documents respectively. Combining the terms yields the desired failure probability. □

3.5 System-Wide Leakage of DDR-SSE

In this section, we formally define the leakage of DDR-SSE and prove its security with respect to Definition 1.

Leakage at Setup. The leakage at setup can be broken down into two parts. The first part of the leakage comes from the setup of encrypted metadata and encrypted inverted index. The encrypted metadata is a key-value store with $|W|$ entries, where W is the set of keywords. For the encrypted inverted index, DDR-SSE inherits the setup leakage from the volume-hiding index-only SSE as we simply invoke the volume-hiding index-only SSE during the setup. One caveat is that we ran bucketization on the inverted index before running the setup of the volume-hiding index-only SSE. This means that instead of leaking the total size of the inverted index (i.e., the total number of keyword-document pairs), DDR-SSE leaks the total size of the bucketized version of the inverted index. We denote the bucketized database as $\mathtt{DB}_{\mathtt{bkt}}$ and the padded size of the inverted index as $|W\{\mathtt{DB}_{\mathtt{bkt}}\}|$.

The second part of the leakage comes from the setup of encrypted documents. DDR-SSE produces two encrypted documents for each raw document so DDR-SSE leaks the number of documents $|\mathtt{DB}|$ in the setup of encrypted documents. Overall, the setup leakage of DDR-SSE is

$$\mathcal{L}_{\mathbf{Setup}}(\mathtt{DB},\mathtt{bkt})=(|W|,|W\{\mathtt{DB}_{\mathtt{bkt}}\}|,|\mathtt{DB}|).$$

Leakage During Queries. Let w be the queried keyword. It is easy to see that metadata retrieval and index retrieval leak query equality and the bucketized query response volume. For document retrieval, the query leakage can be viewed as a reduced and (pseudo)random version of the naive access pattern leakage. To see why DDR-SSE has a reduced leakage, consider an SSE scheme with naive access pattern leakage. If the first query retrieves documents d_0,d_1,d_2 and the second query retrieves d_1,d_2,d_3, the access pattern leakage preserves the information that both queries retrieve d_1 and d_2. Ignore the randomization of document identifiers in DDR-SSE for a bit. In DDR-SSE, we store two copies of each document and split the retrieval of the documents between the two copies. If we write the first copy of the i-th document as d_i and the second copy as d'_i, we may retrieve d_0,d'_1,d'_2 in the first query ($r=1,b=0$ in Algorithm 2) and d_1,d_2,d'_3 in the second query ($r=2,b=0$ in Algorithm 2). In this case, the access pattern no longer contains the information that the two queries have two matching documents in common. Of course, since the query algorithm of DDR-SSE uses pseudorandom numbers (r and b in Algorithm 2), the query leakage is (pseudo-)randomized.

Finally, the server does not learn the actual document identifiers in DDR-SSE. This is because our design uses a response-hiding index-only SSE scheme for index retrieval and the document identifiers in the encrypted document store are masked using a PRF. We also used a PRP to randomize the document identifiers at the very beginning, but the main purpose of the PRP is to give us randomly distributed document identifiers (which is a property required to prove Theorem 3). In any case, we can combine the hiding properties from the PRP and the other mechanisms into a single permutation $\mathtt{Perm}$ in the leakage function (Algorithm 4).

Let $|\mathtt{DB}_{\mathtt{bkt}}(w)|$ denote the query response volume of w on the padded $\mathtt{DB}_{\mathtt{bkt}}$ after bucketization with bucket size $\mathtt{bkt}$. A full description of the query leakage of DDR-SSE can be found in Algorithm 4. The following theorem formalizes the security of DDR-SSE using the standard security notion. The theorem holds for any volume-hiding index-only SSE scheme (before the modification we described in Section 3) that has the size of the inverted search index as the setup leakage, and the maximum query response volume and query equality leakage as the query leakage (which we assumed to be a part

[2] The query response includes noise in DDR-SSE so we are using the weaker version of the correctness definition. It is possible to make DDR-SSE achieve the stronger version of the correctness definition by letting the client store the true query response volumes as well.

Algorithm 4 $\mathcal{L}_{\textbf{Query}}$ of DDR-SSE

procedure $\mathcal{L}_{\textbf{QUERY}}(w, \mathtt{DB}, \mathtt{bkt}; \mathtt{st})$
/ Parse the state as query history (a list of queried keywords), access pattern history (a dictionary with keywords as the keys and lists of document identifiers as values), and a permutation on* $[2 \cdot |DB_{\mathtt{bkt}}|]$ **/*
  $\mathtt{qryHist}, \mathtt{APHist}, \mathtt{Perm} \leftarrow \mathtt{st}$
  if $\mathtt{Perm} = \emptyset$ **then**
    Initialize a random permutation on $[2 \cdot |\mathtt{DB}_{\mathtt{bkt}}|]$ and store it as $\mathtt{Perm}$
  / Query equality leakage */*
  $\mathtt{qeq} \leftarrow \{i \mid w_i \in \mathtt{qryHist} \wedge w_i = w\}$
  $\mathtt{qryHist} \leftarrow \mathtt{qryHist} + [w]$
  / Reduced and randomized access pattern leakage */*
  $\mathtt{AP} \leftarrow \{\}$
  if $w \in \mathtt{APHist}$ **then**
    $\mathtt{AP} \leftarrow \mathtt{APHist}[w]$
  else
    $b \leftarrow\$ \{0,1\}, r \leftarrow\$ \{0, \ldots, |\mathtt{DB}_{\mathtt{bkt}}(w)|\}$
    $\mathtt{AP} \leftarrow \{\mathtt{Perm}(id_i + b \cdot |\mathtt{DB}_{\mathtt{bkt}}|) \mid id_i \in \mathtt{DB}_{\mathtt{bkt}}[w] \wedge i < r\} \cup \{\mathtt{Perm}(id_i + (1-b) \cdot |\mathtt{DB}_{\mathtt{bkt}}|) \mid id_i \in \mathtt{DB}_{\mathtt{bkt}}[w] \wedge i \geq r\}$
    $\mathtt{APHist}[w] \leftarrow \mathtt{AP}$
  $\mathtt{st} \leftarrow \mathtt{qryHist}, \mathtt{APHist}, \mathtt{Perm}$
  return $\mathtt{qeq}, \mathtt{AP}$

of our construction). Due to space constraints, we defer a detailed proof of Theorem 2 to the full version of the paper.

Theorem 2 (System-Wide Security of DDR-SSE)**.** Let the system-wide setup leakage of database $\mathtt{DB}$ with bucket size $\mathtt{bkt}$ be given by $\mathcal{L}_{\textbf{Setup}}(\mathtt{DB}, \mathtt{bkt}) = (|W|, |W\{\mathtt{DB}_{\mathtt{bkt}}\}|, |\mathtt{DB}|)$ and the system-wide query leakage of keyword w on database $\mathtt{DB}$ with bucket size $\mathtt{bkt}$ be $\mathcal{L}_{\textbf{Query}}(w, \mathtt{DB}, \mathtt{bkt}; \mathtt{st})$ defined in Algorithm 4. DDR-SSE is a $(\mathcal{L}_{\textbf{Setup}}, \mathcal{L}_{\textbf{Query}})$-secure end-to-end SSE scheme.

A Note on the Stateful Leakage Function. We argue that the stateful leakage function of DDR-SSE is not worrisome. Specifically, the state of $\mathcal{L}_{\textbf{Query}}$ in DDR-SSE records two pieces of information, namely query equality leakage and the pseudorandom numbers (b and r) for the queries that have been made. Query equality leakage in the literature is typically formulated as a stateless leakage as the leakage function takes *all queries* up to the current query as an input [37]. On the other hand, $\mathcal{L}_{\textbf{Query}}$ only takes the last query as input in our formulation. Hence, we need to keep the previous queries in the state. We also need to keep the pseudorandom numbers for the queries that have been made in the state as repeated queries need to have consistent access pattern leakage. We show why the resulting access pattern leakage is significantly more resilient to leakage-abuse attacks in Section 4.

3.6 Instantiation and Performance Analysis

Instantiation. We now instantiate the volume-hiding index-only SSE scheme with XorMM [39], the state-of-the-art scheme with the smallest server storage. Of course, one may choose a different volume-hiding index-only SSE scheme to optimize for different metrics.

Asymptotic Performance Analysis. Let $\mathtt{DB}$ be a database and $\mathtt{bkt}$ be the bucket size used in the bucketization procedure. Let 1^{λ_0} be the security parameter of the index-only SSE scheme used in DDR-SSE and 1^{λ_1} be the security parameter of F used for the encrypted metadata and the encrypted documents. Let 1^{λ_2} be the security parameter of the encryption scheme used for the encrypted documents. Let l_0 be the length of a ciphertext in the encrypted inverted index, l_1 be the output length of the PRF, and l_2 be the length of an encrypted document. We present the key performance metrics of DDR-SSE in Table 1. A detailed breakdown of them appears in the full version of the paper.

		C/C→S	S/S→C
Storage		$O(\lambda_0 + 2\lambda_1 + \lambda_2)$	$O(3\lvert W\rvert \cdot l_1 + 2\lvert \mathtt{DB}\rvert \cdot (l_1 + l_2) + (1.23\lvert W\{\mathtt{DB}_{\mathtt{bkt}}\}\rvert + \beta) l_0)$
Comm.	Metadata	$O(l_1)$	$O(l_1)$
	Index	$O(\lambda_0)$	$O(\lvert \mathtt{DB}_{\mathtt{bkt}}(w)\rvert \cdot l_0)$
	Documents	$O(\lvert \mathtt{DB}_{\mathtt{bkt}}(w)\rvert \cdot l_1)$	$O(\lvert \mathtt{DB}_{\mathtt{bkt}}(w)\rvert \cdot l_2)$
Computation		$O(\lvert \mathtt{DB}_{\mathtt{bkt}}(w)\rvert)$	$O(\lvert \mathtt{DB}_{\mathtt{bkt}}(w)\rvert)$

Table 1: The key performance metrics of DDR-SSE. C/C→S means client or client to server. S/S→C means server or server to client.

Comparison to SWiSSSE. Compared to SWiSSSE, DDR-SSE requires a smaller client storage (since DDR-SSE does not require a client stash). The communication and computation costs of the two schemes are similar concretely as both are linear in the query response volume.

4 System-Wide Leakage Cryptanalysis of DDR-SSE for Static Databases

In this section, we discuss the leakage of DDR-SSE, and analyze its security via a highly refined query reconstruction attack which we modified from [40] to tailor to the leakage of DDR-SSE. This attack is one of the most refined leakage-abuse attacks in the literature. We also show that DDR-SSE is secure against existing attacks in the literature. We use the Enron email corpus as the target dataset and show that DDR-SSE is secure against this attack for an appropriate choice of bucket size.

4.1 Discussion of the Leakage Profile

Query Equality Leakage. DDR-SSE leaks query equality. This is in line with most of the SSE constructions in the literature. However, unlike other leakage suppression techniques, DDR-SSE ensures that every group of $\mathtt{bkt}$ keywords have the same (padded) query response volume. We show that DDR-SSE is very effective at thwarting attacks such as [22, 25] in Section 4.3.

Volume Leakage. As DDR-SSE uses bucketization, a query only leaks the bucketized query response volume. We show that if the bucket size is chosen carefully, DDR-SSE will remain secure against leakage-abuse attacks while enjoying better search efficiency than worst-case padding.

Access Pattern Leakage. The access pattern leakage of the (non-repeating) queries contains the information which entries of the encrypted database are touched by each query.

Most of the attacks in the literature focus on a reduced version of it known as co-occurrence leakage [17, 18, 20, 21, 45]. Co-occurrence leakage is often represented as a co-occurrence matrix $\mathtt{C}$ where $\mathtt{C}[i,j]$ is equal to the number of documents that are accessed by both the i-th and the j-th query. An attacker is assumed to observe the co-occurrence matrix and try to recover the queries and/or the data with the help of some auxiliary dataset.

For an SSE scheme that leaks the naive co-occurrence pattern (i.e., $\mathtt{C}[i,j]$ equals to the number of documents containing the i-th queried keyword and the j-th queried keyword exactly), the attack described above is very effective. This is not surprising, as $\mathtt{C}$ contains a lot of information. We show next that the co-occurrence matrix $\mathtt{C}$ obtained from DDR-SSE contains significantly less information than the naive access pattern. Specifically, if $\mathtt{C}[i,j] = c, i \neq j$ with the naive access pattern, the corresponding $\mathtt{C}[i,j]$ in DDR-SSE will almost be distributed uniformly between 0 and c. Furthermore, the observed co-occurrence counts are pseudorandom, meaning that running the same queries again will produce the same co-occurrence counts. Thus no attacks are possible based on running more queries and averaging out noise (indeed, there is no noise). DDR-SSE results in a highly obscured co-occurrence leakage that is very effective in defending against leakage-abuse attacks. We state the following theorem. The proof is deferred to Appendix B.1.

Theorem 3 (Distribution of Co-occurrence Counts)**.** Let $\mathtt{DB}$ be a database after bucketization. Let $\mathtt{C} : [k] \times [k] \to \mathbb{N}$ be a co-occurrence matrix for database $\mathtt{DB}$ and queries on (distinct) keywords $w_0, \ldots, w_{k-1}$. Then, the distribution of the observed co-occurrence counts $\mathtt{C}'$ of DDR-SSE, over the randomness from the randomization of the document identifiers, and the randomness in the query procedure (the r's and b's), are:

$$\mathtt{Pr}\left[\mathtt{C}'[i,j] = c\right] =$$

$$\begin{cases} 1 & \text{if } i = j \text{ and } c = \mathtt{C}[i,j], \\ \frac{1}{\mathtt{C}[i,j]+1} + \frac{1-\delta}{(\mathtt{C}[i,j]+1)^2} & \text{if } i \neq j \text{ and } 1 \leq c \leq \mathtt{C}[i,j]-1, \\ \frac{1-\delta}{2(|\mathtt{C}[i,j]|+1)} + \frac{\delta}{\mathtt{C}[i,j]+2} + \frac{1-\delta}{(\mathtt{C}[i,j]+1)^2} + \frac{\delta}{(\mathtt{C}[i,j]+2)(\mathtt{C}[i,j]+1)} & \\ \qquad\qquad \text{if } i \neq j \text{ and } c \in \{0, \mathtt{C}[i,j]\}, & \\ 0 & \text{otherwise,} \end{cases}$$

where $\delta = \frac{(\mathtt{C}[i,i]-\mathtt{C}[i,j])(\mathtt{C}[j,j]-\mathtt{C}[i,j])}{(\mathtt{C}[i,i]+1)(\mathtt{C}[j,j]+1)}$.

One can check that for any $c \in \{0, \ldots, \mathtt{C}[i,j]\}$, $\mathtt{Pr}[\mathtt{C}'[i,j] = c] \leq \frac{1}{\mathtt{C}[i,j]+1} + \frac{1}{(\mathtt{C}[i,j]+1)^2}$. So the probability mass function of $\mathtt{C}'[i,j]$ is almost identical to that of a discrete uniform distribution between 0 and $\mathtt{C}[i,j]$. As a result, the observed co-occurrence counts carry little information about the real co-occurrence counts.

Discussion. A natural alternative to our document retrieval approach is as follows: to retrieve $(id_0, \ldots, id_{k-1})$ for queried keyword w, the client generates pseudorandom bits $(b_0, \ldots, b_{k-1})$ where each bit is pseudorandomly determined by the queried keyword and the index of the document identifier, and retrieves document id_i from copy b_i of the encrypted documents. While this would reduce the leakage of the queries as compared to a scheme that leaks the naive access pattern, the observed co-occurrence counts in the off-diagonal entries would actually be tightly distributed about a value equal to half of the true co-occurrence count. Thus the attacker could still extract useful information from the leakage and exploit it in a leakage-abuse attack. This explains why we adopt a different approach in DDR-SSE, targeting almost uniformly distributed co-occurrence counts.

Variable-length Documents. The construction in Section 3 assumes fixed-length documents. To handle variable-length documents, our implementation splits documents larger than 4KB into 4KB chunks before encryption. In Section 4.2, we present additional leakage cryptanalysis (where we work with variable-length documents directly) to establish that the "additional" leakage from document-splitting is not detrimental to the security of DDR-SSE. See Appendix B.2 for a more detailed discussion.

4.2 System-Wide Leakage-Abuse Attacks

Attack Assumptions. Let $\mathtt{DB}$ be a database (after large documents are split into smaller chunks) and $\mathtt{bkt}$ be the bucket size used in the bucketization procedure. Let $w_0, \ldots, w_{k-1}$ be a sequence of keyword queries on $\mathtt{DB}$. We assume that the attacker has access to the database $\mathtt{DB}$, the bucket size $\mathtt{bkt}$, a transcript of the queries $w_0, \ldots, w_{k-1}$ on the database, and the queried keywords as a set $\{w_0, \ldots, w_{k-1}\}$. The goal of the attacker is to recover the queried keywords $w_0, \ldots, w_{k-1}$.

Note that the attack assumptions made here are very strong. In particular, it is not clear how the attacker could obtain the original database $\mathtt{DB}$ (a typical leakage-abuse attack uses an auxiliary database that is statistically close to $\mathtt{DB}$) or the set of keywords (as the universe of keywords is much larger). Needless to say, if DDR-SSE is secure under these attack assumptions, DDR-SSE will also be secure under more realistic attack assumptions.

Attack Details. Our attack follows the blueprint of [40], i.e. we identify the *likelihood function* of an assignment (between the queries and the keywords) given the observed leakage and auxiliary information. We then run simulated annealing with the likelihood function to identify the most likely assignment. Note that even though the attacker has access to the true co-occurrence counts in our attack setting, they are unable to run the count attack [18] due to the large amount of noise in the co-occurrence counts. In the interest of space, we refer to [40] for the technical details of the leakage-abuse attack.

Cryptanalysis Results on the Enron Email Corpus. We run the likelihood-based simulated annealing attack described in [40] against the Enron email corpus. We pre-process the

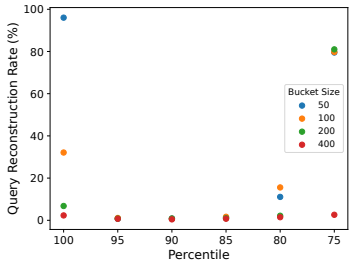


Figure 2: Query reconstruction rates of our attack against DDR-SSE on the Enron email corpus. x-th percentile (the x-axis) means the queries in the attack are chosen in the x-th percentile. 100%-th percentile means the most frequent keyword. The query reconstruction rate is measured as the percentage of queries correctly recovered by our attack.

emails to obtain 400K variable-length documents with keywords associated to them. We try to compress the documents with GZIP. If the compressed document is larger than 4KB in size, we split the document into smaller chunks such that all chunks (treated as separate documents) have compressed size less than 4KB. Each chunk inherits all keywords of the original document. We then pad all documents to 4KB length. After that, we build a plaintext inverted index on these documents. We run the setup of DDR-SSE on these documents and the inverted index with bucket sizes $\texttt{bkt} = 50, 100, 200$ and 400. For each bucket size, we make 1,200 queries where the queried keywords are chosen from the $\texttt{pct} = 100, 95, 90, 85, 80$ and 75-th percentile frequencies respectively. Here, the 100-th percentile frequency corresponds to the most frequent keywords. From each experiment, we extract the leakage (from our implementation of DDR-SSE; the leakage is identical to the one described in Section 3.5) and run the attack on it. The process above is repeated 10 times and the average query reconstruction rate for each bkt and pct is reported in Figure 2.

It is apparent from the experiments that $\texttt{bkt} = 50$ or 100 are not sufficient to protect against query reconstruction attacks. For $\texttt{bkt} = 200$ and 400, and pct between 80 and 100, the query reconstruction rate is well below 20% and DDR-SSE should be deemed secure. Interestingly, the query reconstruction rate of our attack increases for keywords in the 75-th percentile. This may be counter-intuitive as the keywords in those ranges have much lower query response volumes, and thus, contain less information. We attribute the higher query reconstruction rate to the attack assumptions we make. Specifically, we assume that the attacker has access to the *exact* (plaintext) database that it tries to recover the queries from. This means that the attacker has access to the exact co-occurrence counts. In the preprocessing step of the attack, we eliminate potential keyword candidates for each query by checking if all observed co-occurrence counts (which are uniformly distributed between 0 and the actual co-occurrence count) are smaller than or equal to the auxiliary co-occurrence counts. For keywords with smaller query response volumes, the co-occurrence counts are also smaller. This makes obfuscation of the larger co-occurrence counts among the smaller ones harder, leading to a better query reconstruction rate.

We stress that the phenomenon above only occurs because the attacker knows the *exact* database. In a more realistic attack setting where the attacker only has access to an auxiliary database that is statistically close to the target database, query reconstruction will be harder for keyword at around the 75-th percentile. We demonstrate this with additional experiments in the full version. Finally, we refer to Appendix B.3 for a detailed discussion on security versus efficiency trade-offs for bucketization, and to Appendix B.4 for a comparison with the leakage of SWiSSSE [42].

4.3 The Other State-of-the-Art Attacks

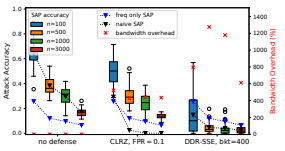
In addition to our own query reconstruction attack, we modify the SAP attack [22], the IHOP attack [25] and the Jigsaw attack [46] to DDR-SSE, and show that it is resilient against these attacks as well. We also report the attack accuracies of these attacks against a scheme with no defence and the (system-wide) SSE scheme by Chen et al. [41] (known as CLRZ) for reference.

Description of the Attacks. SAP [22] is an attack that exploits search pattern and volume leakages. It is significantly more effective than an attack that exploits the search pattern leakage or the volume leakage by itself. IHOP [25] is an attack that exploits search pattern and access pattern leakages. It was claimed in [25] to beat all other attacks in the literature at the time. Jigsaw [46] improves on IHOP by exploiting the query equality leakage more effectively. Jigsaw is able to achieve impressive query recovery rates in settings where IHOP struggles.

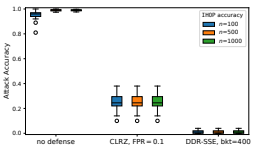
The Attacks Against Countermeasures. The attacks are designed for naive SSE schemes which leak the naive search pattern and volume (or access pattern). The authors adapt their attacks to schemes with countermeasures by modifying the auxiliary volume (or co-occurrence) information; see Equation (19) in [22], Equation (9) in [25], and Equation (8) in [46] for examples.

We apply the same approach to make these attacks compatible with DDR-SSE. Specifically, we apply bucketization to the auxiliary database before generating the auxiliary volume and co-occurrence information. This means that for the auxiliary volume information and the diagonal entries of the co-occurrence information, we expect groups of bkt entries to have the same value. We further modify the auxiliary co-occurrence information by setting the off-diagonal entries to their expected values (i.e., half of the original values).

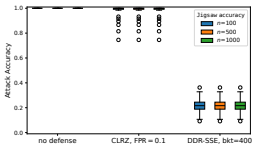
Cryptanalysis Results with the Other State-of-the-Art Attacks. We run SAP, IHOP, and Jigsaw with their respective default parameters (see [22, Sect. 6], [25, Sect. 5], [46, Sect. 5]), and report the query reconstruction rates of these attacks against (1) no defence, (2) CLRZ with $\mathsf{TPR} = 0.999$ and



(a) The SAP attack results.



(b) The IHOP attack results.



(c) The Jigsaw attack results.

Figure 3: Cryptanalysis results with the state-of-the-art attacks.

$\mathsf{FPR} = 0.1$, and (3) DDR-SSEon the Enron email corpus (the dataset used by these attacks contains fewer emails than the one we used in Section 4.2; we decide to not change the dataset so that the experimental results will be consistent with the original papers). The cryptanalysis results are shown in Figure 3. We see that SAP and IHOP achieve less than 10% attack accuracy, whereas Jigsaw achieve 20%. These are significantly lower than the attack accuracies on CLRZ, demonstrating the effectiveness of leakage suppression of DDR-SSE.

Discussion. The cryptanalysis results in Sections 4.2 and 4.3 make it evident that the uniformization of the co-occurrence counts is very effective at thwarting leakage-abuse attacks.

5 Performance Evaluation of DDR-SSE

5.1 Experimental Setup

Choice of Primitives. We use HMAC-SHA-256 for the PRFs and AES-128-CBC for symmetric-key encryption.

Implementation. We make a single-thread implementation of the client and server in Java, using the Java Cryptography Extension as the underlying cryptographic library. The client and server are implemented in a single program so there is no network element in the implementation (and the experiments). We made this choice as the network latency is not the main source of overhead and implementing DDR-SSE without the network layer is a lot simpler. We also did not use a commercial database management system (e.g., Redis) in our implementation as XorMM is incompatible with that. We use a single-thread implementation as it provides the most accurate performance measurements.

Document Preprocessing. We use the Natural Language Toolkit[3] to extract keywords from emails. English stop words and keywords with frequency higher than 5% are removed from the set of keywords for each email. Emails are compressed with gzip. If the compressed email is larger than 4KB, it is split into smaller chunks (each chunk is treated as a separate email that inherits all keywords from the original email) such that the compressed chunks have size less than 4KB. In the baseline benchmark, we pad all emails to 4KB. To investigate the scalability of DDR-SSE on larger documents, we also pad the emails to 8KB and 16KB respectively. We note that this choice allows us to use the same inverted index for all experiments, so that any difference in the performance we measure must come from the emails themselves.

Bucket Size. We use $\mathtt{bkt} = 400$ in our experiment.

Environment. We run our experiments on an AMD EPYC 7543P 32-Core Processor with 950 GB of RAM.

5.2 Benchmarks

Setup Time. If we split the emails into 4KB chunks, the setup of DDR-SSE takes 447 seconds. Most of the setup time (430 seconds) is spent on the setup of XorMM. This is expected as the setup of XorMM involves the construction of an XOR filter of size 2^{25} (there are 29M keyword-document pairs in the database). On the other hand, the setup of the metadata and the encrypted documents only took 0.3 seconds If we split the emails into 8KB or 16KB chunks instead, the setup of DDR-SSE takes 448 seconds and 502 seconds, respectively. One may expect the setup time to grow linearly with the chunk size. However, this is not the case for DDR-SSE. This is because the number of keywords does not change with the chunk size, so the setup time of the encrypted metadata remains a constant. The number of keyword-document pairs does not change significantly with the chunk size, so the setup of XorMM does not grow with the chunk size. However, the setup of XorMM involves trial insertions in a hash table, so the setup time is not a constant. Lastly, the setup time of the encrypted documents does grow linearly with the chunk size, as a larger chunk size implies more data to be encrypted. DDR-SSE took 16 seconds, 56 seconds, and 62 seconds to build the encrypted documents with chunk size 4KB, 8KB, and 16KB respectively.

Server Storage. The experimental data contains 20M keyword-document pairs and 400K documents (before chunking). The documents are compressed, split into 4KB/8KB/16KB chunks, and padded to 4KB/8KB/16KB before encryption. The encrypted database consists of encrypted metadata of size 1MB, an encrypted search index of size 1.3GB, and encrypted documents of size 3.1GB/6.2GB/12.5GB in total. The overall server storage is 4.4GB/7.5GB/13.8GB.

Query Response Time. The query response time of DDR-SSE for various document sizes is shown in Figure 4. Here, query response volume refers to the query response volume before bucketization and query response time is defined to be the time from the start of a query to the point of time for which the client obtains the plaintext documents. DDR-SSE has an

[3] https://www.nltk.org/

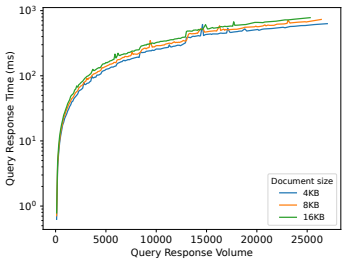


Figure 4: Query response time of the DDR-SSE on 400K documents and document size 4KB, 8KB, and 16KB (log scale).

end-to-end query throughput of 39K documents per second, 35K documents per second and 31K documents per second for documents of size 4KB, 8KB, and 16KB respectively. DDR-SSE should be considered practical with these numbers. One may expect a lower query throughput for larger document size, but this is not the case because the bottleneck of the search queries is the retrieval of the documents (using pseudorandom search tokens), not their decryption.

Comparison to SWiSSSE. We give a quantitative comparison between SWiSSSE and DDR-SSE. Assuming 400K documents and the documents are split into 4KB chunks, the server storage of SWiSSSE is 3.6GB, and the server storage of DDR-SSE is 4.4GB. The client storage of SWiSSSE varies (due to the use of the stash) and can be up to 10MB in size. On the other hand, the client storage of DDR-SSE is 64B.

The next few performance numbers should be taken with a grain of salt, as the server of SWiSSSE is implemented using Java as an interface and Redis as the actual storage, while the server of DDR-SSE is implemented using Java directly. It is not natural to use Redis as the actual server for DDR-SSE, as the encrypted data structures (i.e., arrays) are not Redis-friendly (which expects key-value stores). Part of the overhead of SWiSSSE comes from the use of Redis. The setup of SWiSSSE takes 170 seconds, whereas the setup of DDR-SSE takes 447 seconds. SWiSSSE has a search throughput of 8.4K documents per second, whereas DDR-SSE has a search throughput of 39K documents per second. In addition to the "normal" queries, SWiSSSE also requires auxiliary write-backs after each query. These auxiliary write-backs typically take about 1 second to complete.

Comparison to ORAM and PIR. DDR-SSE is significantly more efficient compared to other, more secure primitives that can be used for document retrieval, such as ORAM or PIR. EnigMap [47] is the state-of-the-art ORAM-based encrypted key-value store. Each document in EnigMap is only 0.5KB in size. On a database with 100K documents (much smaller than the 400K documents in our experiments), each query (that retrieves one document) takes about 128 microseconds. In other words, the search throughput of EnigMap (for document retrieval only) is around 8K documents per second on 0.5KB documents, or one order of magnitude slower than DDR-SSE. If EnigMap were instantiated with larger documents, its performance would drop significantly. This is because EnigMap is optimised for page efficiency, and larger documents means fewer documents can be fitted into a page and more page swaps are required to run a query in EnigMap.

On the other hand, Spiral Batch PIR [15,48], a state-of-the-art PIR scheme, takes 846 seconds to retrieve 512 documents of size 8KB each and 1,290 seconds to retrieve 512 documents of size 16KB each (the experimental results appear in [49]). In other words, Spiral Batch PIR has query throughput (for document retrieval only) of 0.61 documents per second on 8KB documents and 0.40 documents per second on 16KB documents. Hence Spiral Batch PIR is 4-5 orders of magnitude slower than DDR-SSE.

6 Extensions and Future Work

Multi-server DDR-SSE. It is surprisingly easy to convert DDR-SSE into a multi-server scheme with even less leakage, under the assumption that servers do not collude. For the encrypted documents, we create two encrypted documents for each real document (with document identifier id) and store them in addresses $F(sk_F, id \,\|\, 0)$ and $F(sk_F, id \,\|\, 1)$. We then set up two servers, and store all encrypted documents with addresses of the form $F(sk_F, id \,\|\, 0)$ at the first server and store all encrypted documents with addresses of the form $F(sk_F, id \,\|\, 1)$ at the second server. Under the non-collusion assumption, each server only observes part of the overall access pattern leakage. Finally, we also use a multi-server encrypted inverted index such as DORY [50]. We leave the detailed leakage analysis as future work.

Dynamic DDR-SSE. A natural question to ask is whether DDR-SSE can be made *dynamic*. Assuming an upper bound on the number of keywords and documents supported by the encrypted database, a similar strategy to SWiSSSE [42] can be used to achieve dynamic DDR-SSE. We briefly sketch the idea here. The encrypted inverted index and the encrypted documents initially contain dummy entries. An insertion query replaces a dummy entry in the encrypted inverted index and potentially a dummy encrypted document. A deletion query turns a real index entry into a dummy one, and potentially turns a real encrypted document into a dummy one. Search queries on a particular keyword produce the same access pattern regardless of the corresponding update queries.

However, dynamic DDR-SSE for arbitrary updates is challenging. This is because volume-hiding and dynamism are at odds with each other. A naïve volume-hiding technique is to pad each query response volume to the maximum query response volume. This strategy does not work for dynamic SSE since the maximum query response volume can change. Most dynamic (index-only) SSE schemes [32,35,51,52] are *not* volume-hiding. Dynamic SSE schemes that are volume-hiding [10,38,53] are index-only schemes and do not scale

well for document retrieval. We leave dynamic DDR-SSE for arbitrary updates as future work.

Ethical Considerations

In this paper, we proposed a new privacy-enhancing cryptographic scheme, namely DDR-SSE, that enables efficient keyword searches over encrypted document collections while ensuring system-wide security. We established the resilience of DDR-SSE to highly refined query reconstruction attacks through leakage cryptanalysis experiments (described in Section 4), which use the Enron email corpus. We distinguish broadly between two groups of stakeholders potentially affected by our research: stakeholders that are impacted specifically by our leakage cryptanalysis methodology, and stakeholders that are impacted more broadly by the technology researched and developed in this paper. We analyse the potential harms and benefits for these two groups using a consequentialist approach.

The natural set of stakeholders potentially impacted by our leakage cryptanalysis study is the set of individuals named (or identified otherwise) in the Enron email dataset. We argue that it is justifiable to use the Enron email corpus for leakage cryptanalysis given: (a) the lack of alternative realistic datasets for conducting attack experiments, (b) this dataset has been used extensively for attack experiments in previous works [17, 21, 22, 25], (c) the corpus has been public for a long time, (d) researchers have attempted to curate the dataset to remove data concerning individuals upon request,[4] (e) individuals could request to have their data redacted if they so wished at any point of time, (f) no specific elements of the dataset are reported in our paper, and (g) our results depend only on statistical properties of the dataset.

Our research enables secure search over outsourced, encrypted databases and improves over the state-of-the-art in the sense that our proposed scheme DDR-SSE is: (a) either substantially more efficient than known techniques such as ORAM [5–11] or PIR [12–16] while still achieving security against highly refined system-wide leakage cryptanalysis attacks or (b) has a much simpler leakage profile than system-wide secure SSE schemes with comparable security and efficiency, such as SWiSSSE [42]. The main group impacted by our research are potential users of secure encrypted computation. Increased efficiency broadens the availability of encrypted computation, especially to economically marginalized groups. A simpler leakage profile allows for an easier security analysis, which improves the confidence of users in the security of the scheme, and significantly reduces the possibility of undetected attacks that could otherwise compromise the privacy of honest users. Overall, we believe that our proposal has the potential to enable increased adoption of encrypted document search, thus benefiting society at large from the point of view of privacy of outsourced data.

[4] See https://www.cs.cmu.edu/~enron/ for detailed documentation.

Open Science

The implementations of DDR-SSE and the cryptanalysis attack against DDR-SSE are all publicly available: https://doi.org/10.5281/zenodo.17965794.

References

[1] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *2000 IEEE Symposium on Security and Privacy*, pp. 44–55, IEEE Computer Society Press, May 2000.

[2] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *ACM CCS 2006* (A. Juels, R. N. Wright, and S. De Capitani di Vimercati, eds.), pp. 79–88, ACM Press, Oct. / Nov. 2006.

[3] M. Chase and S. Kamara, "Structured encryption and controlled disclosure," in *ASIACRYPT 2010* (M. Abe, ed.), vol. 6477 of *LNCS*, pp. 577–594, Springer, Berlin, Heidelberg, Dec. 2010.

[4] D. Cash, S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for Boolean queries," in *CRYPTO 2013, Part I* (R. Canetti and J. A. Garay, eds.), vol. 8042 of *LNCS*, pp. 353–373, Springer, Berlin, Heidelberg, Aug. 2013.

[5] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious RAMs," *Journal of the ACM*, vol. 43, no. 3, pp. 431–473, 1996.

[6] E. Shi, T.-H. H. Chan, E. Stefanov, and M. Li, "Oblivious RAM with $O((\log N)^3)$ worst-case cost," in *ASIACRYPT 2011* (D. H. Lee and X. Wang, eds.), vol. 7073 of *LNCS*, pp. 197–214, Springer, Berlin, Heidelberg, Dec. 2011.

[7] E. Stefanov, M. van Dijk, E. Shi, C. W. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path ORAM: an extremely simple oblivious RAM protocol," in *ACM CCS 2013* (A.-R. Sadeghi, V. D. Gligor, and M. Yung, eds.), pp. 299–310, ACM Press, Nov. 2013.

[8] S. Garg, P. Mohassel, and C. Papamanthou, "TWORAM: Efficient oblivious RAM in two rounds with applications to searchable encryption," in *CRYPTO 2016, Part III* (M. Robshaw and J. Katz, eds.), vol. 9816 of *LNCS*, pp. 563–592, Springer, Berlin, Heidelberg, Aug. 2016.

[9] J. G. Chamani, D. Papadopoulos, C. Papamanthou, and R. Jalili, "New constructions for forward and backward private symmetric searchable encryption," in *ACM CCS 2018* (D. Lie, M. Mannan, M. Backes, and X. Wang, eds.), pp. 1038–1055, ACM Press, Oct. 2018.

[10] I. Demertzis, D. Papadopoulos, C. Papamanthou, and S. Shintre, "SEAL: Attack mitigation for encrypted databases via adjustable leakage," in *USENIX Security 2020* (S. Capkun and F. Roesner, eds.), pp. 2433–2450, USENIX Association, Aug. 2020.

[11] L. Assouline and B. Minaud, "Weighted oblivious RAM, with applications to searchable symmetric encryption," in *EUROCRYPT 2023, Part I* (C. Hazay and M. Stam, eds.), vol. 14004 of *LNCS*, pp. 426–455, Springer, Cham, Apr. 2023.

[12] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private information retrieval," in *36th FOCS*, pp. 41–50, IEEE Computer Society Press, Oct. 1995.

[13] E. Kushilevitz and R. Ostrovsky, "Replication is NOT needed: SINGLE database, computationally-private information retrieval," in *38th FOCS*, pp. 364–373, IEEE Computer Society Press, Oct. 1997.

[14] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai, "Batch codes and their applications," in *36th ACM STOC* (L. Babai, ed.), pp. 262–271, ACM Press, June 2004.

[15] S. J. Menon and D. J. Wu, "SPIRAL: Fast, high-rate single-server PIR via FHE composition," in *2022 IEEE Symposium on Security and Privacy*, pp. 930–947, IEEE Computer Society Press, May 2022.

[16] M. H. Mughees and L. Ren, "Vectorized batch private information retrieval," in *2023 IEEE Symposium on Security and Privacy*, pp. 437–452, IEEE Computer Society Press, May 2023.

[17] M. S. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation," in *NDSS 2012*, The Internet Society, Feb. 2012.

[18] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-abuse attacks against searchable encryption," in *ACM CCS 2015* (I. Ray, N. Li, and C. Kruegel, eds.), pp. 668–679, ACM Press, Oct. 2015.

[19] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-abuse attacks against searchable encryption." Cryptology ePrint Archive, Report 2016/718, 2016.

[20] D. Pouliot and C. V. Wright, "The shadow nemesis: Inference attacks on efficiently deployable, efficiently searchable encryption," in *ACM CCS 2016* (E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, eds.), pp. 1341–1352, ACM Press, Oct. 2016.

[21] L. Blackstone, S. Kamara, and T. Moataz, "Revisiting leakage abuse attacks," in *NDSS 2020*, The Internet Society, Feb. 2020.

[22] S. Oya and F. Kerschbaum, "Hiding the access pattern is not enough: Exploiting search pattern leakage in searchable encryption," in *USENIX Security 2021* (M. Bailey and R. Greenstadt, eds.), pp. 127–142, USENIX Association, Aug. 2021.

[23] M. Damie, F. Hahn, and A. Peter, "A highly accurate query-recovery attack against searchable encryption using non-indexed documents," in *USENIX Security 2021* (M. Bailey and R. Greenstadt, eds.), pp. 143–160, USENIX Association, Aug. 2021.

[24] S. Kamara, A. Kati, T. Moataz, T. Schneider, A. Treiber, and M. Yonli, "Cryptanalysis of encrypted search with LEAKER - A framework for LEakage AttacK evaluation on real-world data." Cryptology ePrint Archive, Report 2021/1035, 2021.

[25] S. Oya and F. Kerschbaum, "IHOP: Improved statistical query recovery against searchable symmetric encryption through quadratic optimization," in *USENIX Security 2022* (K. R. B. Butler and K. Thomas, eds.), pp. 2407–2424, USENIX Association, Aug. 2022.

[26] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *ACM CCS 2012* (T. Yu, G. Danezis, and V. D. Gligor, eds.), pp. 965–976, ACM Press, Oct. 2012.

[27] D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner, "Dynamic searchable encryption in very-large databases: Data structures and implementation," in *NDSS 2014*, The Internet Society, Feb. 2014.

[28] E. Stefanov, C. Papamanthou, and E. Shi, "Practical dynamic searchable encryption with small leakage," in *NDSS 2014*, The Internet Society, Feb. 2014.

[29] M. Naveed, M. Prabhakaran, and C. A. Gunter, "Dynamic searchable encryption via blind storage," in *2014 IEEE Symposium on Security and Privacy*, pp. 639–654, IEEE Computer Society Press, May 2014.

[30] S. Faber, S. Jarecki, H. Krawczyk, Q. Nguyen, M.-C. Rosu, and M. Steiner, "Rich queries on encrypted data: Beyond exact matches," in *ESORICS 2015, Part II* (G. Pernul, P. Y. A. Ryan, and E. R. Weippl, eds.), vol. 9327 of *LNCS*, pp. 123–145, Springer, Cham, Sept. 2015.

[31] R. Bost, "$\Sigma o\phi o\varsigma$: Forward secure searchable encryption," in *ACM CCS 2016* (E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, eds.), pp. 1143–1154, ACM Press, Oct. 2016.

[32] R. Bost, B. Minaud, and O. Ohrimenko, "Forward and backward private searchable encryption from constrained cryptographic primitives," in *ACM CCS 2017* (B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, eds.), pp. 1465–1482, ACM Press, Oct. / Nov. 2017.

[33] S. Kamara and T. Moataz, "Boolean searchable symmetric encryption with worst-case sub-linear complexity," in *EUROCRYPT 2017, Part III* (J.-S. Coron and J. B. Nielsen, eds.), vol. 10212 of *LNCS*, pp. 94–124, Springer, Cham, Apr. / May 2017.

[34] S. Kamara and T. Moataz, "SQL on structurally-encrypted databases," in *ASIACRYPT 2018, Part I* (T. Peyrin and S. Galbraith, eds.), vol. 11272 of *LNCS*, pp. 149–180, Springer, Cham, Dec. 2018.

[35] S.-F. Sun, R. Steinfeld, S. Lai, X. Yuan, A. Sakzad, J. K. Liu, S. Nepal, and D. Gu, "Practical non-interactive searchable encryption with forward and backward privacy," in *NDSS 2021*, The Internet Society, Feb. 2021.

[36] S. Kamara and T. Moataz, "Computationally volume-hiding structured encryption," in *EUROCRYPT 2019, Part II* (Y. Ishai and V. Rijmen, eds.), vol. 11477 of *LNCS*, pp. 183–213, Springer, Cham, May 2019.

[37] S. Patel, G. Persiano, K. Yeo, and M. Yung, "Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing," in *ACM CCS 2019* (L. Cavallaro, J. Kinder, X. Wang, and J. Katz, eds.), pp. 79–93, ACM Press, Nov. 2019.

[38] M. George, S. Kamara, and T. Moataz, "Structured encryption and dynamic leakage suppression," in *EUROCRYPT 2021, Part III* (A. Canteaut and F.-X. Standaert, eds.), vol. 12698 of *LNCS*, pp. 370–396, Springer, Cham, Oct. 2021.

[39] J. Wang, S.-F. Sun, T. Li, S. Qi, and X. Chen, "Practical volume-hiding encrypted multi-maps with optimal overhead and beyond," in *ACM CCS 2022* (H. Yin, A. Stavrou, C. Cremers, and E. Shi, eds.), pp. 2825–2839, ACM Press, Nov. 2022.

[40] Z. Gui, K. G. Paterson, and S. Patranabis, "Rethinking searchable symmetric encryption," in *2023 IEEE Symposium on Security and Privacy*, pp. 1401–1418, IEEE Computer Society Press, May 2023.

[41] G. Chen, T.-H. Lai, M. K. Reiter, and Y. Zhang, "Differentially private access patterns for searchable symmetric encryption," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pp. 810–818, 2018.

[42] Z. Gui, K. G. Paterson, S. Patranabis, and B. Warinschi, "SWiSSSE: System-wide security for searchable symmetric encryption," *PoPETs*, vol. 2024, pp. 549–581, Jan. 2024.

[43] Z. Gui, K. G. Paterson, S. Patranabis, and B. Warinschi, "SWiSSSE: System-wide security for searchable symmetric encryption." Cryptology ePrint Archive, Report 2020/1328, 2020.

[44] P. Grubbs, A. Khandelwal, M.-S. Lacharité, L. Brown, L. Li, R. Agarwal, and T. Ristenpart, "Pancake: Frequency smoothing for encrypted data stores," in *USENIX Security 2020* (S. Capkun and F. Roesner, eds.), pp. 2451–2468, USENIX Association, Aug. 2020.

[45] M. Naveed, S. Kamara, and C. V. Wright, "Inference attacks on property-preserving encrypted databases," in *ACM CCS 2015* (I. Ray, N. Li, and C. Kruegel, eds.), pp. 644–655, ACM Press, Oct. 2015.

[46] H. Nie, W. Wang, P. Xu, X. Zhang, L. T. Yang, and K. Liang, "Query recovery from easy to hard: Jigsaw attack against SSE," in *USENIX Security 2024* (D. Balzarotti and W. Xu, eds.), USENIX Association, Aug. 2024.

[47] A. Tinoco, S. Gao, and E. Shi, "EnigMap: External-memory oblivious map for secure enclaves," in *USENIX Security 2023* (J. A. Calandrino and C. Troncoso, eds.), pp. 4033–4050, USENIX Association, Aug. 2023.

[48] S. Angel, H. Chen, K. Laine, and S. T. V. Setty, "PIR with compressed queries and amortized query processing," in *2018 IEEE Symposium on Security and Privacy*, pp. 962–979, IEEE Computer Society Press, May 2018.

[49] A. Bienstock, S. Patel, J. Y. Seo, and K. Yeo, "Batch PIR and labeled PSI with oblivious ciphertext compression," in *USENIX Security 2024* (D. Balzarotti and W. Xu, eds.), USENIX Association, Aug. 2024.

[50] E. Dauterman, E. Feng, E. Luo, R. A. Popa, and I. Stoica, "DORY: An encrypted search system with distributed trust," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pp. 1101–1119, USENIX Association, Nov. 2020.

[51] S. Sun, X. Yuan, J. K. Liu, R. Steinfeld, A. Sakzad, V. Vo, and S. Nepal, "Practical backward-secure searchable encryption from symmetric puncturable encryption," in *ACM CCS 2018* (D. Lie, M. Mannan, M. Backes, and X. Wang, eds.), pp. 763–780, ACM Press, Oct. 2018.

[52] I. Demertzis, J. G. Chamani, D. Papadopoulos, and C. Papamanthou, "Dynamic searchable encryption with small client storage," in *NDSS 2020*, The Internet Society, Feb. 2020.

[53] G. Amjad, S. Patel, G. Persiano, K. Yeo, and M. Yung, "Dynamic volume-hiding encrypted multi-maps with applications to searchable encryption," *PoPETs*, vol. 2023, pp. 417–436, Jan. 2023.

[54] K. G. Larsen and J. B. Nielsen, "Yes, there is an oblivious RAM lower bound!," in *CRYPTO 2018, Part II* (H. Shacham and A. Boldyreva, eds.), vol. 10992 of *LNCS*, pp. 523–542, Springer, Cham, Aug. 2018.

A Additional Related Work

ORAM. ORAM allows realization of end-to-end SSE with little (access pattern) leakage. Despite a long line of works on realizing efficient (multi-server) ORAMs [5–7] and ORAM-based SSE [8–11], the bandwidth lower bound for ORAMs (per document retrieval) is logarithmic in the number of documents [54], and typical ORAM schemes require one round of interaction per document retrieved, leading to a linear (in the total number of documents being retrieved) round complexity and high latency. In contrast, searches in DDR-SSE require only two rounds of interaction with a significantly lower bandwidth requirement.

PIR. PIR [12–16] can be used to realize end-to-end SSE with no access pattern leakage. However, (single-server, computational) PIR schemes need to access *all* documents per document retrieval. On the other hand, the number of accessed documents during a search operation in DDR-SSE is linear only in the size of the result set (typically much smaller than the total number of documents). Consequently, DDR-SSE is significantly more efficient than PIR-based schemes, at the cost of incurring some leakage.

B System-Wide Leakage Cryptanalysis: Additional Proofs, Results, and Discussion

B.1 Proof of Theorem 3

We begin with a more formal description of the theorem and a proof sketch.

Without loss of generality, we focus on the queries of the keywords w_i and w_j for some $i \neq j$.

Let $\Pi : [|\mathtt{DB}|] \to [|\mathtt{DB}|]$ be a random permutation applied to the document identifiers. We write $A := \{id_0, \ldots, id_{|A|-1}\}$ and $B := \left\{id'_0, \ldots, id'_{|B|-1}\right\}$ for the sets of document identifiers matching the query of w_i and w_j, respectively, after applying the permutation Π to the document identifiers. Further, we denote their intersection by $I := A \cap B$.

In our analysis we use the following assumption.

Assumption 1. *Given $|A|, |B|$ and $|I|$, we assume for our analysis that A and B are subsets of our database $\mathtt{DB}$ sampled uniformly at random from all possible subsets of the given cardinality that intersect in $|I|$ elements.*

Note that Assumption 1 is equivalent to the sets of identifiers matching the first and second keyword being randomly distributed among all sets intersecting in $|I|$ elements. This is because the bijection Π merely rearranges elements of $\mathtt{DB}$ and thus only shuffles the corresponding subsets of $\mathtt{DB}$.

Let r_0, r_1 be integers sampled uniformly at random from $[0, \ldots, |A|]$ and $[0, \ldots, |B|]$, resp., and let b_0, b_1 be random bits.

We define the observed co-occurrence count $\mathtt{C}'[i, j]$ for w_i and w_j as a random variable where the sample space Ω is defined over all possible values of $(\Pi, r_0, r_1, b_0, b_1)$, and $\mathtt{C}'[i, j]$ maps a realization of $(\Pi, r_0, r_1, b_0, b_1)$ to

$$\mathtt{C}'[i,j]((\Pi, r_0, r_1, b_0, b_1)) := \begin{cases} \left|\{id_0, \ldots, id_{r_0-1}\} \cap \left\{id'_0, \ldots, id'_{r_1-1}\right\}\right| \\ + \left|\left\{id_{r_0}, \ldots, id_{|A|-1}\right\} \cap \left\{id'_{r_1}, \ldots, id'_{|B|-1}\right\}\right|, & \text{if } b_0 = b_1 \\ \left|\{id_0, \ldots, id_{r_0-1}\} \cap \left\{id'_{r_1}, \ldots, id'_{|B|-1}\right\}\right| \\ + \left|\left\{id_{r_0}, \ldots, id_{|A|-1}\right\} \cap \left\{id'_0, \ldots, id'_{r_1-1}\right\}\right|, & \text{if } b_0 \neq b_1 \end{cases}$$

Clearly, the definition of $\mathtt{C}'[i, j]$ above resembles closely our construction, except that we replaced the PRP used to randomize the document identifiers with a random permutation.

In the proof that we will present shortly, we show that it is sufficient to focus on the document identifiers in the intersection instead of all the document identifiers. To explain why this is the case, we introduce two additional random variables s_0 and s_1 with outcome space $\{0, \ldots, |I|\}$.

For an arbitrary but fixed permutation Π (used to define the sets A, B and I), and a realization r_0, we define s_0 to be the outcome $|\{id_0, \ldots, id_{r_0-1}\} \cap I|$. Analogously, given a realization of r_1, we define s_1 to be the outcome $\left|\left\{id'_0, \ldots, id'_{r_1-1}\right\} \cap I\right|$.

In other words, s_0 is the number of documents matching both keywords, restricted to the first r_0 documents matching the first keyword after the randomization of the document identifiers (if $r_0 = 0$, then the number of documents is 0). Similarly, s_1 is the number of documents matching both keywords retrieved by restricting to the first r_1 documents (after randomization by Π) matching the second keyword (if $r_1 = 0$, then the number of documents is 0).

Given s_0 and s_1, we can rewrite $\mathtt{C}'[i, j]$ as follows.

$$\mathtt{C}'[i,j]((\Pi, r_0, r_1, b_0, b_1)) = \begin{cases} |I| - |s_0 - s_1|, & \text{if } b_0 = b_1 \\ |s_0 - s_1|, & \text{if } b_0 \neq b_1. \end{cases} \tag{1}$$

In Lemma 1, we show that s_0 and s_1 are uniformly distributed. Then, we give their joint distribution in Lemma 2 which we will use to derive the distribution of $\mathtt{C}'[i, j]$.

Lemma 1. *The random variables s_0, s_1 are uniformly distributed, i.e. $s_0, s_1 \sim U(0, |I|)$.*

Proof. Note that for any given set I of size $|I|$ there exist an equal number of sets of size $|A|$ containing it. Under Assump-

tion 1, the distribution of s_0 thus only depends on I and r_0. Consider the following two cases:

Case 1 (r_0 is such that $id_{r_0-1} \in I$ or $r_0 = 0$): Each of the $|I|+1$ possible assignments for r_0 in this case is equally likely to appear as r_0 is chosen uniformly at random from $[0,\dots,|A|]$.

Case 2 (r_0 is such that $id_{r_0-1} \notin I$ and $r_0 \neq 0$): In this case, picking $|I|+1$ elements from the database DB and then determining which of them is $id_{r_0-1} \notin I$ yields the same distribution as picking the set I of cardinality $|I|$ and $id_{r_0-1} \notin I$. Given these $|I|+1$ elements, each one of them is equally likely to be picked as id_{r_0-1} and thus a simple stars and bars argument gives the uniform distribution of $|\{id_0,\dots,id_{r_0-1}\} \cap I| \sim U(0,|I|)$ in this case.

The case of s_1 is analogous and the lemma follows. □

Lemma 2. *Let $(k,\ell) \in [0,\dots,|I|]^2$. The density function of the joint distribution of s_0 and s_1 is given by*

$$\begin{aligned}&Pr[s_0 = k, s_1 = \ell] \\ &= \frac{1-\delta}{(|I|+1)^2} + \frac{\delta}{(|I|+2)(|I|+1)} \cdot \begin{cases} 2 & \text{if } k = \ell \\ 1 & \text{else} \end{cases},\end{aligned}$$

where $\delta = \frac{(|A|-|I|)(|B|-|I|)}{(|A|+1)(|B|+1)}$.

Proof. As in the proof of Lemma 1, the joint distribution depends on the relation of id_{r_0-1} and id'_{r_1-1} relative to I only. This is again because one obtains the same distribution over sets whether we first pick A, B of the required cardinality and with an intersection I of a given size, or if we first pick this intersection I, the special elements id_{r_0-1} and id'_{r_1-1} and then extend to all candidates for A and B. Thus, we only need to consider the following three cases:

Case 1 (r_0 such that $id_{r_0-1} \in I$ or $r_0 = 0$ and r_1 such that $id'_{r_1-1} \in I$ or $r_1 = 0$): Since in this case r_0 and r_1 are picked independently uniformly at random from the $|I|+1$ possible assignments within $[0,\dots,|A|]$ and $[0,\dots,|B|]$, respectively, we get a uniform distribution over all $(|I|+1)^2$ pairs for (k,ℓ).

Case 2 (r_0, r_1 such that either $id_{r_0-1} \notin I$ and $r_0 \neq 0$ or $id'_{r_1-1} \notin I$ and $r_1 \neq 0$): Without loss of generality let r_0 such that $r_0 = 0$ or $id_{r_0-1} \in I$. Again we have the same distribution for this case by picking $|I|+1$ elements and then deciding which one of them corresponds to $id'_{r_1-1} \notin I$. For r_0, there are again $|I|+1$ equally likely choices, i.e. $r_0 = 0$ or any index corresponding to a document identifier of I. Further, we have $|I|+1$ equally likely assignments which of the $|I|+1$ elements corresponds to id'_{r_1-1} each of which leads to a different value for s_1. Since both of these assignments are independent, we again have a uniform distribution over all $(|I|+1)^2$ pairs for (k,ℓ).

Case 3 ($r_0 \neq 0, r_1 \neq 0$ and $id_{r_0-1} \notin I$ and $id'_{r_1-1} \notin I$): To determine the distribution, we consider picking $|I|+2$ elements from DB uniformly at random and then assigning which ones correspond to id_{r_0-1} and id'_{r_1-1}. There are $(|I|+2)(|I|+1)$ ways of assigning these two elements, each of which is equally likely. Note that the $2(|I|+1)$ assignments where there is no element of I in between id_{r_0-1} and id'_{r_1-1} lead only to $|I|+1$ different values for (s_0,s_1), namely with $s_0 = s_1$, (in these cases the order of id_{r_0-1} and id'_{r_1-1} does not matter).

Overall, one observes a bias only in Case 3, where the $|I|+1$ outcomes with $s_0 = s_1$ are twice as likely to happen as the remaining ones. Case 3 happens with probability $\frac{(|A|-|I|)(|B|-|I|)}{(|A|+1)(|B|+1)}$, i.e. when $r_0 \neq 0, r_1 \neq 0, id_{r_0-1} \notin I$ and $id'_{r_1-1} \notin I$, and the result follows. □

Lemma 2 shows that the joint distribution of s_0 and s_1 is uniform for all outcomes with $s_0 \neq s_1$ and that it has a bias bounded by 2 (depending on the ratio between the cardinality of A, B and $|I|$) towards $s_0 = s_1$. Hereby, the bias is larger whenever $|I|$ is small compared to $|A|$ and $|B|$.

Proof of Theorem 3. The cases $i = j$ and $|I| = 0$ are obvious so we omit their proof. For the case $i \neq j$, let I again denote the document identifiers in the intersection of A, the ones matching w_i, and B matching w_j. There are two cases to consider:

Case 1 ($\mathbf{C}'[i,j] \neq 0$ and $\mathbf{C}'[i,j] \neq |I|$): As is easy to see from Equation 1 this case only appears when $s_0 \neq s_1$. There are two sub-cases to consider when $\mathtt{C}'[i,j] \neq 0$ and $\mathtt{C}'[i,j] \neq |I|$. The first is when $b_0 = b_1$ and the second is when $b_0 \neq b_1$. For $b_0 = b_1$, the $2(c+1)$ possible pairs (s_0,s_1) that lead to a particular $\mathtt{C}'[i,j] = c$ are $(n, |I|-c+n)$ and $(|I|-c+n, n)$ for $n = 0,\dots,c$. One such case is depicted in Figure 5a.

For $b_0 \neq b_1$, the possible pairs (s_0,s_1) that lead to a particular $\mathtt{C}'[i,j] = c$ are of the form $(n, c+n)$ and $(c+n, n)$ for $n = 0,\dots,|I|-c$, i.e. $2(|I|-c+1)$ distinct pairs (s_0,s_1) in total. One such case is depicted in Figure 5b.

From Lemma 2, the probability of observing any pair (s_0,s_1) with $s_0 \neq s_1$, as they occur in both sub-cases, is

$$\frac{1-\delta}{(|I|+1)^2} + \frac{\delta}{(|I|+2)(|I|+1)},$$

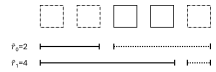
where $\delta = \frac{(|A|-|I|)(|B|-|I|)}{(|A|+1)(|B|+1)}$. Note, this probability is independent of c. As $b_0 = b_1$ and $b_0 \neq b_1$ both appear with probability $\frac{1}{2}$, the probability of observing $\mathtt{C}'[i,j] = c$ for $c = 1,\dots,|I|-1$ is simply

$$\begin{aligned}&\frac{2(c+1)+2(|I|-c+1)}{2} \cdot \mathtt{Pr}[s_0 = 0, s_1 = 1] \\ &= (|I|+2) \cdot \mathtt{Pr}[s_0 = 0, s_1 = 1] \\ &= \frac{1}{|I|+1} + \frac{1-\delta}{(|I|+1)^2}.\end{aligned}$$

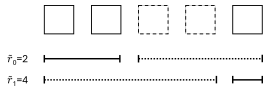
Case 2 ($\mathbf{C}'[i,j] = 0$ or $\mathbf{C}'[i,j] = |I|$):

As above, we can iterate through all relevant cases.

For $\mathtt{C}'[i,j] = 0$, there are 2 distinct choices of s_0 and s_1 whenever $b_0 = b_1$, and $s_0 \neq s_1$ in these cases. Further, there are $|I|+1$ pairs (s_0,s_1) when $b_0 \neq b_1$, each with $s_0 = s_1$.



(a) "Same side" ($b_0 = 0, b_1 = 0$).



(b) "Opposite side" ($b_0 = 0, b_1 = 1$).

Figure 5: The two sub-cases where co-occurrence counts are generated. The boxes represent the documents in the intersection (the true co-occurrence count is 5 in the figures). The solid lines below the boxes represent the document retrieved with addresses of the shape $F(sk_F, id_i \| 0)$ and the dashed lines represent the document retrieved with addresses of the shape $F(sk_F, id_i \| 1)$. The dashed boxes represent the documents that are retrieved by both queries. The number of dashed boxes is the observed co-occurrence count between the two queries.

Similarly, for $\mathtt{C}'[i,j] = |I|$, there are $|I|+1$ pairs (s_0, s_1) for $b_0 = b_1$, each with $s_0 = s_1$. Further, there are 2 distinct pairs (s_0, s_1) for $b_0 \neq b_1$ and $s_0 \neq s_1$ in these cases.

Overall, we can write the probability of observing $\mathtt{C}'[i,j] = c$ for $c \in \{0, |I|\}$ as

$$\begin{aligned}
&\frac{2}{2} \cdot \Pr[s_0 = 0, s_1 = 1] + \frac{|I|+1}{2} \cdot \Pr[s_0 = 0, s_1 = 0] \\
&= \frac{1-\delta}{(|I|+1)^2} + \frac{\delta}{(|I|+2)(|I|+1)} \\
&+ \frac{|I|+1}{2}\left(\frac{1-\delta}{(|I|+1)^2} + \frac{2\delta}{(|I|+2)(|I|+1)}\right) \\
&= \frac{1-\delta}{2(|I|+1)} + \frac{\delta}{|I|+2} + \frac{1-\delta}{(|I|+1)^2} + \frac{\delta}{(|I|+2)(|I|+1)},
\end{aligned}$$

where $\delta = \frac{(|A|-|I|)(|B|-|I|)}{(|A|+1)(|B|+1)}$. This completes the proof of Theorem 3.

□

B.2 Variable-length Documents

The construction we presented in Section 3 assumes that all documents have a fixed-length. On the other hand, documents in practice have variable length, so our implementation splits documents larger than 4KB into 4KB chunks before encryption. This effectively means that the query leakage of our implementation is $\mathcal{L}_{\textbf{Setup}}(\widetilde{\mathtt{DB}}, \mathtt{bkt})$ and the setup leakage of our implementation is $\mathcal{L}_{\textbf{Query}}(w, \widetilde{\mathtt{DB}}, \mathtt{bkt}; \mathtt{st})$ where $\widetilde{\mathtt{DB}}$ is the database after splitting the large documents, as opposed to the leakage specified in Theorem 2.

More specifically, the setup leakage will leak the total number of keyword-document pairs after splitting and bucketization (as opposed to just bucketization) and the total number of documents after splitting. For a keyword where at least one document in which it is contained has to be split, the query response volume of the keyword will be larger than the query response volume without splitting the documents. Similarly, the access pattern of the said keyword will contain more documents.

However, we argue that the "additional" leakage as the result of splitting the documents is not detrimental to the security of DDR-SSE. Intuitively, this is because the security of our scheme comes from bucketization, duplication and (pseudo)randomized document retrieval. We demonstrate this concretely with leakage cryptanalysis in Section 4.2 (where we work with variable-length documents directly).

B.3 Trade-offs for Bucketization

It is clear from the cryptanalysis results that DDR-SSE should be instantiated with bucket size greater or equal to 200 on the Enron email corpus (400K emails). Here, we examine the effect of the choice of the bucket size on efficiency.

Interestingly, we observe that in some cases, it is possible to use a slightly larger bucket size than initially planned to enhance the resilience of DDR-SSE against leakage-abuse attacks, with no additional storage cost and a small increase in the communication cost. Specifically, we note that the encrypted inverted index (instantiated with XorMM) used in DDR-SSE works with inverted indices of size powers of two (padding is used if the inverted index does not have size a power of two; this padding is different from the padding in bucketization). This means that as long as the size of the inverted index is below the next power of two, we can use a larger bucket size without affecting the storage cost of DDR-SSE at all (the size of the encrypted documents are not affected by the bucket size). Of course, a larger bucket size leads to a larger communication cost. However, for text-based databases where the true query response volumes are expected to follow a Laplace distribution, we do not expect a slightly larger bucket size to lead to a significant increase in the communication cost (as the keywords in the same bucket already have similar true query response volumes).

B.4 Leakage: DDR-SSE vs SWiSSSE

While DDR-SSE seems to outperform SWiSSSE [42] in terms of its resilience against leakage-abuse attacks, we stress that we used the exact leakage of DDR-SSE in our attack but the authors of SWiSSSE had to make pessimistic assumptions on the leakage of SWiSSSE in their attack (as it is unclear how SWiSSSE can be attacked otherwise). This makes the leakage cryptanalysis results for the two schemes not directly comparable. Nonetheless, DDR-SSE and SWiSSSE offer different leakage trade-offs, making them useful in different scenarios.