# The Fallacy of Composition of Oblivious RAM and Searchable Encryption

Muhammad Naveed
University of Illinois at Urbana-Champaign

## ABSTRACT

Oblivious RAM (ORAM) is a tool proposed to hide access pattern leakage, and there has been a lot of progress in the efficiency of ORAM schemes; however, less attention has been paid to study the applicability of ORAM for cloud applications such as symmetric searchable encryption (SSE). Although, searchable encryption is one of the motivations for ORAM research, no in-depth study of the applicability of ORAM to searchable encryption exists as of June 2015. In this work, we initiate the formal study of using ORAM to reduce the access pattern leakage in searchable encryption.

We propose four new leakage classes and develop a systematic methodology to study the applicability of ORAM to SSE. We develop a worst-case communication baseline for SSE. We show that completely eliminating leakage in SSE is impossible. We propose single keyword schemes for our leakage classes and show that either they perform worse than streaming the entire outsourced data (for a large fraction of queries) or they do not provide meaningful reduction in leakage. We present detailed evaluation using the Enron email corpus and the complete English Wikipedia corpus.

## 1. INTRODUCTION

Oblivious RAM (ORAM) enables accessing the memory without leaking the access pattern. A lot of effort is underway to make ORAM schemes efficient. Searching for the exact phrase "Oblivious RAM" on the Google Scholar returns 671 articles, 565 of them published since 2010. ORAM is a cryptographic primitive that allows accessing memory without leaking the access pattern; however, it provides a random access memory (RAM) interface that allows accessing a block of memory using its address. Therefore, the access pattern hiding properties of ORAM are only valid if memory is accessed single block at a time, which is not always the case with real applications. Many applications such as searchable encryption (SSE) or cloud storage[1], requires downloading multiple blocks at a time as illustrated in Figure 1, which leaks the number of blocks being accessed for a particular request. The number of blocks in turn leaks partial access pattern information. In SSE, each keyword query requires multiple blocks to be accessed. As different keywords require different number of blocks to be retrieved, the number of blocks retrieved leaks some information about the query. Similarly, in cloud storage, a file may consist of multiple blocks; therefore, downloading the file leaks its size (rounded off to the block size). Suppose that a file has a unique size, then every time the client accesses this file, the server observes, from the number of blocks being accessed, that the client is accessing the same file.

Symmetric Searchable Encryption (SSE) enables a client to store encrypted documents on a server and search over her encrypted documents to selectively retrieve the documents. In the SSE setting, the client prepares an inverted index for her documents and encrypts it using SSE, encrypts her documents using a symmetric cipher (such as AES), and sends both the encrypted index and the encrypted documents to the server. The server does not see the contents of the documents or the queries, but for each query it observes some information leakage: the search and document-access patterns (combination of search and document-access patterns is commonly referred to as access pattern). Access pattern is a significant leakage and may enable the server to infer information about the queries or the documents [18].

Preventing the leakage in SSE is one of the motivations for the applied ORAM research; however, no SSE construction that prevents the access pattern leakage has been proposed yet. A search for articles on the Google Scholar that contains both exact phrases "*Oblivious RAM*" and "*Searchable Encryption*" returns 214 articles. At least 18 ORAM papers cite Islam et al. 2012 paper titled "*Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation*" [18] to motivate the importance of eliminating leakage and the capability of ORAM to do so in applications such as SSE. Similarly, it is widespread in the SSE literature that ORAM can completely eliminate leakage in SSE. We show that using ORAM to completely eliminate access pattern leakage or even achieve weaker notions of access pattern hiding either renders the communication performance worse than the trivial approach of streaming all of the outsourced data for each query or they do not provide any meaningful reduction in leakage. To the best of our knowledge, this paper is the first work studying the applicability of ORAM to SSE.

### 1.1 The fallacy of Composition

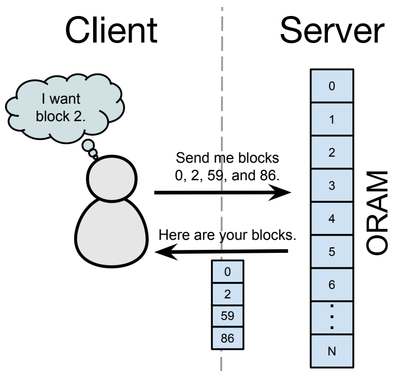
The fallacy of Composition occurs when one infers that something is true of the whole based on the fact that it is true for a part of the whole, without any justification for the inference. An example of the fallacy of Composition follows: Atoms are colorless. Cats are made of atoms. Therefore, cats are colorless.

The widespread fallacy of Composition of Oblivious RAM and Searchable Encryption is: *Searchable encryption leaks access pattern. Oblivious RAM eliminates access pattern. Therefore, there exists a method of using Oblivious RAM to eliminate access pattern in searchable encryption.* This fallacy is prevalent in ORAM and Searchable Encryption literature; however, for brevity we do not cite all of the papers.

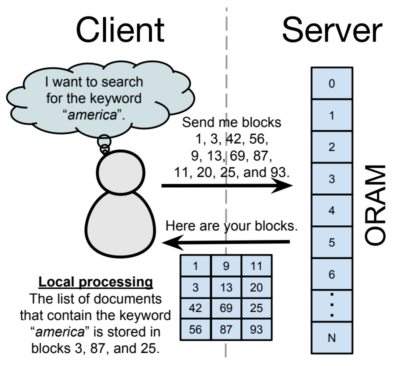
### 1.2 Overview

We propose a new systematic methodology to study the applicability of ORAM to searchable encryption (SSE). We develop a baseline linear searchable encryption scheme: Linear–$\mathcal{LC}0$–SSE. It supports arbitrary queries and leaks absolutely no information, except an upperbound on the total size of the data stored on the server. For each query, Linear–$\mathcal{LC}0$–SSE, however, needs to stream all the outsourced data, storing only a small constant amount of data on the client at a time. The client needs to stream all the outsourced data, but she does *not* need to store all of it locally. She can stream in small chunks and discard any streamed chunk that does not satisfy the query as soon as the local search on it completes. This makes Linear–$\mathcal{LC}0$–SSE an excellent worst-case baseline to gauge the communication performance of any SSE scheme. If an SSE

[1]By cloud storage, we mean services such as Dropbox and Google Drive that allow users to store their files.



**(a)** ORAM functionality


Client
Server
I want to search for the keyword "*america*".
Send me blocks 1, 3, 42, 56, 9, 13, 69, 87, 11, 20, 25, and 93.
Here are your blocks.
Local processing
The list of documents that contain the keyword "*america*" is stored in blocks 3, 87, and 25.
ORAM
1 9 11
3 13 20
42 69 25
56 87 93
0
1
2
3
4
5
6
N

**(b)** SSE on top of ORAM

**Figure 1:** ORAM functionality vs. SSE requirements. For each query, SSE accesses multiple blocks which leaks the number of blocks being accessed. As the number of blocks accessed depends upon the query, it leaks some information about the access pattern. The ORAM overhead of 4 shown is just for ease of exposition. (a). The client downloads multiple blocks (e.g., 4 in the figure) to access a single block from the ORAM. (b). The client needs to retrieve 3 blocks from the ORAM and therefore downloads 12 blocks. The server learns that the client downloaded 3 blocks.

scheme needs to communicate more data then Linear–$\mathcal{LC}0$–SSE, then it is always better to use Linear–$\mathcal{LC}0$–SSE. Note that Linear–$\mathcal{LC}0$–SSE does *not* require ORAM.

We propose four new leakage classes: $\mathcal{LC}0$, $\mathcal{LC}1$, $\mathcal{LC}2$, and $\mathcal{LC}3$. We also discuss $\mathcal{LC}4$ which is the leakage of standard SSE schemes [20, 23]. Each leakage class captures the information leaked to the server, with $\mathcal{LC}0$ leaking the least and $\mathcal{LC}4$ leaking the most information among the five classes. We prove that $\mathcal{LC}0$ is impossible to achieve without downloading all outsourced data for each and every query (communication required by Linear–$\mathcal{LC}0$–SSE). We propose single keyword SSE schemes for $\mathcal{LC}1$, $\mathcal{LC}2$, and $\mathcal{LC}3$. We empirically demonstrate that for a large fraction of queries, $\mathcal{LC}1$–SSE and $\mathcal{LC}2$–SSE perform worse than downloading the entire outsourced data. We emphasize that a small fraction of keywords are accessed most frequently (Zipf's law) and therefore these small number of keywords constitute a large fraction of the queries. Moreover, we demonstrate that $\mathcal{LC}3$ does not provide any meaningful reduction in leakage over $\mathcal{LC}4$.

**Contributions.** We summarize our contributions below:

- **First work on composition of ORAM and SSE.** To the best of our knowledge, this is the first work studying the applicability of ORAM to reduce the leakage in SSE.
- **New Leakage Classes.** We propose four new leakage classes for SSE.
- **New Systematic Methodology.** We develop a new systematic methodology to study the applicability of ORAM to SSE. Although, we specifically developed the methodology for SSE, we believe that it can be useful for other applications as well. First, we design a baseline scheme, called Linear–$\mathcal{LC}0$–SSE, with a worst-case communication performance. Second, we propose static single keyword schemes for $\mathcal{LC}1$, $\mathcal{LC}2$, and $\mathcal{LC}3$. Third, we empirically study the communication performance of $\mathcal{LC}1$–SSE, $\mathcal{LC}2$–SSE, $\mathcal{LC}3$–SSE, and $\mathcal{LC}4$–SSE (Naveed's et al. scheme [23]) relative to Linear–$\mathcal{LC}0$–SSE.
- **Interesting findings.** We report interesting findings about the applicability of ORAM to the leakage prevention in SSE. Using our systematic methodology, we show that it is impossible to eliminate leakage in SSE without downloading the entire outsourced data. We show that $\mathcal{LC}1$–SSE and $\mathcal{LC}2$–SSE have query communication worse than that of Linear–$\mathcal{LC}0$–SSE for a large fraction of queries (a small fraction of keywords make up the large fraction of queries). We also demonstrate that $\mathcal{LC}3$ and $\mathcal{LC}4$ leak almost the same amount of information.
- **Evaluation.** We provide a detailed evaluation of the query communication performance of the single keyword $\mathcal{LC}1$–SSE, $\mathcal{LC}2$–SSE, $\mathcal{LC}3$–SSE, and $\mathcal{LC}4$–SSE schemes using the Enron Email Corpus and the complete English Wikipedia Corpus.

**Scope of the paper.** We focus only on the applicability of ORAM to reduce access pattern leakage in SSE. ORAM has other applications such as secure multiparty computation and secure co-processor; however, these applications are out of scope of this paper.

Most of the ORAM schemes work with a storage-only server with a few exceptions. Throughout the paper, we assume the server to be a storage-only resource that allows the client to download and upload blocks of data.

**Organization.** The paper is organized as follows: Section 2 presents the leakage classes. Section 3 presents a worst-case communication baseline scheme (Linear–$\mathcal{LC}0$–SSE). Section 4 presents single keyword constructions for our leakage classes. Section 5 presents detailed evaluation. Section 6 presents the related work. Finally, Section 7 concludes the paper.

## 2. LEAKAGE CLASSES

We present five different leakage classes: $\mathcal{LC}0$, $\mathcal{LC}1$, $\mathcal{LC}2$, $\mathcal{LC}3$, and $\mathcal{LC}4$. $\mathcal{LC}0$ represents absolute minimum leakage and $\mathcal{LC}4$ captures the leakage of a typical static symmetric searchable encryption (SSE) scheme (e.g., [20, 23]). $\mathcal{LC}1$, $\mathcal{LC}2$, and $\mathcal{LC}3$ leak more information than $\mathcal{LC}0$ but less than $\mathcal{LC}4$. Table 1 shows all the leakage classes. To clearly explain the leakage, Table 1 shows the leakage for $\mathcal{LC}0$, $\mathcal{LC}1$, $\mathcal{LC}2$, and $\mathcal{LC}3$ relative to $\mathcal{LC}4$ which is the prevalent standard for SSE schemes in the literature [20, 23]; we believe that this juxtaposition make the difference in leakage easy to compare and understand.

For each leakage class we describe setup and query leakage. Setup leakage shows the information leaked when the client initially send the documents and index to the server and the query

| | $\mathcal{LC}0$ | $\mathcal{LC}1$ | $\mathcal{LC}2$ | $\mathcal{LC}3$ | $\mathcal{LC}4$ |
|---|---|---|---|---|---|
| | **Leakage Classes** | | | | |
| **Setup Leakage** | | | | | |
| Total combined size of index (if applicable) and all documents | ✓ | ✓ | ✓ | ✓ | ✓ |
| Total size of all documents | ✗ | ✗ | ✓ | ✓ | ✓ |
| Size of index | ✗ | ✗ | ✓ | ✓ | ✓ |
| **Query Leakage** For each queried keyword $q$: | | | | | |
| **Search Pattern** | | | | | |
| Number of times $q$ is queried, i.e., the access frequency of $q$ | ✗ | ✗ | ✗ | ✗ | ✓ |
| If $q$ is the same or different from any of the keywords queried in the past | ✗ | ✗ | ✗ | ✗ | ✓ |
| **Document-Access Pattern** | | | | | |
| Identifiers of the documents that contain $q$ | ✗ | ✗ | ✗ | ✓ | ✓ |
| Number of documents that contain $q$ | ✗ | ✗ | ✓ | ✓ | ✓ |
| Size of each document that contain $q$ | ✗ | ✗ | ✗ | ✓ | ✓ |
| Total size of all documents that contain $q$ | ✗ | ✓ | ✓ | ✓ | ✓ |

**Table 1:** Leakage Classes for Static Single Keyword Searchable Encryption. Padding can be used to hide the exact sizes and leak an upperbound on the sizes instead. ✗ shows the information that is *not leaked* and ✓ shows the information that is *leaked*.

leakage shows the information leaked by the queries. Query leakage is divided into search pattern and document-access pattern: search pattern represents the leakage about the query keyword itself, while document-access pattern represents the leakage about the documents that contain the query keyword. Most of the literature combine search and document-access pattern and call it access pattern, however, for ease of exposition we explain them separately.

**Padding** can be used to hide the exact size and leak an upperbound on the size instead. For simplicity, we omit this fact from this point onwards unless necessary.

**Leakage Class 0 ($\mathcal{LC}0$)** represents the minimum possible leakage for an SSE scheme. It only leaks the combined size (with padding an upperbound on the size) of all documents and index (only if the SSE scheme uses index) during the initial outsourcing to the server; however, this information is impossible to hide given that the server is storing the data. No information whatsoever is leaked during queries, and hence, there is no query leakage.

**Leakage Class 1 ($\mathcal{LC}1$)** captures the minimum amount of leakage of an SSE scheme with the query communication complexity linear in the size of the documents that satisfy the query. As the query communication complexity is linear in the total size of the documents that satisfy the query, an upperbound on the total size of the documents that satisfy the query is inherently leaked. Moreover, without appropriate padding, the exact total size of documents being retrieved is leaked; we consider this case for simplicity.

**Leakage Class 2 ($\mathcal{LC}2$)** leaks the size as well as the number of documents that satisfy the query. Moreover, during the setup the size of the index and total size of all documents is leaked.

**Leakage Class 3 ($\mathcal{LC}3$)** does not explicitly leak the search pattern, but does leak the document-access pattern. Document-access pattern implicitly leaks search pattern except in the following rare situation: If any two keywords $q$ and $q'$ appears in exactly the same set of documents, then the server is not able to distinguish between $q$ and $q'$. This condition is rare in realistic data and therefore $\mathcal{LC}3$ is almost same as $\mathcal{LC}4$ which is leakage of standard SSE schemes. We explain $\mathcal{LC}3$ to demonstrate that a straightforward method of obliviously accessing only the index in an SSE scheme is not useful, we explain this in detail in Section 5. Fig. 10 shows that $\mathcal{LC}3$ is almost same as $\mathcal{LC}4$.

**Leakage Class 4 ($\mathcal{LC}4$)** captures the leakage of a static single keyword SSE scheme (e.g., [20, 23]). It leaks the complete search and document-access patterns.

## 3. COMMUNICATION BASELINE

In this section, we present a simple Linear–$\mathcal{LC}0$–SSE scheme, an SSE scheme with query communication complexity linear in the total size of all documents, as a communication performance baseline to study the communication performance of our single keyword $\mathcal{LC}1$–SSE, $\mathcal{LC}2$–SSE, and $\mathcal{LC}3$–SSE schemes we propose in Section 4.

**Linear–$\mathcal{LC}0$–SSE scheme.** We propose a Linear–$\mathcal{LC}0$–SSE scheme in Figure 2. The client encrypts her documents with any semantically secure symmetric-key encryption scheme (such as AES) and sends the encrypted documents to the server. Later, when the client wants to search her outsourced documents, she streams all of them. The client streams data in small chunks, searches them locally, and discards the streamed documents that do not satisfy the query.

Our Linear–$\mathcal{LC}0$–SSE scheme has the following properties:

- **Absolute minimum leakage ($\mathcal{LC}0$).** It has leakage class $\mathcal{LC}0$, which means it leaks absolute minimum information.
- **Arbitrary queries.** It supports arbitrary type of queries[2].
- **Worst-case query communication.** It streams all outsourced documents to the client for each query.
- **Optimal query communication for $\mathcal{LC}0$.** It has optimal query communication. Theorem 4.1 shows that Linear–$\mathcal{LC}0$–SSE has optimal communication required to achieve $\mathcal{LC}0$.
- **Constant local storage.** It requires the client to store only a single document at any given instant; the client keeps the document if it satisfy the query and discards it otherwise. Therefore, in addition to the documents that satisfy the query, the client only needs to store a single more document.
- It does not need ORAM to achieve access pattern hiding.

[2]Sublinear SSE schemes are designed for a specific type of queries such as single keyword and Boolean keyword queries.

**Keygen**

- The client generates a symmetric key $K$ uniformly at random.

**Setup**

- The client encrypts all the documents as a single file using a semantically secure symmetric-key encryption scheme with key $K$, and sends it to the server.

**Search**

- For each search query, the client streams all of the outsourced documents and searches locally. The client stores only a (small) constant amount of data, say a single document, at any given time. All documents that do not satisfy the query are discarded after the completion of local search.

**Figure 2:** Linear–$\mathcal{LC}0$–SSE Scheme. Baseline for communication performance of SSE schemes we propose in Section 4.

**Baseline Query Communication.** Linear–$\mathcal{LC}0$–SSE supports arbitrary queries and has absolute minimum leakage ($\mathcal{LC}0$). The only problem Linear–$\mathcal{LC}0$–SSE has is that for each query it requires streaming all outsourced data to the client. Therefore, it serves as an excellent worst-case communication baseline for any SSE scheme. *Any SSE scheme with query communication worse than the Linear–$\mathcal{LC}0$–SSE scheme can be trivially replaced with the Linear–$\mathcal{LC}0$–SSE scheme.*

## 4. CONSTRUCTIONS

In this section, we first show that achieving $\mathcal{LC}0$ is, in general, impossible with query communication better than that of Linear–$\mathcal{LC}0$–SSE. We construct single keyword SSE schemes for $\mathcal{LC}1$, $\mathcal{LC}2$, and $\mathcal{LC}3$ such that they have *optimal query communication* for the respective leakage class. These schemes capture simple ways of using Oblivious RAM as a blackbox to reduce leakage in SSE. We show in section 5 that query communication of $\mathcal{LC}1$-SSE and $\mathcal{LC}2$-SSE is worse than that of Linear–$\mathcal{LC}0$–SSE for a large fraction of queries (only a small number of keywords constitute most of the queries). Moreover, $\mathcal{LC}3$-SSE has acceptable query communication but as explained in section 5 it does not provide meaningful reduction in leakage compared to $\mathcal{LC}4$. Security of our schemes follows from the security of the underlying ORAM and ODICT schemes; therefore, we omit the security proofs.

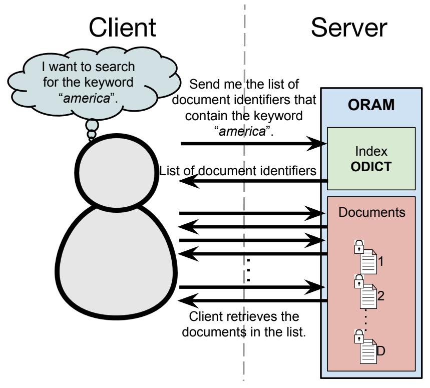
**Oblivious Dictionary (ODICT).** Sublinear SSE schemes require inverted index to function. Index needs a dictionary data structure and ORAM is not a dictionary data structure. Therefore, we use AVL tree based Oblivious Map scheme of Wang et al. [31] as an Oblivious Dictionary in our schemes. We consider that ODICT stores key value pairs and ODICT.Lookup(key) operation returns the value associated with the key.

### 4.1 $\mathcal{LC}0$–SSE

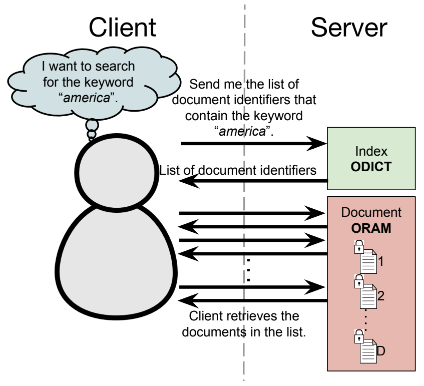
We show that a scheme with Leakage Class 0 ($\mathcal{LC}0$) necessarily requires downloading the entire outsourced data for each query. Therefore, completely eliminating the leakage with communication less than that of the Linear–$\mathcal{LC}0$–SSE scheme (i.e., streaming all of the outsourced data) is impossible.

THEOREM 4.1. *The lower bound on the query communication complexity to achieve the Leakage Class 0 ($\mathcal{LC}0$) is $|\mathbb{D}|$, where $|\mathbb{D}|$ is the total size of all outsourced documents.*

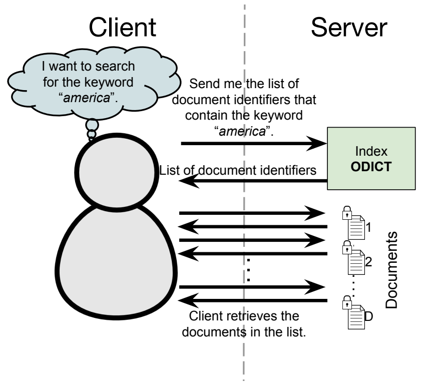
PROOF. We prove that if for any query the communication is less than $|\mathbb{D}|$, then the information leaked to the server is strictly


Client
Server
I want to search for the keyword "america".
Send me the list of document identifiers that contain the keyword "america".
ORAM
Index ODICT
List of document identifiers
Documents
1
2
D
Client retrieves the documents in the list.

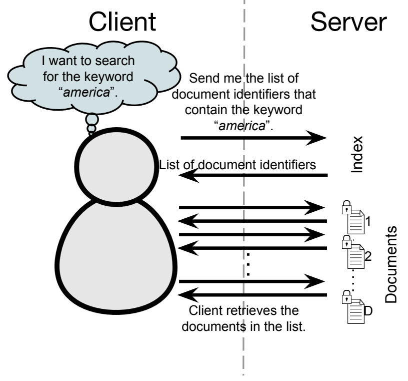
**(a)** Single Keyword $\mathcal{LC}1$-SSE


Client
Server
I want to search for the keyword "america".
Send me the list of document identifiers that contain the keyword "america".
Index ODICT
List of document identifiers
Document ORAM
1
2
D
Client retrieves the documents in the list.

**(b)** Single Keyword $\mathcal{LC}2$-SSE


Client
Server
I want to search for the keyword "america".
Send me the list of document identifiers that contain the keyword "america".
Index ODICT
List of document identifiers
1
2
D
Documents
Client retrieves the documents in the list.

**(c)** Single Keyword $\mathcal{LC}3$-SSE


Client
Server
I want to search for the keyword "america".
Send me the list of document identifiers that contain the keyword "america".
Index
List of document identifiers
1
2
D
Documents
Client retrieves the documents in the list.

**(d)** Single Keyword $\mathcal{LC}4$-SSE. This captures leakage of standard SSE schemes such as [20, 23].

**Figure 3:** In all the schemes, the client searches for a keyword and retrieves a list of document identifiers that contain the keyword. Next the client retrieves the documents in this list from the server. *For simplicity, the figures are only showing the important details.*

more than $\mathcal{LC}0$. Suppose for a query $q$, the client retrieves documents with total size $|\mathbb{D}'| < |\mathbb{D}|$, then the server learns the size of the documents being retrieved $|\mathbb{D}'|$. Suppose the client has another query $q'$ that is satisfied by all the documents. For an SSE scheme to be correct, it has to download all the documents for the query $q'$, i.e., the size of the documents retrieved for the query $q'$ needs to be $|\mathbb{D}|$. As the size of the data retrieved for the query $q$ and the query $q'$ are different, the server can distinguish the query $q$ from the query $q'$, which is strictly more leakage than $\mathcal{LC}0$. Moreover, the server learns an upperbound on the total size $|\mathbb{D}'|$ of the documents that match the query $q$. This shows that the only way to prevent this leakage is to retrieve all documents for every query. Therefore, we conclude that $|\mathbb{D}|$ is the optimal communication for the Leakage Class 0 ($\mathcal{LC}0$). $\square$

## 4.2 $\mathcal{LC}1$–SSE

$\mathcal{LC}1$-SSE prevents as much leakage as possible while keeping the query communication less than the communication of Linear–$\mathcal{LC}0$–SSE. We propose a single keyword $\mathcal{LC}1$-SSE scheme to study the query communication required to achieve $\mathcal{LC}1$.

**Single Keyword $\mathcal{LC}1$–SSE Scheme.** We present our Single Keyword $\mathcal{LC}1$–SSE scheme in fig. 4. The client uses an ORAM $O$ to store both the index ODICT and the documents. To query a keyword, the client looks up the inverted index stored in ODICT $D_O$ to retrieve the list $L$ of document identifiers that contain the query keyword. She then retrieves all the documents in the list $L$ from the ORAM $O$.

**Query Communication.** Let $n$ and $d$ be the number of documents and the total size of the documents that satisfy a query, $e$ be the size of $n$ document identifiers, $o$ be the overhead of the ORAM, $co$ be the overhead of the ODICT (where $c$ shows the number of ORAM accesses required for a single ODICT lookup), and $B$ be the blocksize used in ORAM and ODICT, then the communication overhead of $\mathcal{LC}1$-SSE is $co \times \max(B, \lceil \frac{e}{B} \rceil) + o \times \max(B, \lceil \frac{d}{B} \rceil)$. We prove in appendix A.1 that for the fixed values of $o$ and $c$, this is the optimal query communication required to achieve $\mathcal{LC}1$. We demonstrate using real datasets in Section 5 that the optimal communication single keyword $\mathcal{LC}1$–SSE scheme requires more communication than Linear–$\mathcal{LC}0$–SSE for a large fraction of the queries (a small number of keywords constitute a large fraction of queries).

**Keygen.**

- The client generates a symmetric key $K_O$ uniformly at random and use it in all symmetric key operations of ODICT $D_O$ and ORAM $O$.

**Setup.**

- The client setups an ORAM $O$ on the server. She uses ORAM $O$ to store both an ODICT $D_O$ for the index and the documents.
- The client creates an inverted index for all documents.
- The client writes the index in the ODICT $D_O$. This can be done during ORAM $O$ setup.
- The client writes all the documents in the ORAM $O$. This can also be done during ORAM $O$ setup.

**Search.**

- To query a keyword $q$, the client queries the ODICT $D_O$ using $D_O$.Lookup($q$) and retrieves the list $L$ of documents that contain the keyword.
- The client retrieves all the documents in list $L$ from ORAM $O$.

**Figure 4:** Single Keyword $\mathcal{LC}1$-SSE Scheme

**Keygen.**

- The client generates a symmetric key $K$ uniformly at random, which is used for encrypting blocks of ODICT $D$ and ORAM $O$.

**Setup.**

- The client creates an inverted index for all documents.
- The client writes the index in an ODICT $D$ stored on the server. This can be done during ODICT $D$ setup.
- The client writes all the documents in an ORAM $O$ stored on the server. This can be done during ORAM $O$ setup.

**Search.**

- To query keyword $q$, the client queries ODICT $D$ using $D$.Lookup($q$) and retrieves the list $L$ of document identifiers that contain the keyword.
- The client retrieves all the documents in list $L$ from ORAM $O$.

**Figure 5:** Single Keyword $\mathcal{LC}2$-SSE Scheme

## 4.3 $\mathcal{LC}2$–SSE

We propose a single keyword $\mathcal{LC}2$–SSE scheme to investigate Leakage Class 2 ($\mathcal{LC}2$). Our single keyword $\mathcal{LC}1$–SSE and $\mathcal{LC}2$–SSE schemes differ due the fact that the former stores both index ODICT and documents in the same ORAM while the later stores index ODICT and documents separately as shown in fig. 3.

**Single Keyword $\mathcal{LC}2$–SSE Scheme.** We describe a single keyword $\mathcal{LC}2$ scheme in Figure 5. As shown in Figure 3b, it uses an ODICT to store index and an ORAM to store the documents. To search for a keyword $q$, the client retrieves a list $L$ of document identifiers of documents that satisfy $q$ from the ODICT $D$. Next, the client retrieves the documents in list $L$ from the document ORAM $O$.

**Query Communication.** Let $n$ and $d$ be the number of documents and the total size of the documents that satisfy a query, $e$ be the size of $n$ document identifiers, $o$ be the overhead of the ORAM, $o'$ be the overhead of the ODICT, and $B$ be the blocksize used in ORAM and ODICT, then the communication overhead of $\mathcal{LC}2$-SSE is $o' \times \max(B, \lceil \frac{e}{B} \rceil) + o \times \max(B, \lceil \frac{d}{B} \rceil)$. We prove in appendix A.2 that for the fixed values of $o$ and $o'$, this is the optimal query communication required to achieve $\mathcal{LC}2$. As $\mathcal{LC}1$–SSE stores both index ODICT and documents in the same ORAM and $\mathcal{LC}2$–SSE stores them separately, therefore, the size of the ORAM in $\mathcal{LC}1$–SSE is bigger than index ODICT and document ORAM in $\mathcal{LC}2$–SSE. Moreover, the ORAM and ODICT access increases with their size, the $\mathcal{LC}2$–SSE has slightly better query communication; however, it may not worth the extra amount of information being leaked.

## 4.4 $\mathcal{LC}3$–SSE.

We propose a single keyword $\mathcal{LC}3$–SSE scheme to study Leakage Class 3 ($\mathcal{LC}3$). Our single keyword $\mathcal{LC}3$–SSE scheme uses ODICT for the index but stores document using plain encryption without ORAM.

**Single Keyword $\mathcal{LC}3$–SSE Scheme.** We present a Single Keyword $\mathcal{LC}3$–SSE scheme in Figure 6. This scheme uses ODICT for the inverted index and symmetric key encryption to store the documents.

**Query Communication Complexity.** Let $n$ and $d$ be the number of documents and the total size of the documents that satisfy a query, $e$ be the size of $n$ document identifiers, $o'$ be the overhead of the ODICT, and $B$ be the blocksize used in ODICT, then the communication overhead of $\mathcal{LC}3$-SSE is $o' \times \max(B, \lceil \frac{e}{B} \rceil) + d$.

**Keygen**

- The client generates two keys $K$ and $K'$ uniformly at random. $K$ is used to encrypt blocks of ODICT $D$ and $K'$ to encrypts the documents.

**Setup**:

- The client creates an inverted index for all documents.
- She writes the index in an ODICT $D$. This is done during ODICT $D$ setup.
- She encrypts each document with key $K$ using a semantically secure symmetric key encryption and send the encrypted documents to the server.

**Search**:

- To query a keyword $q$, the client queries the ODICT $D$ using $D$.Lookup($q$) and retrieves the list $L$ of documents that contain the keyword.
- The client retrieves all the documents in the list $L$ from the server.

**Figure 6:** Single Keyword $\mathcal{LC}3$-SSE Scheme

We prove in appendix A.3 that for the fixed values of $o'$, this is the optimal query communication required to achieve $\mathcal{LC}3$.

## 5. EVALUATION

We have implemented our protocols to evaluate the query communication; however, we only simulate the ORAM accesses, which we believe is enough to evaluate the query communication overhead. We present detailed and comprehensive experiments.

In the first set of experiments fig. 7, which we call realistic experiments, we use simulated PathORAM ($4\log(N)$ overhead, where $N$ represents the total number of blocks) [30] for the document ORAM and the Wang's et al. AVL tree based Oblivious Map ($(4\log(N))^2$ overhead) [31] for the index ODICT.

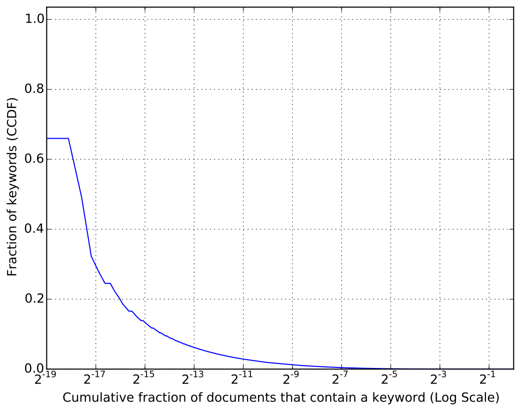
In the second set of more conservative experiments fig. 8, which we call optimistic experiments, we use simulated ideal ORAM with $\log(N)$[3] overhead for the document ORAM. Furthermore, we assume that a single ODICT lookup requires a single ORAM accesses.

We use a blocksize of 4KB throughout.

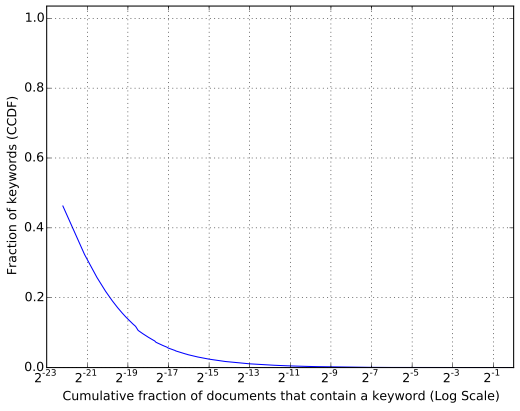
Before presenting our results, we describe the preliminaries required to understand our results.

- **Empirical CCDF Plots.** We present our evaluation results using *empirical* Complementary Cumulative Distribution Function (CCDF) plots. For example, a point $(x, y)$ in figs. 7 and 8 indicate that the query overhead is at least $x$ for $y$ fraction of the keywords.
- **Query overhead.** Query overhead of a query is the ratio of the amount of data retrieved in an SSE scheme to the total size of all the outsourced documents. This allows us to compare the query communication of our SSE schemes to our baseline Linear–$\mathcal{LC}0$–SSE scheme: if the ratio is greater or equal to 1, then the SSE scheme does not perform better than our baseline Linear–$\mathcal{LC}0$–SSE.
- **Keyword Frequency.** We plot keyword frequencies of all the keywords in the Enron email corpus and the English Wikipedia corpus in fig. 9; which shows that keywords follow Zipf's law.

[3]No existing ORAM scheme have this overhead. A loose ORAM bound proved by [16] is $\log(N)$.



**(a)** Enron Email Corpus



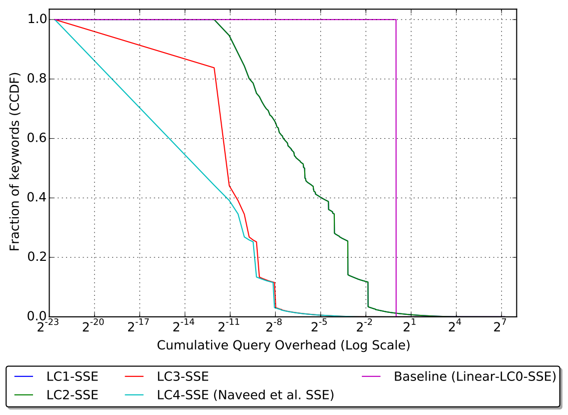
**(b)** English Wikipedia Corpus

**Figure 9:** Keyword Frequency. A point $(x, y)$ in the plots indicates that the fraction of the number of documents that contain a keyword is at least $x$ for $y$ fraction of the keywords.

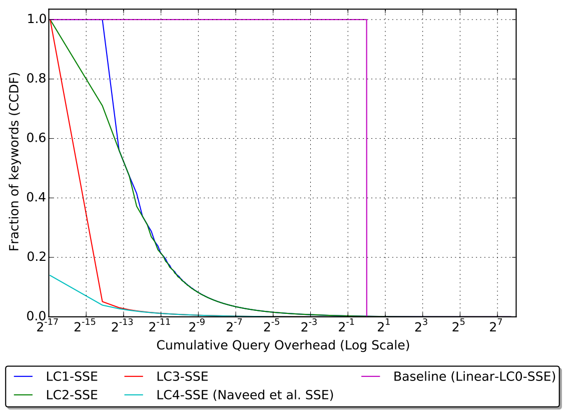
- **Query Pattern.** The more frequent the keyword is the more frequently it will be queried (we have removed all stopwords). As shown in fig. 9, a small number of keywords have very high frequency, therefore, these small number of keywords constitutes most of the queries. This fact is important in understanding the query overhead; in addition to query overhead plots for the all the keywords, we present query overhead plots zoomed in at the tail to clearly show the overhead for the most frequent keywords that make up a large fraction of the queries.

**Datasets.** We use two datasets: the complete English Wikipedia corpus and the Enron email dataset. The English Wikipedia corpus contains 4,825,180 distinct articles and is 9.8GB in size. Enron email dataset [1] contains emails from 150 Enron employees, mostly senior managers. The dataset was made public by the Federal Energy Regulatory Commission during its investigation of Enron. The dataset is 1.32GB in size and has 517,424 emails. Enron dataset has been extensively used to evaluate searchable encryption schemes [20, 4, 23].

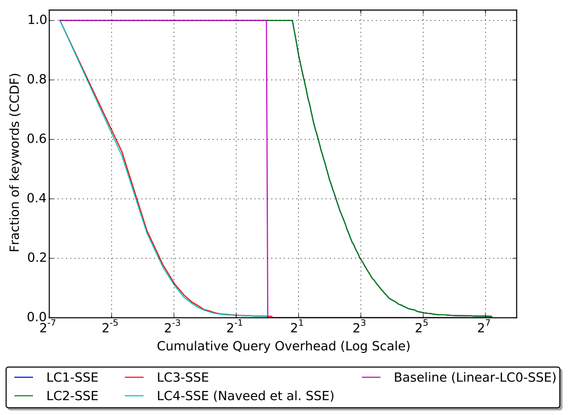
We removed all stopwards and non-alphabetical characters, and converted all keywords to the lowercase. After this preprocessing, English Wikipedia corpus had 4,400,034 keywords and Enron data



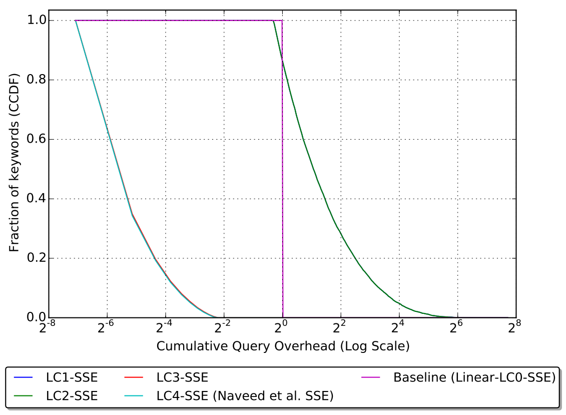
**(a)** Enron Email Corpus (all $628,908$ keywords)



**(b)** English Wikipedia Corpus (all $4,400,034$ keywords)



**(c)** Enron Email Corpus (most frequent $5,000$ keywords)



**(d)** English Wikipedia Corpus (most frequent $10,000$ keywords)

**Figure 7:** Cumulative Query Overhead using PathORAM for the document ORAM and AVL tree based Oblivious Map [31] for the Index ODICT. Query overhead of a query is the ratio of the amount of data retrieved in an SSE scheme to the total size of all the outsourced documents. A point $(x, y)$ in the plots indicates that the query overhead is at least $x$ for $y$ fraction of the keywords.

had 628,908 keywords.

**Realistic Experiments.** As explained above we use PathORAM for the documents and AVL tree based Oblivious Map for the index ODICT. $\mathcal{LC}3$–SSE does not use document ORAM and $\mathcal{LC}4$–SSE does not use both document ORAM and index ODICT.

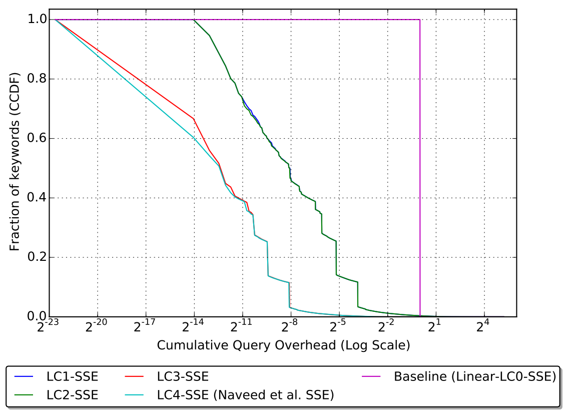
Fig. 7a shows the query overhead for all keywords in the Enron Email Corpus. $\mathcal{LC}1$–SSE and $\mathcal{LC}2$–SSE have the same query overhead and therefore their curves are overlapping. $\mathcal{LC}3$–SSE query overhead is slightly more than that of $\mathcal{LC}4$–SSE. Moreover, $\mathcal{LC}1$–SSE and $\mathcal{LC}2$–SSE query overhead is more than $1$ for a large fraction of queries (a small number of keywords constitutes a large fraction of queries). Fig. 7c shows zoomed in view of the tail of fig. 7a for $5,000$ most frequent keywords, which shows that for all of the $5,000$ most frequent keywords the query overhead is at least $1.75$.

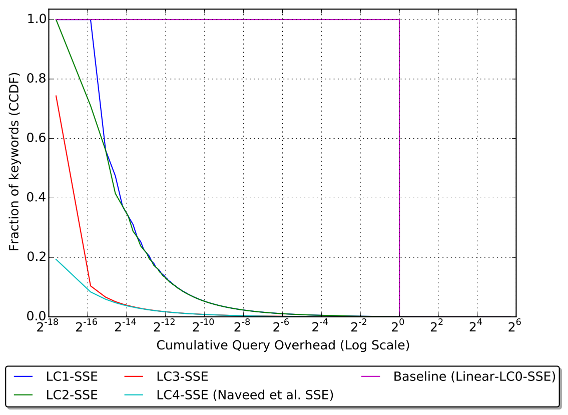
Fig. 7b and fig. 7d shows similar results for the English Wikipedia corpus.

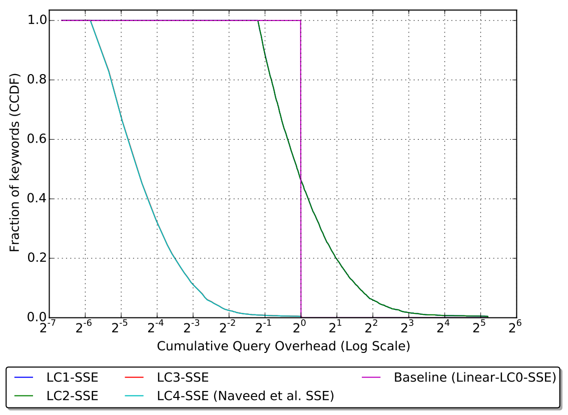
**Optimistic Experiments.** Fig. 8 shows results for our optimistic experiments, where we use ideal ORAM and ODICT, both with the communication overhead of $\log(N)$, where $N$ is the number of blocks. The results are better than our realistic experiments, however, a large fraction of queries still has query overhead of more than $1$.

**Leakage Analysis of $\mathcal{LC}3$–SSE and $\mathcal{LC}4$–SSE.** Our experiments show that $\mathcal{LC}3$–SSE query overhead is almost same as $\mathcal{LC}4$–SSE; therefore, we investigate whether $\mathcal{LC}3$–SSE provide meaningful reduction in leakage compared to $\mathcal{LC}4$–SSE. As explained in Section 2 $\mathcal{LC}3$–SSE prevents explicit leakage of search pattern, but leaks search pattern indirectly from the document access pattern. Specifically, the only information that is not leaked about the search pattern from the document-access pattern is whether the client is querying for two keywords that appear in exactly the same set of documents. Fig. 10 shows the fraction of keywords for which there are at least a fraction of other keywords appearing in the same set of documents. As it can been seen, a very small fraction of keywords have a very small fraction of other keywords that appear in the same set of documents; therefore, we conclude that $\mathcal{LC}3$ leaks almost as much as $\mathcal{LC}4$. This implies that oblivious index accesses alone does not provide a meaningful reduction in leakage if the documents are accessed in a non-oblivious fashion. Therefore, for any meaningful reduction in leakage both the index and documents accesses need be oblivious.

## 6. RELATED WORK


**(a)** Enron Email Corpus (all $628,908$ keywords)


**(b)** English Wikipedia Corpus (all $4,400,034$ keywords)


**(c)** Enron Email Corpus (most frequent $5,000$ keywords)

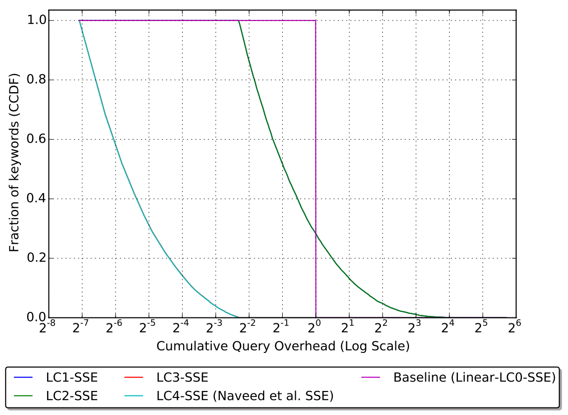

**(d)** English Wikipedia Corpus (most frequent $10,000$ keywords)

**Figure 8:** Cumulative Query Overhead using an ideal ORAM overhead of $\log(N)$, where $N$ is the total number of blocks, for both the documents ORAM and index ODICT. We assume that a single ODICT lookup requires a single ORAM access. Query overhead of a query is the ratio of the amount of data retrieved in an SSE scheme to the total size of all the outsourced documents. A point $(x, y)$ in the plots indicates that the query overhead is at least $x$ for $y$ fraction of the keywords.

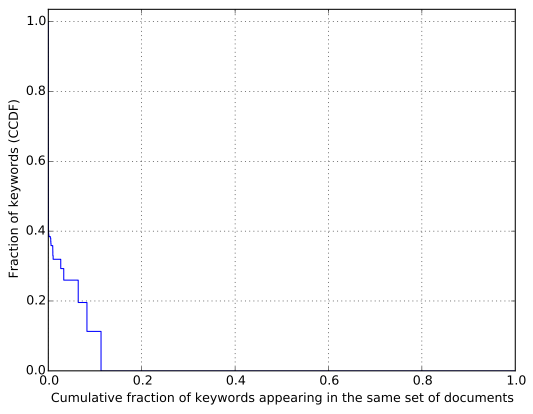


**Figure 10:** Leakage Advantage of $\mathcal{LC}3$–SSE over $\mathcal{LC}4$–SSE for Enron Email Corpus. A point $(x, y)$ in the plots indicates that the fraction of keywords that appear in the same set of documents is at least $x$ for $y$ fraction of the keywords.

**Oblivious RAM (ORAM).** Oblivious RAM was first proposed by Goldreich and Ostrovsky in 1996 [16] in the context of software protection. There has been a lot of progress since then in the efficiency of ORAM schemes [24, 9, 25, 13, 7, 21, 2, 17, 29, 30, 28, 10, 14, 11, 12] . Tree-based schemes deamortize the cost of shuffling to avoid an occasional linear shuffling overhead. Recently, ORAM has find applications in secure multiparty computation [33, 32, 13] and secure-coprocessor [22]. Oblivious RAM has also been used to develop oblivious data structure schemes such as tree and linked list [31].

**Symmetric Searchable Encryption (SSE).** Song et al. first proposed the idea of searching on encrypted data [26]. A lot of progress has been made in making SSE practical and many different schemes have been proposed [15, 5, 8, 6, 20, 20, 4, 19, 3, 23]. Curtmola et al. proposed better security definitions for SSE [8]. Searchable encryption scheme with support for updates was first proposed by Kamara et a. [20]. Naveed et al. proposed a different way of construction SSE schemes with support for updates [23]. Recently, Cash et al. proposed a first sublinear SSE scheme for Boolean queries [4].

Stefanov et al. proposed a forward secure SSE schemes that reduces the leakage in addition and deletion of documents, but leaks access pattern for queries just like other SSE schemes [27].

## 7. CONCLUSIONS

Oblivious RAM enables a client to access memory without leaking the access pattern. However, access pattern hiding properties of ORAM are limited to a random access memory (RAM) interface, which is not always the case with the real applications such as symmetric searchable encryption (SSE). We show that eliminating leakage in SSE is impossible. Moreover, achieving even weaker classes of leakage either requires communication more than downloading the entire outsourced data or does not provide meaningful leakage reduction.

## 8. REFERENCES

[1] Enron dataset. https://www.cs.cmu.edu/~enron/.

[2] D. Boneh, D. Mazieres, and R. A. Popa. Remote oblivious storage: Making oblivious ram practical. 2011.

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

[4] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *CRYPTO 2013*, pages 353–373. 2013.

[5] Y.-C. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *ACNS*, pages 442–455, 2005.

[6] M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *ASIACRYPT*, pages 577–594. 2010.

[7] K.-M. Chung and R. Pass. A simple oram. Technical report, DTIC Document, 2013.

[8] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *CCS*, pages 79–88, 2006.

[9] I. Damgård, S. Meldgaard, and J. B. Nielsen. Perfectly secure oblivious ram without random oracles. In *TCC*, pages 144–163. 2011.

[10] J. Dautrich, E. Stefanov, and E. Shi. Burst oram: Minimizing oram response times for bursty access patterns. In *USENIX Security*, volume 14, 2014.

[11] C. W. Fletcher, L. Ren, A. Kwon, M. Van Dijk, and S. Devadas. Freecursive oram:[nearly] free recursion and integrity verification for position-based oblivious ram. 2015.

[12] C. W. Fletcher, L. Ren, A. Kwon, M. Van Dijk, E. Stefanov, and S. Devadas. Tiny oram: A low-latency, low-area hardware oram controller with integrity verification. 2014.

[13] C. Gentry, K. A. Goldman, S. Halevi, C. Julta, M. Raykova, and D. Wichs. Optimizing oram and using it efficiently for secure computation. In *PETs*, pages 1–18, 2013.

[14] C. Gentry, S. Halevi, C. Jutla, and M. Raykova. Private database access with he-over-oram architecture. Technical report, IACR ePrint, 2014.

[15] E.-J. Goh et al. Secure indexes. *IACR ePrint*, page 216, 2003.

[16] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious rams. *JACM*, 43(3):431–473, 1996.

[17] M. T. Goodrich, M. Mitzenmacher, O. Ohrimenko, and R. Tamassia. Practical oblivious storage. In *CODASPY*, pages 13–24, 2012.

[18] M. S. Islam, M. Kuzu, and M. Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *NDSS*, 2012.

[19] S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Outsourced symmetric private information retrieval. In *CCS*, pages 875–888, 2013.

[20] S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *CCS*, pages 965–976, 2012.

[21] J. R. Lorch, B. Parno, J. W. Mickens, M. Raykova, and J. Schiffman. Shroud: ensuring private access to large-scale data in the data center. In *FAST*, pages 199–213, 2013.

[22] M. Maas, E. Love, E. Stefanov, M. Tiwari, E. Shi, K. Asanovic, J. Kubiatowicz, and D. Song. Phantom: Practical oblivious computation in a secure processor. In *CCS*, pages 311–324, 2013.

[23] M. Naveed, M. Prabhakaran, and C. A. Gunter. Dynamic searchable encryption via blind storage. *IEEE S&P*, 2014.

[24] B. Pinkas and T. Reinman. Oblivious ram revisited. In *CRYPTO*, pages 502–519. 2010.

[25] E. Shi, T.-H. H. Chan, E. Stefanov, and M. Li. Oblivious ram with $O((logN)^3)$ worst-case cost. In *ASIACRYPT*, pages 197–214. 2011.

[26] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *IEE S&P*, pages 44–55, 2000.

[27] E. Stefanov, C. Papamanthou, and E. Shi. Practical dynamic searchable encryption with small leakage. 2014.

[28] E. Stefanov and E. Shi. Multi-cloud oblivious storage. In *CCS*, pages 247–258, 2013.

[29] E. Stefanov and E. Shi. Oblivistore: High performance oblivious cloud storage. In *IEEE S&P*, pages 253–267, 2013.

[30] E. Stefanov, M. Van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas. Path oram: An extremely simple oblivious ram protocol. In *CCS*, pages 299–310, 2013.

[31] X. Wang, K. Nayak, C. Liu, E. Shi, E. Stefanov, and Y. Huang. Oblivious data structures. In *CCS*, page 185, 2014.

[32] X. S. Wang, T. H. Chan, and E. Shi. Circuit oram: On tightness of the goldreich-ostrovsky lower bound, 2014.

[33] X. S. Wang, Y. Huang, T. H. Chan, A. Shelat, and E. Shi. Scoram: Oblivious ram for secure computation. In *CCS*, 2014.

## APPENDIX

## A. PROOF SKETCHES

### A.1 Single Keyword $\mathcal{LC}1$–SSE Scheme

THEOREM A.1. *For given ORAM and ODICT schemes, $\mathcal{LC}1$–SSE scheme has optimal query communication required to achieve $\mathcal{LC}1$.*

We use ORAM as a blackbox in our scheme, therefore we use it as a blackbox in the proof as well.

PROOF. According to Lemma A.4, for a query $q$, $\mathcal{LC}1$–SSE needs to retrieve the list $L$ of document identifiers that satisfy the query $q$ and the set of documents $D$ that satisfy the query $q$.

To achieve Leakage Class 1 ($\mathcal{LC}1$), $\mathcal{LC}1$–SSE uses ODICT for the index accesses and ORAM for the document accesses.

To achieve $\mathcal{LC}1$, $\mathcal{LC}1$–SSE uses single ORAM storing index ODICT and the documents. ODICT for the index and ORAM for the documents are required to achieve $\mathcal{LC}1$, furthermore it is also required that index ODICT and documents are stored in the same ORAM. Otherwise, access pattern leakage from index and document accesses would leak strictly more information than $\mathcal{LC}1$. The overhead of using ODICT and ORAM is the only overhead incurred by $\mathcal{LC}1$–SSE. Therefore, we conclude that our single keyword $\mathcal{LC}1$–SSE has optimal communication for $\mathcal{LC}1$ leakage. □

### A.2 Single Keyword $\mathcal{LC}2$–SSE Scheme

THEOREM A.2. *For given ORAM and ODICT schemes, $\mathcal{LC}2$–SSE scheme has optimal query communication required to achieve $\mathcal{LC}2$.*

PROOF. According to Lemma A.4, for a query $q$, $\mathcal{LC}2$–SSE needs to retrieve the list $L$ of document identifiers that satisfy the query $q$ and the set of documents $D$ that satisfy the query $q$.

To achieve $\mathcal{LC}2$, $\mathcal{LC}2$–SSE use ODICT for the index accesses and ORAM for the document accesses. ODICT for the index and ORAM for the documents are required to achieve $\mathcal{LC}2$. Otherwise, access pattern leakage from index and document accesses would leak strictly more information than $\mathcal{LC}2$. The overhead of using ODICT and ORAM is the only overhead incurred by $\mathcal{LC}2$–SSE.

Therefore, we conclude that our single keyword $\mathcal{LC}2$–SSE has optimal communication for $\mathcal{LC}2$ leakage.
□

## A.3 Single Keyword $\mathcal{LC}3$–SSE Scheme

THEOREM A.3. *For a given ODICT construction, $\mathcal{LC}3$–SSE scheme has optimal query communication required to achieve $\mathcal{LC}3$.*

PROOF. According to Lemma A.4, for a query $q$, single keyword $\mathcal{LC}3$–SSE needs to retrieve the list $L$ of document identifiers that satisfy the query $q$ and the set of documents $D$ that satisfy the query $q$.

To achieve $\mathcal{LC}3$ the index needs to be stored in the ODICT, but the documents do *not* need any oblivious data structure. If index is *not* stored in and accessed from an ODICT then the leakage would be strictly more than $\mathcal{LC}3$. The overhead of ODICT is the only overhead incurred by $\mathcal{LC}3$–SSE. Therefore, we conclude that our single keyword $\mathcal{LC}3$–SSE has optimal communication for $\mathcal{LC}3$ leakage. □

## A.4 Sublinear Communication

LEMMA A.4. *For a computation-less server and a client with* $\text{poly}\log(|\mathbb{D}|)$ *local storage, a query q in an $\mathcal{LC}X$–SSE schemes, the minimum amount of communication requires retrieving the list of document identifiers that satisfy the query q and set of documents $D$ that satisfy the query q.*

PROOF. Let $|\mathbb{D}|$ be the total size of all outsourced documents. Let the client query be $q$ and let $D$ be the documents that satisfy the query $q$.

We require an $\mathcal{LC}X$–SSE scheme to have better performance than our baseline Linear–$\mathcal{LC}0$–SSE. Linear–$\mathcal{LC}0$–SSE has communication linear in the total size of all outsourced data, so $\mathcal{LC}X$–SSE scheme is by definition a sublinear scheme. Any sublinear scheme, first requires to find out which documents (e.g., finding the document identifiers) match the query and then retrieve these documents. Client has limited local storage i.e., $\text{poly}\log(|\mathbb{D}|)$; this means that the client cannot locally find out the document identifiers that match the query as it would require local storage linear in $|\mathbb{D}|$. So, this information needs to be stored on the server and retrieved for each query.

For an $\mathcal{LC}X$–SSE scheme to be correct[4], the minimum amount of communication needs to be the complete list of document identifiers for all documents that satisfy the query (let's call this list $L$), otherwise client would *not* be able to download all the documents. Next, the client needs to download all the documents in the list $L$, that is all the documents that satisfy the query. For an $\mathcal{LC}X$–SSE scheme to be correct, client has to download all the documents $D$. So, we need to retrieve the list of document identifiers $L$ and all the documents $D$. □

[4]Correctness means that the client downloads all the documents that match the query.