Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions *

Reza Curtmola [†] NJIT

Juan Garay [‡] AT&T Labs – Research

Seny Kamara [§] Microsoft Research

Rafail Ostrovsky [¶] UCLA

Abstract

Searchable symmetric encryption (SSE) allows a party to outsource the storage of his data to another party in a private manner, while maintaining the ability to selectively search over it. This problem has been the focus of active research and several security definitions and constructions have been proposed. In this paper we begin by reviewing existing notions of security and propose new and stronger security definitions. We then present two constructions that we show secure under our new definitions. Interestingly, in addition to satisfying stronger security guarantees, our constructions are more efficient than all previous constructions.

Further, prior work on SSE only considered the setting where only the owner of the data is capable of submitting search queries. We consider the natural extension where an arbitrary group of parties other than the owner can submit search queries. We formally define SSE in this *multi-user* setting, and present an efficient construction.

1 Introduction

Private-key storage outsourcing [30, 4, 33] allows clients with either limited resources or limited expertise to store and distribute large amounts of symmetrically encrypted data at low cost. Since regular private-key encryption prevents one from searching over encrypted data, clients also lose the ability to selectively retrieve segments of their data. To address this, several techniques have been proposed for provisioning symmetric encryption with search capabilities [40, 23, 10, 18]; the resulting construct is typically called *searchable encryption*. The area of searchable encryption has been identified by DARPA as one of the technical advances that can be used to balance the need for both privacy and national security in information aggregation systems [1].

One approach to provisioning symmetric encryption with search capabilities is with a so-called *secure index* [23]. An index is a data structure that stores document collections while supporting efficient keyword search, i.e., given a keyword, the index returns a pointer to the documents that contain it. Informally, an index is "secure" if the search operation for a keyword w can only be performed by users that possess a "trapdoor" for w and if the trapdoor can only be generated with a secret key. Without knowledge of trapdoors, the index leaks no information about its contents. As shown by Goh in [23], one can build a symmetric searchable encryption scheme from a secure

*A preliminary version of this article appeared in the 13^{th} ACM Conference on Computer and Communications Security (CCS '06) [20].

[†] crix@njit.edu. Work done in part while at Bell Labs and Johns Hopkins University.

[‡] garay@research.att.com. Work done in part while at Bell Labs.

[§] senyk@microsoft.com. Work done in part while at Johns Hopkins University.

[¶] rafail@cs.ucla.edu.

index as follows: the client indexes and encrypts its document collection and sends the secure index together with the encrypted data to the server. To search for a keyword w, the client generates and sends a trapdoor for w which the server uses to run the search operation and recover pointers to the appropriate (encrypted) documents.

Symmetric searchable encryption can be achieved in its full generality and with optimal security using the work of Ostrovsky and Goldreich on *oblivious RAMs* [35, 25]. More precisely, using these techniques any type of search query can be achieved (e.g., conjunctions or disjunctions of keywords) without leaking *any* information to the server, not even the "access pattern" (i.e., which documents contain the keyword). This strong privacy guarantee, however, comes at the cost of a logarithmic (in the number of documents) number of rounds of interaction for each read and write. In the same paper, the authors show a 2-round solution, but with considerably larger square-root overhead. Therefore, the previously mentioned work on searchable encryption [40, 23, 10, 18] tries to achieve more efficient solutions (typically in one or two rounds) by weakening the privacy guarantees.

1.1 Our contributions

We now give an overview of the contributions of this work.

Revisiting previous definitions. We review existing security definitions for secure indexes, including indistinguishability against chosen-keyword attacks (IND2-CKA) [23] and the simulation-based definition in [18], and highlight some of their limitations. Specifically, we recall that IND2-CKA does not guarantee the privacy of user queries (and is therefore not an adequate notion of security for constructing SSE schemes) and then highlight (and fix) technical issues with the simulation-based definition of [18]. We address both these issues by proposing new game-based and simulation-based definitions that provide security for both indexes and trapdoors.

New definitions. We introduce new adversarial models for SSE. The first, which we refer to as *non-adaptive*, only considers adversaries that make their search queries without taking into account the trapdoors and search outcomes of previous searches. The second—*adaptive*—considers adversaries that choose their queries as a function of previously obtained trapdoors and search outcomes. All previous work on SSE (with the exception of oblivious RAMs) falls within the non-adaptive setting. The implication is that, contrary to the natural use of searchable encryption described in [40, 23, 18], these definitions only guarantee security for users that perform all their searches *at once*. We address this by introducing game-based and simulation-based definitions in the adaptive setting.

New constructions. We present two constructions which we prove secure under our new definitions. Our first scheme is only secure in the non-adaptive setting, but is the most efficient SSE construction to date. In fact, it achieves searches in one communication round, requires an amount of work from the server that is linear in the number of documents that contain the keyword (which is optimal), requires constant storage on the client, and linear (in the size of the document collection) storage on the server. While the construction in [23] also performs searches in one round, it can induce false positives, which is not the case for our construction. Additionally, all the constructions in [23, 18] require the server to perform an amount of work that is linear in the *total* number of documents in the collection.

Our second construction is secure against an adaptive adversary, but at the price of requiring a higher communication overhead per query and more storage at the server (comparable with the storage required in [23]). While our adaptive scheme is conceptually simple, we note that constructing efficient and provably secure adaptive SSE schemes is a non-trivial task. The main challenge lies in proving such constructions secure in the simulation paradigm, since the simulator requires the ability

Properties	[35, 25]	[35, 25]-light	[40]	[23]	[18]	SSE-1	SSE-2
hides access pattern	yes	yes	no	no	no	no	no
server computation	$O(\log^3 n)$	$O(\sqrt{n})$	$O(n)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$
server storage	$O(n \cdot \log n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
number of rounds	$\log n$	2	1	1	1	1	1
communication	$O(\log^3 n)$	$O(\sqrt{n})$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
adaptive adversaries	yes	yes	no	no	no	no	yes

Table 1: Properties and performance (per query) of various SSE schemes. n denotes the number of documents in the collection. For communication costs, we consider only the overhead and omit the size of the retrieved documents, which is the same for all schemes. For server computation, we show the costs per returned document. For simplicity, the security parameter is not included as a factor for the relevant costs.

to commit to a correct index before the adversary has even chosen its search queries—in other words, the simulator needs to commit to an index and then be able to perform some form of equivocation.

Table 1 compares our constructions (SSE-1 and SSE-2) with the previous SSE schemes. To make the comparison easier, we assume that each document in the collection has the same (constant) size (otherwise, some of the costs have to be scaled by the document size). The server computation row shows the costs per returned document for a query. Note that all previous work requires an amount of server computation at least linear with the number of documents in the collection, even if only one document matches a query. In contrast, in our constructions the server computation is constant per each document that matches a query, and the overall computation per query is proportional to the number of documents that match the query. In all the considered schemes, the computation and storage at the user is $O(1)$.

We remark that as an additional benefit, our constructions can also handle updates to the document collection in the sense of [18]. We point out an optimization which lowers the communication complexity per query from linear to logarithmic in the number of updates.

Multi-user SSE. Previous work on searchable encryption only considered the single-user setting. We also consider a natural extension of this setting, namely, the *multi-user* setting, where a user owns the data, but an arbitrary group of users can submit queries to search his document collection. The owner can control the search access by granting and revoking searching privileges to other users. We formally define searchable encryption in the multi-user setting, and present an efficient construction that does not require authentication, thus achieving better performance than simply using access control mechanisms.

Finally, we note that in most of the works mentioned above the server is assumed to be honest-but-curious. However, using techniques for memory checking [14] and universal arguments [7] one can make those solutions robust against malicious servers at the price of additional overhead. We restrict our attention to honest-but-curious servers as well.

1.2 On different models for private search

Before providing a detailed comparison to existing work, we put our work in context by providing a classification of the various models for privacy-preserving search. In recent years, there has been some confusion regarding three distinct models: searching on *private-key* encrypted data (which is the subject of this work); searching on *public-key* encrypted data; and single-database *private information retrieval* (PIR).

Common to all three models is a server (sometimes called the "database") that stores data, and a user that wishes to access, search, or modify the data while revealing as little as possible to the server. There are, however, important differences between these three settings.

Private-key searchable encryption. In the setting of *searching on private-key*-encrypted data, the user himself encrypts the data, so he can organize it in an arbitrary way (before encryption) and include additional data structures to allow for efficient access of relevant data. The data and the additional data structures can then be encrypted and stored on the server so that only someone with the private key can access it. In this setting, the initial work for the user (i.e., for preprocessing the data) is at least as large as the data, but subsequent work (i.e., for accessing the data) is very small relative to the size of the data for both the user and the server. Furthermore, everything about the user's access pattern can be hidden [35, 25].

Public-key searchable encryption. In the setting of *s*earching on public-key-encrypted data, users who encrypt the data (and send it to the server) can be different from the owner of the decryption key. In a typical application, a user publishes a public key while multiple senders send e-mails to the mail server [15, 2]. Anyone with access to the public key can add words to the index, but only the owner of the private key can generate "trapdoors" to test for the occurrence of a keyword. Although the original work on public-key encryption with keyword search (PEKS) by Boneh, di Crescenzo, Ostrosvky and Persiano [15] reveals the user's access pattern, Boneh, Kushilevitz, Ostrovsky and Skeith [16] have shown how to build a public-key encryption scheme that hides even the access pattern. This construction, however, has an overhead in search time that is proportional to the square root of the database size, which is far less efficient then the best private-key solutions.

Recently, Bellare, Boldyreva and O'Neill [8] introduced the notion of public key *efficiently searchable encryption* (ESE) and proposed constructions in the random oracle model. Unlike PEKS, ESE schemes allow anyone with access to a user's public key to add words to the index *and* to generate trapdoors to search. While ESE schemes achieve optimal search time (same as our constructions – see below), they are inherently deterministic and therefore provide security guarantees that are weaker than the ones considered in this work.

Single-database PIR. In single-database private information retrieval (or PIR), introduced by Kushilevitz and Ostrovsky [31], a user can retrieve data from a server containing *unencrypted* data without revealing the access pattern and with total communication less then the data size. This was extended to keyword searching, including searching on streaming data [36]. We note, however, that since the data in PIR is always unencrypted, any scheme that tries to hide the access pattern must touch all data items. Otherwise, the server learns information: namely, that the untouched item was not of interest to the user. Thus, PIR schemes require work which is linear in the database size. Of course, one can amortize this work for multiple queries and multiple users in order to save work of the database per query, as shown in [27, 28], but the key feature of all PIR schemes is that the data is always unencrypted, unlike the previous two settings on searching on *encrypted* data.

1.3 Versions of this Paper

This is the full version of [20] and includes all omitted proofs and several improvements. Following [19], the definition of SSE used in this version explicitly captures the encryptions of the documents. Using the terminology of [19], we consider *pointer-output* SSE schemes as opposed to [20] which considered *structure-only* schemes. While most previous work on SSE considers only the latter (ignoring how the documents are encrypted), we prefer the former definition of SSE. Another difference with [20] is in our treatment of multi-user SSE. Here, we describe the algorithms of a multi-user SSE scheme

as stateful which allows us to provide a "cleaner" description of our construction. Finally, we note that the simulation-based definitions used in this work (i.e., Definitions 4.8 and 4.11) differ from the definitions that appeared in a preliminary full version of this paper (i.e., Definitions 3.6 and 3.9 in [21]). We believe that the formulations provided here are easier to work with and intuitively more appealing.

2 Related Work

We already mentioned the work on oblivious RAMs [35, 25]. In an effort to reduce the round complexity associated with oblivious RAMs, Song, Wagner and Perrig [40] showed that a solution for searchable encryption was possible for a weaker security model. Specifically, they achieve searchable encryption by crafting, for each word, a special two-layered encryption construct. Given a trapdoor, the server can strip the outer layer and assert whether the inner layer is of the correct form. This construction, however, has some limitations: while the construction is proven to be a secure encryption scheme, it is not proven to be a secure *searchable* encryption scheme; the distribution of the underlying plaintexts is vulnerable to statistical attacks; and searching is linear in the length of the document collection.

The above limitations are addressed by the works of Goh [23] and of Chang and Mitzenmacher [18], who propose constructions that associate an "index" to each document in a collection. As a result, the server has to search each of these indexes, and the amount of work required for a query is proportional to the number of documents in the collection. Goh introduces a notion of security for indexes (IND-CKA and the slightly stronger IND2-CKA), and puts forth a construction based on Bloom filters [13] and pseudo-random functions. Chang and Mitzenmacher achieve a notion of security similar to IND2-CKA, except that it also tries to guarantee that the trapdoors not leak any information about the words being queried. We discuss these security definitions and their limitations in more detail in Section 4 and Appendix B.

As mentioned above, encryption with keyword search has also been considered in the public-key setting [15, 2], where anyone with access to a user's public-key can add words to an index, but only the owner of the private-key can generate trapdoors to test for the occurrence of a keyword. While related, the public-key solutions are suitable for different applications and are not as efficient as private-key solutions, which is the main subject of this work. Public key efficiently searchable encryption (ESE) [8] achieves efficiency comparable to ours, but at the price of providing weaker security guarantees. The notion of ESE, originally proposed in a public key setting was extended to the symmetric key setting [5], which views the outsourced data as a relational database and seeks to achieve query-processing efficiency comparable to that for unencrypted databases. These schemes sacrifice security in order to preserve general efficiency and functionality: Similar to our work, the efficiency of operations on encrypted and unencrypted databases are comparable; unlike our work, this comes at the cost of weakening the security definition (in addition to revealing the user's query access pattern, the frequency distribution of the plaintext data is also revealed to the server prior to any client queries). Further, we also note that the notion of multi-user SSE—which we introduce in this work—combined with a classical public-key encryption scheme, achieves a functionality similar to that of public key ESE, with the added benefit of allowing the owner to revoke search privileges.

Whereas this work focuses on the case of single-keyword equality queries, we note that more complex queries have also been considered. This includes conjunctive queries in the symmetric key setting [26, 6]; it also includes conjunctive queries [37, 17], comparison and subset queries [17], and range queries [39] in the public-key setting.

Unlike the above mentioned work on searchable encryption that relies on computational assumptions, Sedghi *et al.* [38] propose a model that targets an information theoretic security analysis.

Naturally, SSE can also be viewed as an instance of secure two-party/multi-party computation [41, 24, 11]. However, the weakening and refinement of the privacy requirement (more on this below) as

well as efficiency considerations (e.g., [29]), mandate a specialized treatment of the problem, both at the definitional and construction levels.[1]

A different notion of privacy is considered by Narayanan and Shmatikov [34], who propose schemes for obfuscating a database so that only certain queries can be evaluated on it. However, their goal is not to hide data from an untrusted server, but to transform the database such that it prevents users that do not abide by the privacy policy from querying the database.

3 Notation and Preliminaries

We write $x \leftarrow \chi$ to represent an element x being sampled from a distribution χ, and $x \xleftarrow{\$} X$ to represent an element x being sampled uniformly from a set X. The output x of an algorithm $\mathcal{A}$ is denoted by $x \leftarrow \mathcal{A}$. We write $a||b$ to refer to the concatenation of two strings a and b. Let $\mathsf{Func}[n, m]$ be the set of all functions from $\{0,1\}^n$ to $\{0,1\}^m$. Throughout, k will refer to the security parameter and we will assume that all algorithms take it as input. A function $\nu : \mathbb{N} \to \mathbb{N}$ is negligible in k if for every positive polynomial $p(\cdot)$ and sufficiently large k, $\nu(k) < 1/p(k)$. Let $\mathsf{poly}(k)$ and $\mathsf{negl}(k)$ denote unspecified polynomial and negligible functions in k, respectively.

In this work, honest users are modeled as probabilistic polynomial-time Turing machines, while adversaries and simulators are modeled as (deterministic) polynomial-size circuits. As every probabilistic polynomial-time algorithm can be simulated by a (deterministic) polynomial-size circuit [3], our schemes guarantee security against any *probabilistic* polynomial-time adversary.

Document collections. Let $\Delta = (w_1, \ldots, w_d)$ be a dictionary of d words in lexicographic order, and 2^{Δ} be the set of all possible documents with words in Δ. We assume $d = \mathsf{poly}(k)$ and that all words $w \in \Delta$ are of length polynomial in k. Furthermore, let $\mathbf{D} \subseteq 2^{\Delta}$ be a collection of $n = \mathsf{poly}(k)$ documents $\mathbf{D} = (D_1, \ldots, D_n)$, each containing $\mathsf{poly}(k)$ words. Let $\mathsf{id}(D)$ be the identifier of document D, where the identifier can be any string that uniquely identifies a document such as a memory location. We denote by $\mathbf{D}(w)$ the lexicographically ordered list consisting of the identifiers of all documents in $\mathbf{D}$ that contain the word w.

Symmetric encryption. A symmetric encryption scheme is a set of three polynomial-time algorithms $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ such that Gen takes a security parameter k and returns a secret key K; Enc takes a key K and a message m and returns a ciphertext c; Dec takes a key K and a ciphertext c and returns m if K was the key under which c was produced. Intuitively, a symmetric encryption scheme is secure against chosen-plaintext attacks (CPA) if the ciphertexts it outputs do not leak any useful information about the plaintext even to an adversary that can query an encryption oracle. In this work, we consider a stronger notion, which we refer to as pseudo-randomness against chosen-plaintext attacks (PCPA), that guarantees that the ciphertexts are indistinguishable from random (a formal definition is provided in Appendix A). We note that common private-key encryption schemes such as AES in counter mode satisfy this definition.

Pseudo-random functions. In addition to encryption schemes, we also make use of pseudo-random functions (PRF) and permutations (PRP), which are polynomial-time computable functions that cannot be distinguished from random functions by any probabilistic polynomial-time adversary (see Appendix A for a formal definition).

[1] Indeed, some of the results we show—equivalence of SSE security definitions (Section 4)—are known not to hold for the general secure multi-party computation case.

Broadcast encryption. A broadcast encryption scheme is tuple of four polynomial-time algorithms $\mathsf{BE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Add}, \mathsf{Dec})$ that work as follows. Let $\mathcal{U}$ be BE's user space, i.e., the set of all possible user identifiers. Gen is a probabilistic algorithm that takes as input a security parameter k and outputs a master key mk. Enc is a probabilistic algorithm that takes as input a master key mk, a set of users $G \subseteq \mathcal{U}$ and a message m, and outputs a ciphertext c. Add is a probabilistic algorithm that takes as input a master key mk and a user identifier $U \in \mathcal{U}$, and outputs a user key uk_U. Finally, Dec is a deterministic algorithm that takes as input a user key uk_U and a ciphertext c and outputs either a message m or the failure symbol $\perp$. Informally, a broadcast encryption scheme is secure if its ciphertexts leak no useful information about the message to any user not in G.

4 Definitions for Searchable Symmetric Encryption

We begin by reviewing the formal definition of an index-based SSE scheme. The participants in a single-user SSE scheme include a client that wants to store a private document collection $\mathbf{D} = (D_1, \ldots, D_n)$ on an honest-but-curious server in such a way that (1) the server will not learn any useful information about the collection; and that (2) the server can be given the ability to search through the collection and return the appropriate (encrypted) documents to the client. We consider searches to be over documents but, of course, any SSE scheme as described below can be used with collections of arbitrary files (e.g., images or audio files) as long as the files are labeled with keywords.

Definition 4.1 (Searchable symmetric encryption)**.** *An index-based SSE scheme over a dictionary Δ is a collection of five polynomial-time algorithms $\mathsf{SSE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Trpdr}, \mathsf{Search}, \mathsf{Dec})$ such that,*

$K \leftarrow \mathsf{Gen}(1^k)$*: is a probabilistic key generation algorithm that is run by the user to setup the scheme. It takes as input a security parameter k, and outputs a secret key K.*

$(I, \boldsymbol{c}) \leftarrow \mathsf{Enc}(K, \mathbf{D})$*: is a probabilistic algorithm run by the user to encrypt the document collection. It takes as input a secret key K and a document collection $\mathbf{D} = (D_1, \ldots, D_n)$, and outputs a secure index I and a sequence of ciphertexts $\boldsymbol{c} = (c_1, \ldots, c_n)$. We sometimes write this as $(I, \boldsymbol{c}) \leftarrow \mathsf{Enc}_K(\mathbf{D})$.*

$t \leftarrow \mathsf{Trpdr}(K, w)$*: is a deterministic algorithm run by the user to generate a trapdoor for a given keyword. It takes as input a secret key K and a keyword w, and outputs a trapdoor t. We sometimes write this as $t \leftarrow \mathsf{Trpdr}_K(w)$.*

$X \leftarrow \mathsf{Search}(I, t)$*: is a deterministic algorithm run by the server to search for the documents in $\mathbf{D}$ that contain a keyword w. It takes as input an encrypted index I for a data collection $\mathbf{D}$ and a trapdoor t and outputs a set X of (lexicographically-ordered) document identifiers.*

$D_i \leftarrow \mathsf{Dec}(K, c_i)$*: is a deterministic algorithm run by the client to recover a document. It takes as input a secret key K and a ciphertext c_i, and outputs a document D_i. We sometimes write this as $D_i \leftarrow \mathsf{Dec}_K(c_i)$.*

An index-based SSE scheme is correct if for all $k \in \mathbb{N}$, for all K output by $\mathsf{Gen}(1^k)$, for all $\mathbf{D} \subseteq 2^{\Delta}$, for all $(I, \boldsymbol{c})$ output by $\mathsf{Enc}_K(\mathbf{D})$, for all $w \in \Delta$,

$$\mathsf{Search}\big(I, \mathsf{Trpdr}_K(w)\big) = \mathbf{D}(w) \bigwedge \mathsf{Dec}_K(c_i) = D_i, \text{ for } 1 \leq i \leq n.$$

4.1 Revisiting searchable symmetric encryption definitions

While security for searchable encryption is typically characterized as the requirement that nothing be leaked beyond the "outcome of a search" or the "access pattern" (i.e., the identifiers of the documents that contain a keyword), we are not aware of any previous work other than that of [25, 35] that satisfies this intuition. In fact, with the exception of oblivious RAMs, all the constructions in the literature also reveal whether searches were for the same word or not. We refer to this as the *search pattern* and note that it is clearly revealed by the schemes presented in [40, 23, 18] since their trapdoors are deterministic. Therefore, a more accurate characterization of the security notion achieved for SSE is that nothing is leaked beyond the *access pattern* and the *search pattern* (precise definitions in Section 4.2).

Having clarified our intuition, it remains to precisely describe our adversarial model. SSE schemes based on secure indexes are typically used in the following manner: the client generates a secure index from its document collection, sends the index and the encrypted documents to the server and, finally, performs various search queries by sending trapdoors for a given set of keywords. Here, it is important to note that the user may or may not generate its keywords as a function of the outcome of previous searches. We call queries that do depend on previous search outcomes *adaptive*, and queries that do not, *non-adaptive*. This distinction in keyword generation is important because it gives rise to definitions that achieve different privacy guarantees: non-adaptive definitions only provide security to clients that generate their keywords in one batch, while adaptive definitions provide privacy even to clients who generate keywords as a function of previous search outcomes. The most natural use of searchable encryption is for making adaptive queries.

Limitations of previous definitions. To date, two definitions of security have been used for SSE: indistinguishability against chosen-keyword attacks (IND2-CKA), introduced by Goh [23][2], and a simulation-based definition introduced by Chang and Mitzenmacher [18].[3]

Intuitively, the security guarantee that IND2-CKA achieves can be described as follows: given access to an index, the adversary (i.e., the server) is not able to learn any partial information about the underlying documents that he cannot learn from using a trapdoor that was given to him by the client, and this holds even against adversaries that can convince the client to generate indexes and trapdoors for documents and keywords chosen by the adversary (i.e., chosen-keyword attacks). A formal specification of IND2-CKA is presented in Appendix B.

We remark that Goh's work addresses the problem of secure indexes which have many uses, only one of which is searchable encryption. And as Goh remarks (*cf.* Note 1, p. 5 of [23]), IND2-CKA does not explicitly require that trapdoors be secure since this is not a requirement for all applications of secure indexes.

Although one might be tempted to remedy the situation by introducing a second definition to guarantee that trapdoors not leak any information, this cannot be done in a straightforward manner. Indeed, as we show in Appendix B, proving that an SSE scheme is IND2-CKA and then proving that its trapdoors are secure (in a sense made precise in Appendix B) *does not* imply that an adversary cannot recover the word being queried (a necessary requirement for searchable encryption).

Regarding existing simulation-based definitions, Chang and Mitzenmacher present a security definition for SSE in [18] that is intended to be stronger than IND2-CKA in the sense that it requires secure trapdoors. Unfortunately, as we also show in Appendix B, this definition can be trivially satisfied by any SSE scheme, even one that is insecure. Moreover, this definition is inherently non-adaptive.

[2] Goh also defines a weaker notion, IND-CKA, that allows an index to leak the number of words in the document.

[3] We note that, unlike the latter and our own definitions, IND2-CKA applies to indexes that are built for individual documents, as opposed to indexes built from entire document collections.

4.2 Our security definitions

We now address the above issues. Before stating our definitions for SSE, we introduce four auxiliary notions which we make use of. The interaction between the client and server is determined by a document collection and a sequence of keywords that the client wants to search for and that we wish to hide from the adversary. We call an instantiation of such an interaction a *history*.

Definition 4.2 (History)**.** *Let Δ be a dictionary and $\mathbf{D} \subseteq 2^{\Delta}$ be a document collection over Δ. A q-query* history *over $\mathbf{D}$ is a tuple $H = (\mathbf{D}, \boldsymbol{w})$ that includes the document collection $\mathbf{D}$ and a vector of q keywords $\boldsymbol{w} = (w_1, \ldots, w_q)$.*

Definition 4.3 (Access Pattern)**.** *Let Δ be a dictionary and $\mathbf{D} \subseteq 2^{\Delta}$ be a document collection over Δ. The access pattern induced by a q-query history $H = (\mathbf{D}, \boldsymbol{w})$, is the tuple $\alpha(H) = (\mathbf{D}(w_1), \ldots, \mathbf{D}(w_q))$.*

Definition 4.4 (Search Pattern)**.** *Let Δ be a dictionary and $\mathbf{D} \subseteq 2^{\Delta}$ be a document collection over Δ. The search pattern induced by a q-query history $H = (\mathbf{D}, \boldsymbol{w})$, is a symmetric binary matrix $\sigma(H)$ such that for $1 \leq i, j \leq q$, the element in the i^{th} row and j^{th} column is 1 if $w_i = w_j$, and 0 otherwise.*

The final notion is that of the *trace* of a history, which consists of exactly the information we are willing to leak about the history and nothing else. More precisely, this should include the identifiers of the documents that contain each keyword in the history, and information that describes which trapdoors correspond to the same underlying keywords in the history. According to our intuitive formulation of security this should be no more than the access and search patterns. However, since in practice the encrypted documents will also be stored on the server, we can assume that the document sizes and identifiers will also be leaked. Therefore we choose to include these in the trace.[4]

Definition 4.5 (Trace)**.** *Let Δ be a dictionary and $\mathbf{D} \subseteq 2^{\Delta}$ be a document collection over Δ. The trace induced by a q-query history $H = (\mathbf{D}, \boldsymbol{w})$, is a sequence $\tau(H) = (|D_1|, \ldots, |D_n|, \alpha(H), \sigma(H))$ comprised of the lengths of the documents in $\mathbf{D}$, and the access and search patterns induced by H.*

Throughout this work, we will assume that the dictionary Δ and the trace are such that all histories H over Δ are *non-singular* as defined below.

Definition 4.6 (Non-singular history)**.** *We say that a history H is non-singular if* (1) *there exists at least one history $H' \neq H$ such that $\tau(H) = \tau(H')$; and if* (2) *such a history can be found in polynomial-time given $\tau(H)$.*

Note that the existence of a second history with the same trace is a necessary assumption, otherwise the trace would immediately leak all information about the history.

4.2.1 Non-adaptive security for SSE

We are now ready to state our first security definition for SSE. First, we assume that the adversary generates the histories at once. In other words, it is not allowed to see the index of the document collection or the trapdoors of any keywords it chooses before it has finished generating the history. We call such an adversary *non-adaptive*.

Definition 4.7 (Non-adaptive indistinguishability)**.** *Let $\mathsf{SSE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Trpdr}, \mathsf{Search}, \mathsf{Dec})$ be an index-based* SSE *scheme over a dictionary Δ, $k \in \mathbb{N}$ be the security parameter, and $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a non-uniform adversary and consider the following probabilistic experiment $\mathbf{Ind}_{\mathsf{SSE},\mathcal{A}}(k)$:*

[4] On the other hand, if we wish not to disclose the size of the documents, this can be easily achieved by "padding" each plaintext document such that all documents have a fixed size and omitting the document sizes from the trace.

$\mathbf{Ind}_{\mathsf{SSE},\mathcal{A}}(k)$
$K \leftarrow \mathsf{Gen}(1^k)$
$(st_{\mathcal{A}}, H_0, H_1) \leftarrow \mathcal{A}_1(1^k)$
$b \xleftarrow{\$} \{0,1\}$
parse H_b *as* $(\mathbf{D}_b, \boldsymbol{w}_b)$
$(I_b, \boldsymbol{c}_b) \leftarrow \mathsf{Enc}_K(\mathbf{D}_b)$
for $1 \leq i \leq q$,
  $t_{b,i} \leftarrow \mathsf{Trpdr}_K(w_{b,i})$
let $\boldsymbol{t}_b = (t_{b,1}, \ldots, t_{b,q})$
$b' \leftarrow \mathcal{A}_2(st_{\mathcal{A}}, I_b, \boldsymbol{c}_b, \boldsymbol{t}_b)$
if $b' = b$, *output* 1
otherwise output 0

with the restriction that $\tau(H_0) = \tau(H_1)$*, and where* $st_{\mathcal{A}}$ *is a string that captures* $\mathcal{A}_1$*'s state. We say that* SSE *is secure in the sense of* non-adaptive indistinguishability *if for all polynomial-size adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*,*

$$\Pr[\,\mathbf{Ind}_{\mathsf{SSE},\mathcal{A}}(k) = 1\,] \leq \frac{1}{2} + \mathsf{negl}(k),$$

where the probability is taken over the choice of b *and the coins of* Gen *and* Enc*.*

Note that, unlike the notion of IND2-CKA [23], our definition does not give the adversary access to an Enc or a Trpdr oracle. This, however, *does not* weaken our security guarantee in any way. The reason oracle access is not necessary is because our definition of SSE is formulated with respect to document *collections*, as opposed to individual documents, and therefore it is sufficient for security to hold for a single use.

Our simulation-based definition requires that the view of an adversary (i.e., the index, the ciphertexts and the trapdoors) generated from an adversarially and non-adaptively chosen history be simulatable given only the trace.

Definition 4.8 (Non-adaptive semantic security)**.** *Let* $\mathsf{SSE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Trpdr}, \mathsf{Search}, \mathsf{Dec})$ *be an index-based* SSE *scheme,* $k \in \mathbb{N}$ *be the security parameter,* $\mathcal{A}$ *be an adversary,* $\mathcal{S}$ *be a simulator and consider the following probabilistic experiments:*

$\mathbf{Real}_{\mathsf{SSE},\mathcal{A}}(k)$
$K \leftarrow \mathsf{Gen}(1^k)$
$(st_{\mathcal{A}}, H) \leftarrow \mathcal{A}(1^k)$
parse H *as* $(\mathbf{D}, \boldsymbol{w})$
$(I, \boldsymbol{c}) \leftarrow \mathsf{Enc}_K(\mathbf{D})$
for $1 \leq i \leq q$,
  $t_i \leftarrow \mathsf{Trpdr}_K(w_i)$
let $\boldsymbol{t} = (t_1, \ldots, t_q)$
output $\mathrm{v} = (I, \boldsymbol{c}, \boldsymbol{t})$ *and* $st_{\mathcal{A}}$

$\mathbf{Sim}_{\mathsf{SSE},\mathcal{A},\mathcal{S}}(k)$
$(H, st_{\mathcal{A}}) \leftarrow \mathcal{A}(1^k)$
$\mathrm{v} \leftarrow \mathcal{S}(\tau(H))$
output v *and* $st_{\mathcal{A}}$

We say that SSE *is semantically secure if for all polynomial-size adversaries* $\mathcal{A}$*, there exists a polynomial-size simulator* $\mathcal{S}$ *such that for all polynomial-size distinguishers* $\mathcal{D}$*,*

$$|\Pr[\,\mathcal{D}(\mathrm{v}, st_{\mathcal{A}}) = 1 : (\mathrm{v}, st_{\mathcal{A}}) \leftarrow \mathbf{Real}_{\mathsf{SSE},\mathcal{A}}(k)\,] - \Pr[\,\mathcal{D}(\mathrm{v}, st_{\mathcal{A}}) = 1 : (\mathrm{v}, st_{\mathcal{A}}) \leftarrow \mathbf{Sim}_{\mathsf{SSE},\mathcal{A},\mathcal{S}}(k)\,]| \leq \mathsf{negl}(k),$$

where the probabilities are over the coins of Gen *and* Enc*.*

We now prove that our two definitions of security for non-adaptive adversaries are equivalent.

Theorem 4.9. *Non-adaptive indistinguishability security of SSE is equivalent to non-adaptive semantic security of SSE.*

Proof. Let $\mathsf{SSE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Trpdr}, \mathsf{Search}, \mathsf{Dec})$ be an index-based SSE scheme. We make the following two claims, from which the theorem follows.

Claim. If SSE is non-adaptively semantically secure for SSE, then it is non-adaptively indistingishable for SSE.

We show that if there exists a polynomial-size adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ that succeeds in an $\mathbf{Ind}_{\mathsf{SSE},\mathcal{A}}(k)$ experiment with non-negligible probability over $1/2$, then there exists a polynomial-size adversary $\mathcal{B}$ and a polynomial-size distinguisher $\mathcal{D}$ such that for all polynomial-size simulators $\mathcal{S}$, $\mathcal{D}$ distinguishes between the output of $\mathbf{Real}_{\mathsf{SSE},\mathcal{B}}(k)$ and $\mathbf{Sim}_{\mathsf{SSE},\mathcal{B},\mathcal{S}}(k)$.

Let $\mathcal{B}$ be the adversary that computes $(st_{\mathcal{A}}, H_0, H_1) \leftarrow \mathcal{A}_1(1^k)$; samples $b \xleftarrow{\$} \{0,1\}$; and outputs the history H_b and state $st_{\mathcal{B}} = (st_{\mathcal{A}}, b)$. Let $\mathcal{D}$ be the distinguisher that, given v and $st_{\mathcal{B}}$ (which are either output by $\mathbf{Real}_{\mathsf{SSE},\mathcal{B}}(k)$ or $\mathbf{Sim}_{\mathsf{SSE},\mathcal{S},\mathcal{B}}(k)$), works as follows:

1. it parses $st_{\mathcal{B}}$ into $(st_{\mathcal{A}}, b)$ and v into $(I, \mathbf{c}, \mathbf{t})$,
2. it computes $b' \leftarrow \mathcal{A}_2(st_{\mathcal{A}}, I, \mathbf{c}, \mathbf{t})$,
3. it outputs 1 if $b' = b$ and 0 otherwise.

Clearly, $\mathcal{B}$ and $\mathcal{D}$ are polynomial-size since $\mathcal{A}_1$ and $\mathcal{A}_2$ are. So it remains to analyze $\mathcal{D}$'s success probability. First, notice that if the pair $(\mathrm{v}, st_{\mathcal{B}})$ are the output of $\mathbf{Real}_{\mathsf{SSE},\mathcal{B}}(k)$ then $\mathrm{v} = (I_b, \mathbf{c}_b, \mathbf{t}_b)$ and $st_{\mathcal{B}} = (st_{\mathcal{A}}, b)$. Therefore, $\mathcal{D}$ will output 1 if and only if $\mathcal{A}_2(st_{\mathcal{A}}, I_b, \mathbf{c}_b, \mathbf{t}_b)$ succeeds in guessing b. Notice, however, that $\mathcal{A}_1$ and $\mathcal{A}_2$'s views while being simulated by $\mathcal{B}$ and $\mathcal{D}$, respectively, are identical to the views they would have during an $\mathbf{Ind}_{\mathsf{SSE},\mathcal{A}}(k)$ experiment. We therefore have that

$$\begin{aligned} \Pr\left[\mathcal{D}(\mathrm{v}, st_{\mathcal{B}}) = 1 : (\mathrm{v}, st_{\mathcal{B}}) \leftarrow \mathbf{Real}_{\mathsf{SSE},\mathcal{B}}(k)\right] &= \Pr\left[\mathbf{Ind}_{\mathsf{SSE},\mathcal{A}}(k) = 1\right] \\ &\geq \frac{1}{2} + \varepsilon(k), \end{aligned}$$

where $\varepsilon(k)$ is some non-negligible function in k and the inequality follows from our original assumption about $\mathcal{A}$.

Let $\mathcal{S}$ be an arbitrary polynomial-size simulator and consider what happens when the pair (v, st_B) is output by a $\mathbf{Sim}_{\mathsf{SSE},\mathcal{B},\mathcal{S}}(k)$ experiment. First, note that any v output by $\mathcal{S}$ will be independent of b since $\tau(H_b) = \tau(H_0) = \tau(H_1)$ (by the restriction imposed in $\mathbf{Ind}_{\mathsf{SSE},\mathcal{A}}(k)$). Also, note that the string $st_{\mathcal{A}}$ output by $\mathcal{A}_1$ (while being simulated by $\mathcal{B}$) is independent of b. It follows then that $\mathcal{A}_2$ will guess b with probability at most $1/2$ and that,

$$\Pr\left[\mathcal{D}(\mathrm{v}, st_{\mathcal{B}}) = 1 : (\mathrm{v}, st_{\mathcal{B}}) \leftarrow \mathbf{Sim}_{\mathsf{SSE},\mathcal{B},\mathcal{S}}(k)\right] \leq \frac{1}{2}.$$

Combining the two previous Equations we get that,

$$\left|\Pr\left[\mathcal{D}(\mathrm{v}, st_{\mathcal{B}}) = 1 : (\mathrm{v}, st_{\mathcal{B}}) \leftarrow \mathbf{Real}_{\mathsf{SSE},\mathcal{B}}(k)\right] - \Pr\left[\mathcal{D}(\mathrm{v}, st_{\mathcal{B}}) = 1 : (\mathrm{v}, st_{\mathcal{B}}) \leftarrow \mathbf{Sim}_{\mathsf{SSE},\mathcal{B},\mathcal{S}}(k)\right]\right|$$

is non-negligible in k, from which the claim follows.

□

Claim. If SSE is non-adaptively indistinguishable, then it is non-adaptively semantically secure.

We show that if there exists a polynomial-size adversary $\mathcal{A}$ such that for all polynomial-size simulators $\mathcal{S}$, there exists a polynomial-size distinguisher $\mathcal{D}$ that can distinguish between the outputs of $\mathbf{Real}_{\mathsf{SSE},\mathcal{A}}(k)$ and $\mathbf{Sim}_{\mathsf{SSE},\mathcal{A},\mathcal{S}}(k)$, then there exists a polynomial-size adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ that succeeds in an $\mathbf{Ind}_{\mathsf{SSE},\mathcal{B}}(k)$ experiment with non-negligible probability over $1/2$.

Let H and $st_{\mathcal{A}}$ be the output of $\mathcal{A}(1^k)$ and recall that H is non-singular so there exists at least one history $H' \neq H$ such that $\tau(H') = \tau(H)$ and, furthermore, such a H' can be found efficiently. Now consider the simulator $\mathcal{S}^*$ that works as follows:

1. it generates a key $K^* \leftarrow \mathsf{Gen}(1^k)$,
2. given $\tau(H)$ it finds some H' such that $\tau(H') = \tau(H)$,
3. it builds an index I^*, a sequence of ciphertexts $\mathbf{c}^*$ and a sequence of trapdoors $\mathbf{t}^*$ from H' under key K^*,
4. it outputs $\mathrm{v} = (I^*, \mathbf{c}^*, \mathbf{t}^*)$ and $st^* = st_{\mathcal{A}}$.

Let $\mathcal{D}^*$ be the polynomial-size distinguisher (which depends on $\mathcal{S}^*$) guaranteed to exist by our initial assumption. Without loss of generality we assume $\mathcal{D}^*$ outputs 0 when given the output of a $\mathbf{Real}_{\mathsf{SSE},\mathcal{A}}(k)$ experiment. If this is not the case, then we consider the distinguisher that runs $\mathcal{D}^*$ and outputs its complement.

$\mathcal{B}_1$ is the adversary that computes $(H, st_{\mathcal{A}}) \leftarrow \mathcal{A}(1^k)$, uses $\tau(H)$ to find H' (as the simulator does) and returns $(H, H', st_{\mathcal{A}})$ as its output. $\mathcal{B}_2$ is the adversary that, given $st_{\mathcal{A}}$ and $(I_b, \mathbf{c}_b, \mathbf{t}_b)$, sets $\mathrm{v} = (I_b, \mathbf{c}_b, \mathbf{t}_b)$ and outputs the bit b obtained by running $\mathcal{D}^*(\mathrm{v}, st_{\mathcal{A}})$.

It remains to analyze $\mathcal{B}$'s success probability. Since b is chosen uniformly at random,

$$\Pr[\,\mathbf{Ind}_{\mathsf{SSE},\mathcal{B}}(k) = 1\,] = \frac{1}{2} \cdot \Big(\Pr[\,\mathbf{Ind}_{\mathsf{SSE},\mathcal{B}}(k) = 1 \mid b = 0\,] + \Pr[\,\mathbf{Ind}_{\mathsf{SSE},\mathcal{B}}(k) = 1 \mid b = 1\,] \Big). \quad (1)$$

If $b = 0$ occurs then $\mathcal{B}$ succeeds if and only if $\mathcal{D}^*(\mathrm{v}, st_{\mathcal{A}})$ outputs 0. Notice, however, that v and $st_{\mathcal{A}}$ are generated as in a $\mathbf{Real}_{\mathsf{SSE},\mathcal{A}}(k)$ experiment so it follows that,

$$\Pr[\,\mathbf{Ind}_{\mathsf{SSE},\mathcal{B}}(k) = 1 \mid b = 0\,] = \Pr[\,\mathcal{D}^*(\mathrm{v}, st_{\mathcal{A}}) = 0 : (\mathrm{v}, st_{\mathcal{A}}) \leftarrow \mathbf{Real}_{\mathsf{SSE},\mathcal{A}}(k)\,]. \quad (2)$$

On the other hand, if $b = 1$ then $\mathcal{B}$ succeeds if and only if $\mathcal{D}^*(\mathrm{v}, st_{\mathcal{A}})$ outputs 1. In this case, $st_{\mathcal{A}}$ and v are constructed as in a $\mathbf{Sim}_{\mathsf{SSE},\mathcal{A},\mathcal{S}^*}(k)$ experiment so we have,

$$\Pr[\,\mathbf{Ind}_{\mathsf{SSE},\mathcal{B}}(k) = 1 \mid b = 1\,] = \Pr[\,\mathcal{D}^*(\mathrm{v}, st_{\mathcal{A}}) = 1 : (\mathrm{v}, st_{\mathcal{A}}) \leftarrow \mathbf{Sim}_{\mathsf{SSE},\mathcal{A},\mathcal{S}^*}(k)\,]. \quad (3)$$

Combining Equations (2) and (3) with Equation (1) we get

$$\begin{aligned}
\Pr[\,\mathbf{Ind}_{\mathsf{SSE},\mathcal{B}}(k) = 1\,] &= \frac{1}{2} \cdot \Big(1 - \Pr[\,\mathcal{D}^*(\mathrm{v}, st_{\mathcal{A}}) = 1 : (\mathrm{v}, st_{\mathcal{A}}) \leftarrow \mathbf{Real}_{\mathsf{SSE},\mathcal{A}}(k)\,] \\
&\qquad + \Pr[\,\mathcal{D}^*(\mathrm{v}, st_{\mathcal{A}}) = 1 : (\mathrm{v}, st_{\mathcal{A}}) \leftarrow \mathbf{Sim}_{\mathsf{SSE},\mathcal{A},\mathcal{S}^*}(k) = 1\,] \Big) \\
&= \frac{1}{2} + \frac{1}{2} \cdot \Big(\Pr[\,\mathcal{D}^*(\mathrm{v}, st_{\mathcal{A}}) = 1 : (\mathrm{v}, st_{\mathcal{A}}) \leftarrow \mathbf{Sim}_{\mathsf{SSE},\mathcal{A},\mathcal{S}^*}(k) = 1\,] \\
&\qquad - \Pr[\,\mathcal{D}^*(\mathrm{v}, st_{\mathcal{A}}) = 1 : (\mathrm{v}, st_{\mathcal{A}}) \leftarrow \mathbf{Real}_{\mathsf{SSE},\mathcal{A}}(k) = 1\,] \Big) \\
&\geq \frac{1}{2} + \varepsilon(k),
\end{aligned}$$

where $\varepsilon(k)$ is a non-negligible function in k, and where the inequality follows from our original assumption about $\mathcal{A}$.

■

4.2.2 Adaptive security for SSE

We now turn to adaptive security definitions. Our indistinguishability-based definition is similar to the non-adaptive counterpart, with the exception that we allow the adversary to choose its history adaptively. More precisely, the challenger begins by flipping a coin b; then the adversary first submits two document collections $(\mathbf{D}_0, \mathbf{D}_1)$, subject to some constraints which we describe below, and receives the index of one of the collections $\mathbf{D}_b$; it then submits two keywords (w_0, w_1) and receives the trapdoor of one of the words w_b. This process goes on until the adversary has submitted polynomially-many queries and is then challenged to output the bit b.

Definition 4.10 (Adaptive indistinguishability security for SSE)**.** *Let* $\mathsf{SSE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Trpdr}, \mathsf{Search}, \mathsf{Dec})$ *be an index-based* SSE *scheme,* $k \in \mathbb{N}$ *be a security parameter,* $\mathcal{A} = (\mathcal{A}_0, \ldots, \mathcal{A}_{q+1})$ *be such that* $q \in \mathbb{N}$ *and consider the following probabilistic experiment* $\mathbf{Ind}^{\star}_{\mathcal{A},\mathsf{SSE}}(k)$*:*

$\mathbf{Ind}^{\star}_{\mathsf{SSE},\mathcal{A}}(k)$
$K \leftarrow \mathsf{Gen}(1^k)$
$b \xleftarrow{\$} \{0,1\}$
$(st_{\mathcal{A}}, \mathbf{D}_0, \mathbf{D}_1) \leftarrow \mathcal{A}_0(1^k)$
$(I_b, \boldsymbol{c}_b) \leftarrow \mathsf{Enc}_K(\mathbf{D}_b)$
$(st_{\mathcal{A}}, w_{0,1}, w_{1,1}) \leftarrow \mathcal{A}_1(st_{\mathcal{A}}, I_b)$
$t_{b,1} \leftarrow \mathsf{Trpdr}_K(w_{b,1})$
for $2 \leq i \leq q$,
  $(st_{\mathcal{A}}, w_{0,i}, w_{1,i}) \leftarrow \mathcal{A}_i(st_{\mathcal{A}}, I_b, \boldsymbol{c}_b, t_{b,1}, \ldots, t_{b,i-1})$
  $t_{b,i} \leftarrow \mathsf{Trpdr}_K(w_{b,i})$
let $\boldsymbol{t}_b = (t_{b,1}, \ldots, t_{b,q})$
$b' \leftarrow \mathcal{A}_{q+1}(st_{\mathcal{A}}, I_b, \boldsymbol{c}_b, \boldsymbol{t}_b)$
if $b' = b$, *output* 1
otherwise output 0

with the restriction that $\tau(\mathbf{D}_0, w_{0,1}, \ldots, w_{0,q}) = \tau(\mathbf{D}_1, w_{1,1}, \ldots, w_{1,q})$ *and where* $st_{\mathcal{A}}$ *is a string that captures* $\mathcal{A}$*'s state. We say that* SSE *is secure in the sense of* adaptive indistinguishability *if for all polynomial-size adversaries* $\mathcal{A} = (\mathcal{A}_0, \ldots, \mathcal{A}_{q+1})$ *such that* $q = \mathsf{poly}(k)$*,*

$$\Pr\left[\,\mathbf{Ind}^{\star}_{\mathsf{SSE},\mathcal{A}}(k) = 1\,\right] \leq \frac{1}{2} + \mathsf{negl}(k),$$

where the probability is over the choice of b*, and the coins of* Gen *and* Enc*.*

We now present our simulation-based definition, which is similar to the non-adaptive definition, except that the history is generated adaptively. More precisely, we require that the view of an adversary (i.e., the index, the ciphertexts and the trapdoors) generated from an adversarially and *adaptively* chosen history be simulatable given only the trace.

Definition 4.11 (Adaptive semantic security)**.** *Let* $\mathsf{SSE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Trpdr}, \mathsf{Search}, \mathsf{Dec})$ *be an index-based SSE scheme,* $k \in \mathbb{N}$ *be the security parameter,* $\mathcal{A} = (\mathcal{A}_0, \ldots, \mathcal{A}_q)$ *be an adversary such that* $q \in \mathbb{N}$ *and* $\mathcal{S} = (\mathcal{S}_0, \ldots, \mathcal{S}_q)$ *be a simulator and consider the following probabilistic experiments* $\mathbf{Real}^{\star}_{\mathsf{SSE},\mathcal{A}}(k)$ *and* $\mathbf{Sim}^{\star}_{\mathsf{SSE},\mathcal{A},\mathcal{S}}(k)$*:*

$\mathbf{Real}^{\star}_{\mathsf{SSE},\mathcal{A}}(k)$
$K \leftarrow \mathsf{Gen}(1^k)$
$(\mathbf{D}, st_{\mathcal{A}}) \leftarrow \mathcal{A}_0(1^k)$
$(I, \boldsymbol{c}) \leftarrow \mathsf{Enc}_K(\mathbf{D})$
$(w_1, st_{\mathcal{A}}) \leftarrow \mathcal{A}_1(st_{\mathcal{A}}, I, \boldsymbol{c})$
$t_1 \leftarrow \mathsf{Trpdr}_K(w_1)$
for $2 \leq i \leq q$,
$(w_i, st_{\mathcal{A}}) \leftarrow \mathcal{A}_i(st_{\mathcal{A}}, I, \boldsymbol{c}, t_1, \ldots, t_{i-1})$
$t_i \leftarrow \mathsf{Trpdr}_K(w_i)$
let $\boldsymbol{t} = (t_1, \ldots, t_q)$
output $\mathrm{v} = (I, \boldsymbol{c}, \boldsymbol{t})$ *and* $st_{\mathcal{A}}$

$\mathbf{Sim}^{\star}_{\mathsf{SSE},\mathcal{A},\mathcal{S}}(k)$
$(\mathbf{D}, st_{\mathcal{A}}) \leftarrow \mathcal{A}_0(1^k)$
$(I, \boldsymbol{c}, st_{\mathcal{S}}) \leftarrow \mathcal{S}_0(\tau(\mathbf{D}))$
$(w_1, st_{\mathcal{A}}) \leftarrow \mathcal{A}_1(st_{\mathcal{A}}, I, \boldsymbol{c})$
$(t_1, st_{\mathcal{S}}) \leftarrow \mathcal{S}_1(st_{\mathcal{S}}, \tau(\mathbf{D}, w_1))$
for $2 \leq i \leq q$,
$(w_i, st_{\mathcal{A}}) \leftarrow \mathcal{A}_i(st_{\mathcal{A}}, I, \boldsymbol{c}, t_1, \ldots, t_{i-1})$
$(t_i, st_{\mathcal{S}}) \leftarrow \mathcal{S}_i(st_{\mathcal{S}}, \tau(\mathbf{D}, w_1, \ldots, w_i))$
let $\boldsymbol{t} = (t_1, \ldots, t_q)$
output $\mathrm{v} = (I, \boldsymbol{c}, \boldsymbol{t})$ *and* $st_{\mathcal{A}}$

We say that SSE *is adaptively semantically secure if for all polynomial-size adversaries* $\mathcal{A} = (\mathcal{A}_0, \ldots, \mathcal{A}_q)$ *such that* $q = \mathsf{poly}(k)$*, there exists a non-uniform polynomial-size simulator* $\mathcal{S} = (\mathcal{S}_0, \ldots, \mathcal{S}_q)$*, such that for all polynomial-size* $\mathcal{D}$*,*

$$|\Pr[\mathcal{D}(\mathrm{v}, st_{\mathcal{A}}) = 1 : (\mathrm{v}, st_{\mathcal{A}}) \leftarrow \mathbf{Real}^{\star}_{\mathsf{SSE},\mathcal{A}}(k)] - \Pr[\mathcal{D}(\mathrm{v}, st_{\mathcal{A}}) = 1 : (\mathrm{v}, st_{\mathcal{A}}) \leftarrow \mathbf{Sim}^{\star}_{\mathsf{SSE},\mathcal{A},\mathcal{S}}(k)]| \leq \mathsf{negl}(k),$$

where the probabilities are over the coins of Gen *and* Enc*.*

In the following theorem we show that adaptive semantic security implies adaptive indistinguishability for SSE.

Theorem 4.12. *Adaptive semantic security of SSE implies adaptive indistinguishability of SSE.*

Proof. We show that if there exists a PPT adversary $\mathcal{A} = (\mathcal{A}_0, \ldots, \mathcal{A}_{q+1})$, where $q = \mathsf{poly}(k)$, that succeeds in an $\mathbf{Ind}^{\star}_{\mathsf{SSE},\mathcal{A}}$ experiment with non-negligible probability over $1/2$, then there exists a polynomial-size adversary $\mathcal{B} = (\mathcal{B}_0, \ldots, \mathcal{B}_q)$ and a polynomial-size distinguisher $\mathcal{D}$ such that for all polynomial-size simulators $\mathcal{S} = (\mathcal{S}_0, \ldots, \mathcal{S}_q)$, $\mathcal{D}$ distinguishes between the output of $\mathbf{Real}^{\star}_{\mathsf{SSE},\mathcal{B}}(k)$ and $\mathbf{Sim}^{\star}_{\mathsf{SSE},\mathcal{B},\mathcal{S}}(k)$.

The adversary $\mathcal{B} = (\mathcal{B}_0, \ldots, \mathcal{B}_q)$ works as follows:

- $\mathcal{B}_0$ computes $(st_{\mathcal{A}}, \mathbf{D}_0, \mathbf{D}_1) \leftarrow \mathcal{A}_0(1^k)$, samples $b \xleftarrow{\$} \{0,1\}$ and outputs $\mathbf{D}_b$ and $st_{\mathcal{B}} = (st_{\mathcal{A}}, b)$,
- $\mathcal{B}_1$ is given $(st_{\mathcal{B}}, I, \mathbf{c})$ and parses $st_{\mathcal{B}}$ into $(st_{\mathcal{A}}, b)$, computes $(w_{0,1}, w_{1,1}, st_{\mathcal{A}}) \leftarrow \mathcal{A}_1(st_{\mathcal{A}}, I, \mathbf{c})$ and outputs $w_{b,1}$ and $st_{\mathcal{B}} = (st_{\mathcal{A}}, b)$,
- for $2 \leq i \leq q$, $\mathcal{B}_i$ is given $(st_{\mathcal{B}}, I, \mathbf{c}, t_1, \ldots, t_{i-1})$. It parses $st_{\mathcal{B}}$ into $(st_{\mathcal{A}}, b)$, computes $(w_{0,i}, w_{1,i}, st_{\mathcal{A}}) \leftarrow \mathcal{A}_i(st_{\mathcal{A}}, I, \mathbf{c}, t_1, \ldots, t_{i-1})$, and outputs $w_{b,i}$ and $st_{\mathcal{B}} = (st_{\mathcal{A}}, b)$.

Let $\mathcal{D}$ be the distinguisher that, given $(\mathrm{v}, st_{\mathcal{B}})$ (which is either output by $\mathbf{Real}^{\star}_{\mathsf{SSE},\mathcal{B}}(k)$ or $\mathbf{Sim}^{\star}_{\mathsf{SSE},\mathcal{S},\mathcal{B}}(k)$) works as follows:

- it parses $st_{\mathcal{B}}$ into $(st_{\mathcal{A}}, b)$ and v into $(I, \mathbf{c}, \mathbf{t})$, where $\mathbf{t} = (t_1, \ldots, t_q)$,
- it computes $b' \leftarrow \mathcal{A}_{q+1}(st_{\mathcal{A}}, I, \mathbf{c}, \mathbf{t})$,
- it outputs 1 if $b' = b$ and 0 otherwise.

Clearly, $\mathcal{B}$ and $\mathcal{D}$ are polynomial-size since $\mathcal{A}$ is. So it remains to analyze $\mathcal{D}$'s success probability. First, notice that if the pair $(\mathrm{v}, st_{\mathcal{B}})$ is output by $\mathbf{Real}^{\star}_{\mathsf{SSE},\mathcal{B}}(k)$ then $\mathrm{v} = (I_b, \mathbf{c}_b, \mathbf{t}_b)$, where $\mathbf{t}_b = (t_{b,1}, \ldots, t_{b,q})$, and $st_{\mathcal{B}} = (st_{\mathcal{A}}, b)$. Therefore, $\mathcal{D}$ will output 1 if and only if $\mathcal{A}_{q+1}(st_{\mathcal{A}}, I_b, \mathbf{c}_b, \mathbf{t}_b)$ succeeds in guessing

b. Notice, however, that $\mathcal{A}_0$ through $\mathcal{A}_{q+1}$'s views while being simulated by $\mathcal{B}$ and $\mathcal{D}$, respectively, are identical to the views they would have during an $\mathbf{Ind}^{\star}_{\mathsf{SSE},\mathcal{A}}(k)$ experiment. We therefore have

$$\begin{aligned}\Pr\left[\,\mathcal{D}(\mathrm{v}, st_{\mathcal{B}}) = 1 : (\mathrm{v}, st_{\mathcal{B}}) \leftarrow \mathbf{Real}^{\star}_{\mathsf{SSE},\mathcal{B}}(k)\,\right] &= \Pr\left[\,\mathbf{Ind}^{\star}_{\mathsf{SSE},\mathcal{A}}(k) = 1\,\right] \\ &\geq \frac{1}{2} + \varepsilon(k),\end{aligned}$$

where $\varepsilon(k)$ is some non-negligible function in k and the inequality follows from our original assumption about $\mathcal{A}$.

Let $\mathcal{S}$ be an arbitrary polynomial-size simulator and consider what happens when the pair $(\mathrm{v}, st_{\mathcal{B}})$ is the output of a $\mathbf{Sim}^{\star}_{\mathsf{SSE},\mathcal{B},\mathcal{S}}(k)$ experiment. First, note that any v output by $\mathcal{S}$ will be independent of b since $\tau(H_b) = \tau(H_0) = \tau(H_1)$ (by the restriction imposed in $\mathbf{Ind}^{\star}_{\mathsf{SSE},\mathcal{A}}(k)$). Also, note that the string $st_{\mathcal{A}}$ is independent of b. It follows then that $\mathcal{A}_{q+1}(st_{\mathcal{A}}, \mathrm{v})$ will guess b with probability at most $1/2$ and that

$$\Pr\left[\,\mathcal{D}(\mathrm{v}, st_{\mathcal{B}}) = 1 : (\mathrm{v}, st_{\mathcal{B}}) \leftarrow \mathbf{Sim}^{\star}_{\mathsf{SSE},\mathcal{B},\mathcal{S}}(k)\,\right] \leq \frac{1}{2}.$$

Combining the two previous Equations we get that

$$\left|\Pr\left[\,\mathcal{D}(\mathrm{v}, st_{\mathcal{B}}) = 1 : (\mathrm{v}, st_{\mathcal{B}}) \leftarrow \mathbf{Real}^{\star}_{\mathsf{SSE},\mathcal{B}}(k)\,\right] - \Pr\left[\,\mathcal{D}(\mathrm{v}, st_{\mathcal{B}}) = 1 : (\mathrm{v}, st_{\mathcal{B}}) \leftarrow \mathbf{Sim}^{\star}_{\mathsf{SSE},\mathcal{B},\mathcal{S}}(k)\,\right]\right|$$

is non-negligible in k, from which the claim follows.

■

5 Efficient and Secure Searchable Symmetric Encryption

We now present our SSE constructions, and state their security in terms of the definitions presented in Section 4. We start by introducing some additional notation and the data structures used by the constructions. Let $\delta(\mathbf{D}) \subseteq \Delta$ be the set of distinct keywords in the document collection $\mathbf{D}$, and $\delta(D) \subseteq \Delta$ be the set of distinct keywords in the document $D \in \mathbf{D}$. We assume that keywords in Δ can be represented using at most ℓ bits. Also, recall that n is the number of documents in the collection and that $\mathbf{D}(w)$ is the set of identifiers of documents in $\mathbf{D}$ that contain keyword w ordered in lexicographic order.

We use several data structures, including arrays, linked lists and look-up tables. Given an array $\mathtt{A}$, we refer to the element at address i in $\mathtt{A}$ as $\mathtt{A}[i]$, and to the address of element x relative to $\mathtt{A}$ as $\mathsf{addr}_{\mathtt{A}}(x)$. So if $\mathtt{A}[i] = x$, then $\mathsf{addr}_{\mathtt{A}}(x) = i$. In addition, a linked list $\mathtt{L}$ of n nodes that is stored in an array $\mathtt{A}$ is a sequence of nodes $\mathtt{N}_i = \langle v_i, \mathsf{addr}_{\mathtt{A}}(\mathtt{N}_{i+1})\rangle$, where $1 \leq i \leq n$, and where v_i is an arbitrary string and $\mathsf{addr}_{\mathtt{A}}(\mathtt{N}_{i+1})$ is the memory address of the next node in the list. We denote by $\#\mathtt{L}$ the number of nodes in the list $\mathtt{L}$.

5.1 An efficient non-adaptively secure construction (SSE-1)

We first give an overview of our one-round non-adaptively secure SSE construction. First, each document in the collection $\mathcal{D}$ is encrypted using a symmetric encryption scheme. We then construct a single index I which consists of two data structures:

$\mathtt{A}$: an array in which, for all $w \in \delta(\mathbf{D})$, we store an encryption of the set $\mathbf{D}(w)$.

$\mathtt{T}$: a look-up table in which, for all $w \in \delta(\mathbf{D})$, we store information that enables one to locate and decrypt the appropriate element from $\mathtt{A}$.

For each distinct keyword $w_i \in \delta(\mathbf{D})$, we start by creating a linked list $\mathtt{L}_i$ where each node contains the identifier of a document in $\mathbf{D}(w_i)$. We then store all the nodes of all the lists in the array $\mathtt{A}$ permuted in a random order and encrypted with randomly generated keys. Before encrypting the j^{th} node of list $\mathtt{L}_i$, it is augmented with a pointer (with respect to $\mathtt{A}$) to the $(j+1)$-th node of $\mathtt{L}_i$, together with the key used to encrypt it. In this way, given the location in $\mathtt{A}$ and the decryption key for the first node of a list $\mathtt{L}_i$, the server will be able to locate and decrypt all the nodes in $\mathtt{L}_i$. Note that by storing the nodes of all lists $\mathtt{L}_i$ in a random order, the length of each individual $\mathtt{L}_i$ is hidden.

We then build a look-up table $\mathtt{T}$ that allows one to locate and decrypt the first node of each list $\mathtt{L}_i$. Each entry in $\mathtt{T}$ corresponds to a keyword $w_i \in \delta(\mathbf{D})$ and consists of a pair <`address`,`value`>. The field `value` contains the location in $\mathtt{A}$ and the decryption key for the first node of $\mathtt{L}_i$. `value` is itself encrypted using the output of a pseudo-random function. The other field, `address`, is simply used to locate an entry in $\mathtt{T}$. The look-up table $\mathtt{T}$ is managed using *indirect addressing* (described below).

The client generates both $\mathtt{A}$ and $\mathtt{T}$ based on the plaintext document collection $\mathbf{D}$, and stores them on the server together with the *encrypted* documents. When the user wants to retrieve the documents that contain keyword w_i, it computes the decryption key and the address for the corresponding entry in $\mathtt{T}$ and sends them to the server. The server locates and decrypts the given entry of $\mathtt{T}$, and gets a pointer to and the decryption key for the first node of $\mathtt{L}_i$. Since each node of $\mathtt{L}_i$ contains a pointer to the next node, the server can locate and decrypt all the nodes of $\mathtt{L}_i$, revealing the identifiers in $\mathbf{D}(w_i)$.

Efficient storage and access of sparse tables. We describe the indirect addressing method that we use to efficiently manage look-up tables. The entries of a look-up table $\mathtt{T}$ are tuples <`address`,`value`> in which the `address` field is used as a *virtual address* to locate the entry in $\mathtt{T}$ that contains some `value` field. Given a parameter ℓ, a virtual address is from a domain of exponential size, i.e., from $\{0,1\}^\ell$. However, the maximum number of entries in a look-up table will be polynomial in ℓ, so the number of virtual addresses that are used is $\mathsf{poly}(\ell)$. If, for a table $\mathtt{T}$, the `address` field is from $\{0,1\}^\ell$, the `value` field is from $\{0,1\}^v$ and there are at most s entries in $\mathtt{T}$, then we say $\mathtt{T}$ is a $(\{0,1\}^\ell \times \{0,1\}^v \times s)$ look-up table.

Let Addr be the set of virtual addresses that are used for entries in a look-up table $\mathtt{T}$. We can efficiently store $\mathtt{T}$ such that, when given a virtual address, it returns the associated `value` field. We achieve this by organizing Addr in a so-called *FKS dictionary* [22], an efficient data structure for storage of sparse tables that requires $O(|\mathsf{Addr}|)$ storage and $O(1)$ look-up time. In other words, given some virtual address a, we are able to tell if $a \in \mathsf{Addr}$ and if so, return the associated `value` in constant look-up time. Addresses that are not in Addr are considered undefined.

Our construction in detail. We are now ready to proceed to the details of the construction. Let $\mathsf{SKE1}$ and $\mathsf{SKE2}$ be PCPA-secure symmetric encryption schemes, respectively. In addition, we make use of a pseudo-random function f and two pseudo-random permutations π and ψ with the following parameters:

$$f : \{0,1\}^k \times \{0,1\}^\ell \to \{0,1\}^{k+\log_2(s)};$$

$$\pi : \{0,1\}^k \times \{0,1\}^\ell \to \{0,1\}^\ell;$$

$$\psi : \{0,1\}^k \times \{0,1\}^{\log_2(s)} \to \{0,1\}^{\log_2(s)},$$

where s is the total size of the encrypted document collection in "min-units", where a min-unit is the smallest possible size for a keyword (e.g., one byte)[5]. Let $\mathtt{A}$ be an array with s non-empty cells, and let $\mathtt{T}$ be a $(\{0,1\}^\ell \times \{0,1\}^{k+\log_2(s)} \times |\Delta|)$ look-up table, managed using indirect addressing as described previously. Our construction is described in Fig. 1.

[5]If the documents are not encrypted with a length preserving encryption scheme or if they are compressed before encryption, then s is the maximum of {total size of the plaintext $\mathbf{D}$, total size of the encrypted $\mathbf{D}$}.

$\mathsf{Gen}(1^k)$: sample $K_1, K_2, K_3 \xleftarrow{\$} \{0,1\}^k$, generate $K_4 \leftarrow \mathsf{SKE2.Gen}(1^k)$ and output $K = (K_1, K_2, K_3, K_4)$.

$\mathsf{Enc}_K(\mathbf{D})$:

Initialization:

1. scan $\mathbf{D}$ and generate the set of distinct keywords $\delta(\mathbf{D})$
2. for all $w \in \delta(\mathbf{D})$, generate $\mathbf{D}(w)$
3. initialize a global counter $\mathsf{ctr} = 1$

Building the array $\mathtt{A}$:

4. for $1 \leq i \leq |\delta(\mathbf{D})|$, build a list $\mathtt{L}_i$ with nodes $\mathtt{N}_{i,j}$ and store it in array $\mathtt{A}$ as follows:
   (a) sample a key $K_{i,0} \xleftarrow{\$} \{0,1\}^k$
   (b) for $1 \leq j \leq |\mathbf{D}(w_i)| - 1$:
   - let $\mathsf{id}(D_{i,j})$ be the j^{th} identifier in $\mathbf{D}(w_i)$
   - generate a key $K_{i,j} \leftarrow \mathsf{SKE1.Gen}(1^k)$
   - create a node $\mathtt{N}_{i,j} = \langle \mathsf{id}(D_{i,j}) \| K_{i,j} \| \psi_{K_1}(\mathsf{ctr}+1) \rangle$
   - encrypt node $\mathtt{N}_{i,j}$ under key $K_{i,j-1}$ and store it in $\mathtt{A}$:

$$\mathtt{A}[\psi_{K_1}(\mathsf{ctr})] \leftarrow \mathsf{SKE1.Enc}_{K_{i,j-1}}(\mathtt{N}_{i,j})$$

   - set $\mathsf{ctr} = \mathsf{ctr} + 1$

   (c) for the last node of $\mathtt{L}_i$,
   - set the address of the next node to NULL: $\mathtt{N}_{i,|\mathbf{D}(w_i)|} = \langle \mathsf{id}(D_{i,|\mathbf{D}(w_i)|}) \| 0^k \| \mathsf{NULL} \rangle$
   - encrypt the node $\mathtt{N}_{i,|\mathbf{D}(w_i)|}$ under key $K_{i,|\mathbf{D}(w_i)|-1}$ and store it in $\mathtt{A}$:

$$\mathtt{A}[\psi_{K_1}(\mathsf{ctr})] \leftarrow \mathsf{SKE1.Enc}_{K_{i,|\mathbf{D}(w_i)|-1}}\left(\mathtt{N}_{i,|\mathbf{D}(w_i)|}\right)$$

   - set $\mathsf{ctr} = \mathsf{ctr} + 1$

5. let $s' = \sum_{w_i \in \delta(\mathbf{D})} |\mathbf{D}(w_i)|$. If $s' < s$, then set the remaining $s - s'$ entries of $\mathtt{A}$ to random values of the same size as the existing s' entries of $\mathtt{A}$

Building the look-up table $\mathtt{T}$:

6. for all $w_i \in \delta(\mathbf{D})$, set $\mathtt{T}[\pi_{K_3}(w_i)] = \langle \mathsf{addr}_{\mathtt{A}}(\mathtt{N}_{i,1}) || K_{i,0} \rangle \oplus f_{K_2}(w_i)$
7. if $|\delta(\mathbf{D})| < |\Delta|$, then set the remaining $|\Delta| - |\delta(\mathbf{D})|$ entries of $\mathtt{T}$ to random values of the same size as the existing $|\delta(\mathbf{D})|$ entries of $\mathtt{T}$

Preparing the output:

8. for $1 \leq i \leq n$, let $c_i \leftarrow \mathsf{SKE2.Enc}_{K_4}(D_i)$
9. output $(I, \mathbf{c})$, where $I = (\mathtt{A}, \mathtt{T})$ and $\mathbf{c} = (c_1, \ldots, c_n)$

$\mathsf{Trpdr}_K(w)$: output $t = (\pi_{K_3}(w), f_{K_2}(w))$

$\mathsf{Search}(I, t)$:

1. parse t as (γ, η), and set $\theta \leftarrow \mathtt{T}[\gamma]$
2. if $\theta \neq \perp$, then parse $\theta \oplus \eta$ as $\langle \alpha || K' \rangle$ and continue, otherwise return $\perp$
3. use the key K' to decrypt the list $\mathtt{L}$ starting with the node stored at address α in $\mathtt{A}$
4. output the list of document identifiers contained in $\mathtt{L}$

$\mathsf{Dec}_K(c_i)$: output $D_i \leftarrow \mathsf{SKE2.Dec}_{K_4}(c_i)$

Figure 1: A non-adaptively secure SSE scheme (SSE-1)

Padding. Consistent with our security definitions, SSE-1 reveals only the access pattern, the search pattern, the total size of the encrypted document collection, and the number of documents it contains. To achieve this, a certain amount of padding to the array and the table are necessary. To see why, recall that the array $\mathtt{A}$ stores a collection of linked lists $(\mathtt{L}_1, \ldots, \mathtt{L}_{|\delta(\mathbf{D})|})$, where each $\mathtt{L}_i$ contains the identifiers of all the documents that contain the keyword $w_i \in \delta(\mathbf{D})$. Note that the number of non-empty cells in $\mathtt{A}$, denoted by $\#\mathtt{A}$, is equal to the total number of nodes contained in all the lists. In other words,

$$\#\mathtt{A} = \sum_{w_i \in \delta(\mathbf{D})} \#\mathtt{L}_i.$$

Notice, however, that this is also equal to the sum (over all the documents) of the number of distinct keywords found in each document. In other words,

$$\#\mathtt{A} = \sum_{w_i \in \delta(\mathbf{D})} \#\mathtt{L}_i = \sum_{i=1}^{n} |\delta(D_i)|.$$

Let $\#\mathbf{D}$ be the number of (non-distinct) words in the document collection $\mathbf{D}$. Clearly, if

$$\sum_{i=1}^{n} |\delta(D_i)| < \#\mathbf{D},$$

then there exists at least one document in $\mathbf{D}$ that contains a certain word more than once. Our goal, therefore, will be to pad $\mathtt{A}$ so that this leakage does not occur.

In practice, the adversary (i.e., the server) will not know $\#\mathbf{D}$ explicitly, but it can approximate it as follows using the encrypted documents it stores. Recall that s is the total size of the encrypted document collection in "min-units", where a min-unit is the smallest possible size for a keyword (e.g., one byte). Also, let s' be the total size of the encrypted document collection in "max-units", where a max-unit is the largest possible size for a keyword (e.g., ten bytes). It follows then that

$$s' \leq \#\mathbf{D} \leq s.$$

From the previous argument, it follows that $\mathtt{A}$ must be padded so that $\#\mathtt{A}$ is at least s'. Note, however, that setting $\#\mathtt{A} = s'$ is not sufficient since an adversary will know that in all likelihood $\#\mathbf{D} > s'$. We therefore pad $\mathtt{A}$ so that $\#\mathtt{A} = s$. The padding is done using random values, which are indistinguishable from the (useful) entries in $\mathtt{A}$.

We follow the same line of reasoning for the look-up table $\mathtt{T}$, which has at least one entry for each distinct keyword in $\mathbf{D}$. To avoid revealing the number of distinct keywords in $\mathbf{D}$, we add an additional $|\Delta| - |\delta(\mathbf{D})|$ entries in $\mathtt{T}$ filled with random values so that the total number of entries is always equal to $|\Delta|$.

Theorem 5.1. *If f is a pseudo-random function, if π and ψ are pseudo-random permutations, and if* SKE1 *and* SKE2 *are PCPA-secure, then* SSE-1 *is non-adaptively secure.*

Proof. We describe a polynomial-size simulator $\mathcal{S}$ such that for all polynomial-size adversaries $\mathcal{A}$, the outputs of $\mathbf{Real}_{\mathsf{SSE},\mathcal{A}}(k)$ and $\mathbf{Sim}_{\mathsf{SSE},\mathcal{A},\mathcal{S}}(k)$ are indistinguishable. Consider the simulator $\mathcal{S}$ that, given the trace of a history H, generates a string $\mathrm{v}^* = (I^*, \mathbf{c}^*, \mathbf{t}^*) = \big((\mathtt{A}^*, \mathtt{T}^*), c_1^*, \ldots, c_n^*, t_1^*, \ldots, t_q^*\big)$ as follows:

1. (Simulating $\mathtt{A}^*$) if $q = 0$ then for $1 \leq i \leq s$, $\mathcal{S}$ sets $\mathtt{A}^*[i]$ to a string of length $\log_2(n) + k + \log_2(s)$ selected uniformly at random. If $q \geq 1$, it sets $|\delta(\mathbf{D})| = q$ and runs Step 4 of the Enc algorithm on the sets $\mathbf{D}(w_1)$ through $\mathbf{D}(w_q)$ using different random strings of size $\log_2(s)$ instead of $\psi(\mathsf{ctr})$. Note that $\mathcal{S}$ knows $\mathbf{D}(w_1)$ through $\mathbf{D}(w_q)$ from the trace it receives.

2. (Simulating $\mathtt{T}^*$) if $q = 0$ then for $1 \leq i \leq |\Delta|$, $\mathcal{S}$ generates pairs (a_i^*, c_i^*) such that the a_i^* are distinct strings of length ℓ chosen uniformly at random, and the c_i^* are strings of length $\log_2(s)+k$ also chosen uniformly at random. If $q \geq 1$, then for $1 \leq i \leq q$, $\mathcal{S}$ generates random values β_i^* of length $\log_2(s) + k$ and a_i^* of length ℓ, and sets

$$\mathtt{T}^*[a_i^*] = \langle \mathsf{addr}_{\mathtt{A}^*}(\mathtt{N}_{i,1}) || K_{i,0} \rangle \oplus \beta_i^*.$$

It then inserts dummy entries into the remaining entries of $\mathtt{T}^*$. So, in other words, $\mathcal{S}$ runs Step 6 of the Enc algorithm with $|\delta(\mathbf{D})| = q$, using $\mathtt{A}^*$ instead of $\mathtt{A}$, and using β_i^* and a_i^* instead of $f_y(w_i)$ and $\pi_z(w_i)$, respectively.

3. (Simulating t_i^*) it sets $t_i^* = (a_i^*, \beta_i^*)$

4. (Simulating c_i^*) it sets c_i^* to a $|D_i|$-bit string chosen uniformly at random (recall that $|D_i|$ is included in the trace).

It follows by construction that searching on I^* using trapdoors t_i^* will yield the expected search outcomes.

Let v be the outcome of a $\mathbf{Real}_{\mathsf{SSE},\mathcal{A}}(k)$ experiment. We now claim that no polynomial-size distinguisher $\mathcal{D}$ that receives $st_{\mathcal{A}}$ can distinguish between the distributions v^* and v, otherwise, by a standard hybrid argument, $\mathcal{D}$ could distinguish between at least one of the elements of v and its corresponding element in v^*. We argue that this is not possible by showing that each element of v^* is computationally indistinguishable from its corresponding element in v to a distinguisher $\mathcal{D}$ that is given $st_{\mathcal{A}}$.

1. ($\mathtt{A}$ and $\mathtt{A}^*$) Recall that $\mathtt{A}$ consists of s' $\mathsf{SKE1}$ encryptions and $s - s'$ random strings of the same size. If $q = 0$, $\mathtt{A}^*$ consists of all random strings. While if $q \geq 1$, $\mathtt{A}^*$ consists of q $\mathsf{SKE1}$ encryptions and $s - q$ random strings of the same size. In either case, with all but negligible probability, $st_{\mathcal{A}}$ does not include the keys $K_{i,j}$ used to encrypt the list nodes stored in $\mathtt{A}$. The PCPA-security of $\mathsf{SKE1}$ then guarantees that each element in $\mathtt{A}^*$ is indistinguishable from its counterpart in $\mathtt{A}$.

2. ($\mathtt{T}$ and $\mathtt{T}^*$) Recall that $\mathtt{T}$ consists of $|\delta(\mathbf{D})|$ ciphertexts, c_i, generated by XOR-ing a message with the output of f, and of $|\Delta| - |\delta(\mathbf{D})|$ random values of size $k + \log_2(s)$. If $q = 0$, $\mathtt{T}^*$ consists of all random values. While if $q \geq 1$, $\mathtt{T}^*$ consists of q ciphertexts generated by XOR-ing a message with a random string β_i^* of length $k + \log_2(s)$, and $|\Delta| - q$ random strings of the same length. In either case, with all but negligible probability, $st_{\mathcal{A}}$ does not include the PRF key K_2, and therefore the pseudo-randomness of f guarantees that each element of $\mathtt{T}$ is indistinguishable from its counterpart in $\mathtt{T}^*$.

3. (t_i and t_i^*) Recall that t_i consists of evaluations of the PRP π and the PRF f. With all but negligible probability $st_{\mathcal{A}}$ will not contain the keys K_2 and K_3, so the pseudo-randomness of π and f then will guarantee that each t_i is indistinguishable from t_i^*.

4. (c_i and c_i^*) Recall that c_i is $\mathsf{SKE2}$ encryption. Since, with all but negligible probability, $st_{\mathcal{A}}$ will not contain the encryption key K_4, the PCPA-security of $\mathsf{SKE2}$ will guarantee that c_i and c_i^* are indistinguishable.

■

Regarding efficiency, we remark that each query takes only one round, and $O(1)$ message size. In terms of storage, the demands are $O(1)$ on the user and $O(s)$ on the server; more specifically, in addition to the encrypted $\mathbf{D}$, the server stores the index I, which has size $O(s)$, and the look-up

table T, which has size $O(|\Delta|)$. Since the size of the encrypted documents is $O(s)$, accommodating the auxiliary data structures used for searching does not change (asymptotically) the storage requirements for the server. The user spends $O(1)$ time to compute a trapdoor, while for a query for keyword w, the server spends time proportional to $|\mathbf{D}(w)|$.

5.2 An adaptively secure construction

While our SSE-1 construction is efficient, it is only proven secure against non-adaptive adversaries. We now show a second construction, SSE-2, which achieves semantic security against adaptive adversaries at the price of requiring higher communication size per query and more storage on the server. Asymptotically, however, the costs are the same.

The difficulty of proving our SSE-1 construction secure against an adaptive adversary stems from the difficulty of simulating in advance an index for the adversary that will be consistent with future unknown queries. Given the intricate structure of the SSE-1 construction, with each keyword having a corresponding linked list whose nodes are stored encrypted and in a random order, building an index that allows for such a simulation seems challenging. We circumvent this problem as follows.

For a keyword w and an integer j, we derive a *label* for w by concatenating w with j, where j is first converted to a string of characters. So, for example, if w is the keyword "coin" and $j = 1$, then $w||j$ is the string "coin1". We define the *family* of a keyword $w \in \delta(\mathbf{D})$ to be the set of labels $\text{FAM}_w = \{w||j : 1 \leq j \leq |\mathbf{D}(w)|\}$. So if the keyword "coin" appears in three documents, then $\text{FAM}_w = \{$"coin1", "coin2", "coin3"$\}$. Note that the maximum size of a keyword's family is n, i.e., the number of documents in the collection. We associate with the document collection $\mathbf{D}$ an index I, which is a look-up table managed using the indirect addressing technique described in Section 5.1 (thus, I has entries of the form `<address, value>`). For each label in a keyword's family, we add an entry in I whose `value` field is the identifier of the document that contains an instance of w. So for each $w \in \delta(\mathbf{D})$, instead of keeping a list, we simply derive the family FAM_w and for each label in FAM_w we add into the table an entry with the identifier of a document in $\delta(\mathbf{D})$. So if "coin" is contained in documents (D_5, D_8, D_9), then we add the entries `<address1,5>`, `<address2,8>`, `<address3,9>` (in which the `address` field is a function of the labels "coin1", "coin2", "coin3", respectively). In order to hide the number of distinct keywords in each document, we pad the look-up table so that the identifier of each document appears in the same number of entries. To search for the documents that contain w, it now suffices to search for all the labels in w's family. Since each label is unique, a search for it "reveals" a single document identifier. Translated to the proof, this will allow the simulator to construct an index for the adversary that is indistinguishable from a real index, even before it knows any of the adversary's queries.

Let k be security parameter and $s = \mathtt{max} \cdot n$, where n is the number of documents in $\mathbf{D}$ and `max` is the maximum number of distinct keywords that can fit in the largest document in $\mathbf{D}$ (an algorithm to determine `max` is given below). Recall that keywords in Δ can be represented using at most ℓ bits. We use a pseudo-random permutation $\pi : \{0,1\}^k \times \{0,1\}^{\ell+\log_2(n+\mathtt{max})} \rightarrow \{0,1\}^{\ell+\log_2(n+\mathtt{max})}$ and a PCPA-secure symmetric encryption scheme SKE. Let I be a $(\{0,1\}^{\ell+\log_2(n+\mathtt{max})} \times \{0,1\}^{\log_2(n)} \times s)$ look-up table, managed using indirect addressing. The SSE-2 construction is described in Fig. 2.

Determining `max`. Recall that $\delta(\mathbf{D})$ is the set of distinct keywords that exist in $\mathbf{D}$. Assuming the minimum size for a keyword is one byte, we give an algorithm to determine `max`, given the size (in bytes) of the largest document in $\mathbf{D}$, which we denote by `MAX`. In step 1 we try to fit the maximum number of distinct 1-byte keywords; there are 2^8 such keywords, which gives a total size of 256 bytes ($2^8 \cdot 1$ bytes). If $\mathtt{MAX} > 256$, then we continue to step 2. In step 2 we try to fit the maximum number of distinct 2-byte keywords; there are 2^{16} such keywords, which gives a total size of 131328 bytes ($2^8 \cdot 1 + 2^{16} \cdot 2$ bytes). Generalizing, in step i we try to fit the maximum number of distinct i-byte

$\mathsf{Gen}(1^k)$: sample $K_1 \xleftarrow{\$} \{0,1\}^k$ and generate $K_2 \leftarrow \mathsf{SKE.Gen}(1^k)$. Output $K = (K_1, K_2)$.

$\mathsf{Enc}_K(\mathbf{D})$:

Initialization:

1. scan $\mathbf{D}$ and generate the set of distinct keywords $\delta(\mathbf{D})$
2. for all $w \in \delta(\mathbf{D})$, generate $\mathbf{D}(w)$ (i.e., the set of documents that contain w)

Building the look-up table I:

3. for $1 \leq i \leq |\delta(\mathbf{D})|$ and $1 \leq j \leq |\mathbf{D}(w_i)|$,
   (a) let $\mathsf{id}(D_{i,j})$ be the j^{th} identifier in $\mathbf{D}(w_i)$
   (b) set $I[\pi_K(w_i||j)] = \mathsf{id}(D_{i,j})$
4. let $s' = \sum_{w_i \in \delta(\mathbf{D})} |\mathbf{D}(w_i)|$
5. if $s' < s$, then set values for the remaining $(s - s')$ entries in I such that for all documents $D \in \mathbf{D}$, the identifier $\mathsf{id}(D)$ appears exactly $\mathtt{max}$ times. This can be done as follows:
   * for all $D_i \in \mathbf{D}$:
     · let c be the number of entries in I that already contain $\mathsf{id}(D_i)$
     · for $1 \leq l \leq \mathtt{max} - c$, set $I[\pi_K(0^\ell||n+l)] = \mathsf{id}(D_i)$

Preparing the output:

6. for $1 \leq i \leq n$, let $c_i \leftarrow \mathsf{SKE.Enc}_{K_2}(D_i)$
6. output $(I, \mathbf{c})$, where $\mathbf{c} = (c_1, \ldots, c_n)$

$\mathsf{Trpdr}_K(w)$: output $t = (t_1, \ldots, t_n) = (\pi_K(w||1), \ldots, \pi_K(w||n))$

$\mathsf{Search}(I, t)$: for all $1 \leq i \leq n$, if $I[t_w] \neq \perp$, then add $I[t_w]$ to X. Output X.

$\mathsf{Dec}_K(c_i)$: output $D_i \leftarrow \mathsf{SKE.Dec}_{K_2}(c_i)$

Figure 2: An adaptively secure SSE scheme (SSE-2)

keywords, which is $2^{8 \cdot i}$. We continue similarly until step i when $\mathtt{MAX}$ becomes smaller than the total size accumulated so far. Then we go back to step $i-1$ and try to fit as many $(i-1)$-byte distinct keywords as possible in a document of size $\mathtt{MAX}$. For example, when the largest document in $\mathbf{D}$ has size $\mathtt{MAX} = 1$ MByte, we can fit at most $\mathtt{max} = 355349$ distinct keywords (2^8 distinct 1-byte keywords + 2^{16} distinct 2-byte keywords + 289557 distinct 3-byte keywords). Note that $\mathtt{max}$ cannot be larger than $|\Delta|$; thus, if we get a value for $\mathtt{max}$ (using the previously described algorithm) that is larger than $|\Delta|$, then we set $\mathtt{max} = |\Delta|$.

Theorem 5.2. *If π is a pseudo-random permutation and* SKE *is PCPA-secure, then the* SSE-2 *construction is adaptively secure.*

Proof. We describe a polynomial-size simulator $\mathcal{S} = (\mathcal{S}_0, \ldots \mathcal{S}_q)$ such that for all polynomial-size adversaries $\mathcal{A} = (\mathcal{A}_0, \ldots, \mathcal{A}_q)$, the outputs of $\mathbf{Real}^\star_{\mathsf{SSE},\mathcal{A}}(k)$ and $\mathbf{Sim}^\star_{\mathsf{SSE},\mathcal{A},\mathcal{S}}(k)$ are computationally indistinguishable. Consider the simulator $\mathcal{S} = (\mathcal{S}_0, \ldots, \mathcal{S}_q)$ that adaptively generates a string $\mathrm{v}^* = (I^*, \mathbf{c}^*, \mathbf{t}^*) = (I^*, c_1^*, \ldots, c_n^*, t_1^*, \ldots, t_n^*)$ as follows:

$\mathcal{S}_0(1^k, \tau(\mathbf{D}))$: it computes $\mathtt{max}$ using the algorithm described above. Note that it can do this since it knows the size of all the documents from the trace of $\mathbf{D}$. It then sets I^* to be a $\left(\{0,1\}^{\ell + \log_2(n+\mathtt{max})} \times \{0,1\}^{\log_2(n)} \times s\right)$ look-up table, where $s = \mathtt{max} \cdot n$, with $\mathtt{max}$ copies of each document's identifier inserted at random locations. $\mathcal{S}_0$ then includes I^* in $st_{\mathcal{S}}$ and outputs $(I^*, \mathbf{c}^*, st_{\mathcal{S}})$, where $c_i^* \xleftarrow{\$} \{0,1\}^{|D_i|}$.

Since, with all but negligible probability, $st_{\mathcal{A}}$ does not include K_1, I^* is indistinguishable from a real index otherwise one could distinguish between the output of π and a random string of size $\ell + \log_2(n + \mathtt{max})$. Similarly, since, with all but negligible probability, $st_{\mathcal{A}}$ does not include K_2, the PCPA-security of SKE guarantees that each c_i^* is indistinguishable from a real ciphertext.

$\mathcal{S}_1(st_{\mathcal{S}}, \tau(\mathbf{D}, w_1))$: Recall that $\mathbf{D}(w_i) = (\mathbf{D}(w_i||1), \ldots, \mathbf{D}(w_i||n))$. Note that each $\mathbf{D}(w_i||j)$, for $1 \leq j \leq n$, contains only one document identifier which we refer to as $\mathsf{id}(D_{i,j})$. For all $1 \leq j \leq n$, $\mathcal{S}_1$ randomly picks an address $\mathtt{addr}_j$ from I^* such that $I^*[\mathtt{addr}_j] = \mathsf{id}(D_{i,j})$, making sure that all $\mathtt{addr}_j$ are pairwise distinct. It then sets $t_1^* = (\mathtt{addr}_1, \ldots, \mathtt{addr}_n)$. Also, $\mathcal{S}_1$ remembers the association between t_1^* and w_i by including it in $st_{\mathcal{S}}$. It then outputs $(t_1^*, st_{\mathcal{S}})$.

Since, with all but negligible probability, $st_{\mathcal{A}}$ does not include K_1, t_1^* is indistinguishable from a real trapdoor t_1, otherwise one could distinguish between the output of π and a random string of size $\ell + \log_2(n + \mathtt{max})$.

$\mathcal{S}_i(st_{\mathcal{S}}, \tau(\mathbf{D}, w_1, \ldots, w_i))$ for $2 \leq i \leq q$: first $\mathcal{S}_i$ checks whether (the unknown) w_i has appeared before. This can be done by checking whether there exists a $1 \leq j \leq i-1$ such that $\sigma[i, j] = 1$. If w_i has not previously appeared, then $\mathcal{S}_i$ generates a trapdoor the same way $\mathcal{S}_1$ does (making sure not to reuse any previously used $\mathtt{addr}$'s). On the other hand, if w_i did previously appear, then $\mathcal{S}_i$ retrieves the trapdoor previously used for w_i and uses it as t_i^*. $\mathcal{S}_i$ outputs (t_i^*, st_S) and, clearly, t_i^* is indistinguishable from t_i (again since $st_{\mathcal{A}}$ does not include K_1 with all but negligible probability).

■

Just like our non-adaptively secure scheme, this construction requires one round of communication for each query and an amount of computation on the server proportional with the number of documents that contain the query (i.e., $O(|\mathbf{D}(w)|)$. Similarly, the storage and computational demands on the user are $O(1)$. The communication is equal to $O(n)$ and the storage on the server is increased by a factor of $\mathtt{max}$ when compared to the $\mathsf{SSE\text{-}1}$ construction. We note that the communication cost can be reduced if in each entry of I corresponding to an element in some keyword w's family, we also store an encryption of $|\mathbf{D}(w)|$. In this way, after searching for a label in w's family, the user will know $|\mathbf{D}(w)|$ and can derive $\textsc{fam}_w$. The user can then send in a single round all the trapdoors corresponding to the remaining labels in w's family.

5.3 Secure updates

We consider a limited notion of document updates, in which new documents can be *added* to the existing document collection. We allow for secure updates to the document collection in the sense defined by Chang and Mitzenmacher [18]: each time the user adds a new set ζ of encrypted documents, ζ is considered a separate document collection. Old trapdoors cannot be used to search newly submitted documents, as the new documents are part of a collection indexed using different secrets. If we consider the submission of the original document collection an update, then after u updates, there will be u document collections stored on the server. In the previously proposed solution [18], the user sends a pseudo-random seed for each document collection, which implies that the trapdoors have length $O(u)$. We propose a solution that achieves better bounds for the length of trapdoors (namely $O(\log u)$) and for the amount of computation at the server. For applications where the number of queries dominates the number of updates, our solution may significantly reduce the communication size and the server's computation. A thorough evaluation of the cost of updates for real-world workloads is outside the scope of this work.

When the user performs an update, i.e., submits a set ζ^a of new documents, the server checks if there exists (from previous updates) a document collection ζ^b, such that $|\zeta^b| \leq |\zeta^a|$. If so, the server

sends back ζ^b and the user combines ζ^a and ζ^b into a single collection ζ^c with $|\zeta^a| + |\zeta^b|$ documents. The user then computes an index for ζ^c. The server stores the combined document collection ζ^c and its index I_c, and deletes the document collections ζ^a, ζ^b and their indexes I_a, I_b. Note that ζ^c and its index I_c will not reveal anything more than what was already revealed by the ζ^a, ζ^b and their indexes I_a, I_b, since one can trivially reduce the security of the combined collection to the security of the composing collections.

Next, we analyze the number of document collections that results after u updates using the method proposed above. Without loss of generality, we assume that each update consists of one new document. Then, it can be shown that after u updates, the number of document collections is given by $f(u)$, by which we denote the Hamming weight of u (*i.e.*, the number of 1's in the binary representation of u"). Note that $f(u) \in [1, \lfloor \log(u+1) \rfloor]$. This means that after u updates, there will be at most $\log(u)$ document collections, thus the queries sent by the user have size $O(\log u)$ and the search can be done in $O(\log u)$ by the server (as opposed to $O(u)$ in [18]).

6 Multi-User Searchable Encryption

In this section we consider a natural extension of SSE to the setting where a user owns a document collection, but an arbitrary group of users can submit queries to search the collection. A familiar question arises in this new setting, that of managing access privileges while preserving privacy with respect to the server. We first present a definition of a multi-user searchable encryption scheme (MSSE) and some of its desirable security properties, followed by an efficient construction which, in essence, combines a single-user SSE scheme with a broadcast encryption scheme.

Definition 6.1 (Multi-user searchable symmetric encryption)**.** *An index-based multi-user SSE scheme is a collection of seven polynomial-time algorithms* $\mathsf{MSSE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Add}, \mathsf{Revoke}, \mathsf{Trpdr}, \mathsf{Search}, \mathsf{Dec})$ *such that,*

$K_O \leftarrow \mathsf{Gen}(1^k)$*: is a probabilistic key generation algorithm that is run by the owner to set up the scheme. It takes as input a security parameter* k*, and outputs an owner secret key* K_O*.*

$(I, \boldsymbol{c}, st_O, st_S) \leftarrow \mathsf{Enc}(K_O, G, \mathbf{D})$*: is a probabilistic algorithm run by the owner to encrypt the document collection. It takes as input the owner's secret key* K_O *a set of authorized users* $G \subseteq \mathcal{U}$ *and a document collection* $\mathbf{D}$*. It outputs a secure index* I*, a sequence of ciphertexts* $\boldsymbol{c}$*, an owner state* st_O *and a server state* st_S*. We sometimes write this as* $(I, \boldsymbol{c}, st_O, st_S) \leftarrow \mathsf{Enc}_{K_O}(G, \mathbf{D})$*.*

$K_U \leftarrow \mathsf{Add}(K_O, st_O, U)$*: is a probabilistic algorithm run by the owner to add a user. It takes as input the owner's secret key* K_O *and state* st_O *and a unique user id* U *and outputs* U*'s secret key* K_U*. We sometimes write this as* $K_U \leftarrow \mathsf{Add}_{K_O}(st_O, U)$*.*

$(st_O, st_S) \leftarrow \mathsf{Revoke}(K_O, st_O, U)$*: is a probabilistic algorithm run by the owner to remove a user from* G*. It takes as input the owner's secret key* K_O *and state* st_O *and a unique user id* U*. It outputs an updated owner state* st_O *and an updated server state* st_S*. We sometimes write this as* $(st_O, st_S) \leftarrow \mathsf{Revoke}_{K_O}(st_O, U)$*.*

$t \leftarrow \mathsf{Trpdr}(K_U, w)$*: is a deterministic algorithm run by a user (including* O*) to generate a trapdoor for a keyword. It takes as input a user* U*'s secret key* K_U *and a keyword* w*, and outputs a trapdoor* t *or the failure symbol* $\perp$*. We sometimes write this as* $t \leftarrow \mathsf{Trpdr}_{K_U}(w)$*.*

$X \leftarrow \mathsf{Search}(st_S, I, t)$*: is a deterministic algorithm run by the server* S *to perform a search. It takes as input a server state* st_S*, an index* I *and a trapdoor* t*, and outputs a set* $X \in 2^{[1,n]} \cup \{\perp\}$*, where* $\perp$ *denotes the failure symbol.*

$D_i \leftarrow \mathsf{Dec}(K_U, c_i)$*: is a deterministic algorithm run by the users to recover a document. It takes as input a user key* K_U *and a ciphertext* c_i*, and outputs a document* D_i*. We sometimes write this as* $D_i \leftarrow \mathsf{Dec}_{K_U}(c_i)$*.*

The security of a multi-user scheme can be defined similarly to the security of a single-user scheme, as the server should not learn anything about the documents and queries beyond what can be inferred from the access and search patterns. One distinct property in this new setting is that of *revocation*, which essentially requires that a revoked user no longer be able to perform searches on the owner's documents.

Definition 6.2 (Revocation)**.** *Let* $\mathsf{MSSE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Add}, \mathsf{Revoke}, \mathsf{Trpdr}, \mathsf{Search})$ *be a multi-user SSE scheme,* $k \in \mathbb{N}$ *be the security parameter, and* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ *be an adversary. We define* $\mathbf{Rev}_{\mathsf{MSSE},\mathcal{A}}(k)$ *as the following probabilistic experiment:*

$$
\begin{array}{l}
\mathbf{Rev}_{\mathsf{MSSE},\mathcal{A}}(k) \\
\quad K_O \leftarrow \mathsf{Gen}(1^k) \\
\quad (st_{\mathcal{A}}, \mathbf{D}) \leftarrow \mathcal{A}_1(1^k) \\
\quad K_{\mathcal{A}} \leftarrow \mathsf{Add}(K_O, \mathcal{A}) \\
\quad (I, \boldsymbol{c}, st_O, st_S) \leftarrow \mathsf{Enc}_{K_O}(\mathbf{D}) \\
\quad st_{\mathcal{A}} \leftarrow \mathcal{A}_2^{\mathcal{O}(I,\boldsymbol{c},st_S,\cdot)}(st_{\mathcal{A}}, K_{\mathcal{A}}) \\
\quad (st_O, st_S) \leftarrow \mathsf{Revoke}_{K_O}(\mathcal{A}) \\
\quad t \leftarrow \mathcal{A}_3(st_{\mathcal{A}}) \\
\quad X \leftarrow \mathsf{Search}(st_{\mathcal{S}}, I, t) \\
\quad \textit{if } X \neq \perp \textit{ output } 1 \\
\quad \textit{else output } 0
\end{array}
$$

where $\mathcal{O}(I, \boldsymbol{c}, st_S, \cdot)$ *is an oracle that takes as input a token* t *and returns the ciphertexts in* $\boldsymbol{c}$ *indexed by* $X \leftarrow \mathsf{Search}(I, t, st_S)$ *if* $X \neq \perp$ *and* $\perp$ *otherwise. We say that* MSSE *achieves revocation if for all polynomial-size adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$*,*

$$\Pr\left[\,\mathbf{Rev}_{\mathsf{MSSE},\mathcal{A}}(k) = 1\,\right] \leq \mathsf{negl}(k),$$

where the probability is over the coins of Gen*,* Add*,* Revoke *and* Index*.*

6.1 Our construction

We assume the honest-but-curious adversarial model for the server; we also assume that the server does not collude with revoked users (if such collusion occurs, then our construction cannot prevent a revoked user from searching). In general, it is challenging to provide security against such collusion without re-computing the secure index after each user revocation.

Our construction makes use of a single-user SSE scheme $\mathsf{SSE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Trpdr}, \mathsf{Search})$ and a broadcast encryption scheme $\mathsf{BE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Add}, \mathsf{Dec})$. We require standard security notions for broadcast encryption: namely, that in addition to being PCPA-secure it provide revocation-scheme security against a coalition of all revoked users. Let $\mathcal{U}$ denote the set of all users and $G \subseteq \mathcal{U}$ the set of users (currently) authorized to search. Let ϕ be a pseudo-random permutation such that $\phi : \{0,1\}^k \times \{0,1\}^t \rightarrow \{0,1\}^t$, where t is the size of a trapdoor in the underlying single-user SSE scheme. ϕ can be constructed using techniques for building pseudo-random permutations over domains of arbitrary size [12, 9, 32].

Our multi-user construction $\mathsf{MSSE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Add}, \mathsf{Revoke}, \mathsf{Trpdr}, \mathsf{Search})$ is described in detail in Fig. 3. The owner key is composed of a key K for the underlying single-user scheme, a key r for the pseudo-random permutation ϕ and a master key mk for the broadcast encryption scheme. To

$\mathsf{Gen}(1^k)$: generate $K \leftarrow \mathsf{SSE.Gen}(1^k)$, $mk \leftarrow \mathsf{BE.Gen}(1^k)$ and output $K_O = (K, mk)$.

$\mathsf{Enc}(K_O, G, \mathbf{D})$: compute $(I, \mathbf{c}) \leftarrow \mathsf{SSE.Enc}_K(\mathbf{D})$ and $st_S \leftarrow \mathsf{BE.Enc}(mk, G, r)$, where G includes the server and $r \xleftarrow{\$} \{0,1\}^k$. Set $st_O = r$ and output $(I, \mathbf{c}, st_S, st_O)$.

$\mathsf{Add}(K_O, st_O, U)$: compute $uk_U \leftarrow \mathsf{BE.Add}(mk, U)$ and output $K_U = (K, uk_U, r)$.

$\mathsf{Revoke}(K_O, st_O, U)$: sample $r \xleftarrow{\$} \{0,1\}^k$ and output $st_S = \mathsf{BE.Enc}(mk, \mathsf{G}\backslash U, \mathsf{r})$ and $st_O = r$.

$\mathsf{Trpdr}(K_U, w)$: retrieve st_S from the server. If $\mathsf{BE.Dec}(uk_U, st_S) = \perp$ output $\perp$, else compute $r \leftarrow \mathsf{BE.Dec}(uk_U, st_S)$ and $t' \leftarrow \mathsf{SSE.Trpdr}_K(w)$. Output $t \leftarrow \phi_r(t')$.

$\mathsf{Search}(st_S, I, t)$: compute $r \leftarrow \mathsf{BE.Dec}(uk_S, st_S)$, $t' \leftarrow \phi_r^{-1}(t)$ and output $X \leftarrow \mathsf{SSE.Search}(I, t')$.

Figure 3: A multi-user SSE scheme

encrypt a data collection, the owner first encrypts the collection using the single-user SSE scheme. This results in a secure index I and a sequence of ciphertexts $\mathbf{c}$. It then generates a server state st_S that consists of a broadcast encryption of r. Finally, it stores the secure index I, the ciphertexts $\mathbf{c}$ and the server state st_S on the server. To add a user U, the owner generates a user key uk_U for the broadcast encryption scheme and sends U the triple (K, r, uk_U) (thus, the owner acts as the center in a broadcast encryption scheme).

To search for a keyword w, an authorized user first retrieves the latest server state st_S from the server and uses its user key uk_U to recover r. It generates a single-user trapdoor t, encrypts it using ϕ keyed with r, and sends the result to the server. The server, upon receiving $\phi_r(t)$, recovers the trapdoor by computing $t = \phi_r^{-1}(\phi_r(t))$. The key r currently used for ϕ is only known by the owner and by the set of currently authorized users (which includes the server). Each time a user U is revoked, the owner picks a new r' and generates a new server state st'_S by encrypting r' with the broadcast encryption scheme for the set $G\backslash U$. The new state st'_S is then sent to the server who uses it to replace the old state. For all subsequent queries, the server uses the new r' when inverting ϕ. Since revoked users will not be able to recover r', with overwhelming probability, their queries will not yield a valid trapdoor after the server applies $\phi_{r'}^{-1}$.

Notice that to give a user U permission to search through $\mathbf{D}$, the owner sends it all the secret information needed to perform searches in a single-user context. This means that the owner should possess an additional secret that will not be shared with U and that allows him to perform authentication with the server when he wants to update $\mathbf{D}$ or revoke users from searching. The extra layer given by the pseudo-random permutation ϕ, together with the guarantees offered by the broadcast encryption scheme and the assumption that the server is honest-but-curious, is what prevents users from performing successful searches once they are revoked. We leave the formal treatment of the security of the multi-user scheme for future work.

We point out that users receive their keys for the broadcast encryption scheme only when they are given authorization to search. So while a user U that has not joined the system yet could retrieve the broadcast encryption of r (i.e., the state st_S) from the server, since it does not have an authorized key it will not be able to recover r. Similarly, when a revoked user U retrieves the broadcast encryption of r from the server, it cannot recover r because $U \notin G$. Moreover, even though a revoked user which has been re-authorized to search could recover (old) values of r that were used while he was revoked, these values are no longer of interest. The fact that backward secrecy is not needed for the BE scheme makes the Add algorithm more efficient, since it does not require the owner to send a message to the server.

Our multi-user construction is very efficient on the server side during a query: when given a trapdoor, the server only needs to evaluate a pseudo-random permutation in order to determine if

the user is revoked. If access control mechanisms were used instead for this step, a more expensive authentication protocol would be required for each search query in order to establish the identity of the querier.

7 Conclusions

In this article, we have revisited the problem of *searchable symmetric encryption*, which allows a client to store its data on a remote server in such a way that it can search over it in a private manner. We make several contributions including new security definitions and new constructions. Motivated by subtle problems in all previous security definitions for SSE, we propose new definitions and point out that the existing notions have significant practical drawbacks: contrary to the natural use of searchable encryption, they only guarantee security for users that perform all their searches at once. We address this limitation by introducing stronger definitions that guarantee security even when users perform more realistic searches. We also propose two new SSE constructions. Surprisingly, despite being provably secure under our stronger security definitions, these are the most efficient schemes to date and are (asymptotically) optimal (i.e., the work performed by the server per returned document is constant in the size of the data). Finally, we also consider multi-user SSE, which extends the searching ability to parties other than the owner.

Acknowledgements

We thank Fabian Monrose for helpful discussions during the early stages of this work. We also thank the anonymous referees for helpful comments and, in particular, for suggesting a way to remove the need for non-uniformity in the proof of Theorem 4.9. During part of this work, the third author was supported by a Bell Labs Graduate Research Fellowship. The fourth author is supported in part by an IBM Faculty Award, a Xerox Innovation Group Award, a gift from Teradata, an Intel equipment grant, a UC-MICRO grant, and NSF Cybertrust grant No. 0430254.

References

[1] Privacy with Security. Technical report, DARPA Information Science and Technology Study Group, December 2002. `http://www.cs.berkeley.edu/~tygar/papers/ISAT-final-briefing.pdf`.

[2] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. M. Lee, G. Neven, P. Paillier, and H. Shi. Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions. In V. Shoup, editor, *Advances in Cryptology – CRYPTO '05*, volume 3621 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2005.

[3] Leonard Adleman. Two theorems on random polynomial time. In *Symposium on Foundations of Computer Science (FOCS '78)*, pages 75–83. IEEE Computer Society, 1978.

[4] A. Adya, M. Castro, R. Chaiken, J. Douceur, J. Howell, and J. Lorch. Federated, available and reliable storage for an incompletely trusted environment (Farsite), 2002.

[5] G. Amanatidis, A. Boldyreva, and A O'Neill. Provably-secure schemes for basic query support in outsourced databases. In *Data and Applications Security XXI*, volume 4602 of *Lecture Notes in Computer Science*, pages 14–30. Springer, 2007.

[6] L. Ballard, S. Kamara, and F. Monrose. Achieving efficient conjunctive keyword searches over encrypted data. In S. Qing, W. Mao, J. Lopez, and G. Wang, editors, *Seventh International Conference on Information and Communication Security (ICICS '05)*, volume 3783 of *Lecture Notes in Computer Science*, pages 414–426. Springer, 2005.

[7] B. Barak and O. Goldreich. Universal arguments and their applications. In *IEEE Conference on Computational Complexity (CCC '02)*, pages 194–203. IEEE Computer Society, 2002.

[8] M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In A. Menezes, editor, *Advances in Cryptology – CRYPTO '07*, Lecture Notes in Computer Science, pages 535–552. Springer, 2007.

[9] M. Bellare, T. Ristenpart, P. Rogaway, and T. Stegers. Format-preserving encryption. In *Proc. of Selected Areas in Cryptography '09*, volume 5867 of *Lecture Notes in Computer Science*, pages 295–312. Springer-Verlag, 2009. full version available as ePrint report 2009/251.

[10] S.M. Bellovin and W.R. Cheswick. Privacy-enhanced searches using encrypted Bloom filters. Technical Report 2004/022, IACR ePrint Cryptography Archive, 2004.

[11] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for fault-tolerant distributed computing. In *ACM Symposium on the Theory of Computation (STOC '88)*, pages 1–10. ACM, 1988.

[12] J. Black and P. Rogaway. Ciphers with arbitrary finite domains. In B. Preneel, editor, *The Cryptographers' Track at the RSA Conference (CT-RSA '02)*, volume 2271 of *Lecture Notes in Computer Science*, pages 114–130. Springer-Verlag, 2002.

[13] B. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[14] M. Blum, W. S. Evans, P. Gemmell, S. Kannan, and M. Naor. Checking the correctness of memories. In *IEEE Symposium on Foundations of Computer Science (FOCS '91)*, pages 90–99. IEEE Computer Society, 1991.

[15] D. Boneh, G. di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Advances in Cryptology – EUROCRYPT '04*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522. Springer, 2004.

[16] D. Boneh, E. Kushilevitz, R. Ostrovsky, and W. Skeith. Public-key encryption that allows PIR queries. In A. Menezes, editor, *Advances in Cryptology – CRYPTO '07*, volume 4622 of *Lecture Notes in Computer Science*, pages 50–67. Springer, 2007.

[17] D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In *Theory of Cryptography*, volume 4392 of *Lecture Notes in Computer Science*, pages 535–554. Springer, 2007.

[18] Y. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *Applied Cryptography and Network Security (ACNS '05)*, volume 3531 of *Lecture Notes in Computer Science*, pages 442–455. Springer, 2005.

[19] M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *Advances in Cryptology - ASIACRYPT '10*, volume 6477 of *Lecture Notes in Computer Science*, pages 577–594. Springer, 2010.

[20] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In *ACM Conference on Computer and Communications Security (CCS '06)*, pages 79–88. ACM, 2006.

[21] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. Technical Report 2006/210 (version 20060626:205325), IACR ePrint Cryptography Archive, 2006.

[22] M.L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with 0(1) worst case access time. *Journal of the ACM*, 31(3):538–544, 1984.

[23] E-J. Goh. Secure indexes. Technical Report 2003/216, IACR ePrint Cryptography Archive, 2003. See `http://eprint.iacr.org/2003/216`.

[24] O. Goldreich, S. Micali, and A. Wigderson. How to play ANY mental game. In *ACM Symposium on the Theory of Computation (STOC '87)*, pages 218–229. ACM, 1987.

[25] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM*, 43(3):431–473, 1996.

[26] P. Golle, J. Staddon, and B. Waters. Secure conjunctive keyword search over encrypted data. In M. Jakobsson, M. Yung, and J. Zhou, editors, *Applied Cryptography and Network Security Conference (ACNS '04)*, volume 3089 of *Lecture Notes in Computer Science*, pages 31–45. Springer, 2004.

[27] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Batch codes and their applications. In *ACM Symposium on Theory of Computing (STOC '04)*, pages 262–271. ACM, 2004.

[28] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Cryptography from anonymity. In *IEEE Symposium on Foundations of Computer Science (FOCS '06)*. IEEE Computer Society, 2006.

[29] J. Katz and R. Ostrovsky. Round-optimal secure two-party computation. In M. Franklin, editor, *Advances in Cryptology – CRYPTO '04*, volume 3152 of *Lecture Notes in Computer Science*, pages 335–354. Springer, 2004.

[30] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, and B. Zhao. Oceanstore: an architecture for global-scale persistent storage. In *Architectural support for programming languages and operating systems*, pages 190–201. ACM, 2000.

[31] E. Kushilevitz and R. Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *IEEE Symposium on Foundations of Computer Science (FOCS '97)*, pages 364–373. IEEE Computer Society, 1997.

[32] B. Morris, P. Rogaway, and T. Stegers. How to encipher messages on a small domain. In *Proc. of CRYPTO '09*, pages 286–302. Springer-Verlag, 2009.

[33] A. Muthitacharoen, R. Morris, T.M. Gil, and B. Chen. Ivy: A read/write peer-to-peer file system. In *Symposium on Operating System Design and Implementation (OSDI '02)*. USENIX Association, 2002.

[34] A. Narayanan and V. Shmatikov. Obfuscated databases and group privacy. In *Proc. of ACM CCS '05*, pages 102–111, 2005.

[35] R. Ostrovsky. Efficient computation on oblivious RAMs. In *ACM Symposium on Theory of Computing (STOC '90)*, pages 514–523. ACM, 1990.

[36] R. Ostrovsky and W. Skeith. Private searching on streaming data. In V. Shoup, editor, *Advances in Cryptology – CRYPTO '05*, volume 3621 of *Lecture Notes in Computer Science*, pages 223–240. Springer, 2005.

[37] D. Park, K. Kim, and P. Lee. Public key encryption with conjunctive field keyword search. In C. H. Lim and M. Yung, editors, *Workshop on Information Security Applications (WISA '04)*, volume 3325 of *Lecture Notes in Computer Science*, pages 73–86. Springer, 2004.

[38] S. Sedghi, J. Doumen, P. H. Hartel, and W. Jonker. Towards an information theoretic analysis of searchable encryption. In *Proc. of ICICS*, pages 345–360, 2008.

[39] E. Shi, J. Bethencourt, H. T.-H. Chan, D. Song, and A. Perrig. Multi-dimensional range query over encrypted data. In *IEEE Symposium on Security and Privacy*, pages 350–364, 2007.

[40] D. Song, D. Wagner, and A. Perrig. Practical techniques for searching on encrypted data. In *IEEE Symposium on Research in Security and Privacy*, pages 44–55. IEEE Computer Society, 2000.

[41] A. Yao. Protocols for secure computations. In *IEEE Symposium on Foundations of Computer Science (FOCS '82)*, pages 160–164. IEEE Computer Society, 1982.

A Security Definitions of Basic Primitives

Definition A.1 (PCPA-security)**.** *Let* $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a symmetric encryption scheme and* $\mathcal{A}$ *be an adversary and consider the following probabilistic experiment* $\mathbf{PCPA}_{\mathsf{SKE},\mathcal{A}}(k)$*:*

1. *a key* $K \leftarrow \mathsf{Gen}(1^k)$ *is generated,*
2. $\mathcal{A}$ *is given oracle access to* $\mathsf{Enc}_K(\cdot)$*,*
3. $\mathcal{A}$ *outputs a message* m*,*
4. *two ciphertexts* c_0 *and* c_1 *are generated as follows:* $c_0 \leftarrow \mathsf{Enc}_K(m)$ *and* $c_1 \xleftarrow{\$} \boldsymbol{C}$*, where* $\boldsymbol{C}$ *denotes the ciphertext space of* SKE *(i.e., the set of all possible ciphertexts). A bit* b *is chosen at random and* c_b *is given to* $\mathcal{A}$*,*
5. $\mathcal{A}$ *is again given access to the encryption oracle, and after polynomially-many queries it outputs a bit* b'*.*
6. *if* $b' = b$*, the experiment returns* 1 *otherwise it returns* 0*.*

We say that SKE *is CPA-secure if for all polynomial-size adversaries* $\mathcal{A}$*,*

$$\Pr\left[\,\mathbf{PCPA}_{\mathsf{SKE},\mathcal{A}}(k) = 1\,\right] \leq \frac{1}{2} + \mathsf{negl}(k),$$

where the probability is over the choice of b *and the coins of* Gen *and* Enc*.*

Definition A.2 (Pseudo-random function)**.** *A function* $f : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^m$ *is pseudo-random if it is computable in polynomial time (in* k*) and if for all polynomial-size* $\mathcal{A}$*,*

$$\left|\Pr\left[\,\mathcal{A}^{f_K(\cdot)} = 1 : K \xleftarrow{\$} \{0,1\}^k\,\right] - \Pr\left[\,\mathcal{A}^{g(\cdot)} = 1 : g \xleftarrow{\$} \mathsf{Func}[n,m]\,\right]\right| \leq \mathsf{negl}(k)$$

where the probabilities are taken over the choice of K *and* g*. If* f *is bijective then it is a pseudo-random permutation.*

B Limitations of Previous SSE Definitions

As discussed in the Introduction, SSE schemes can be constructed by combining a secure index and a symmetric encryption scheme. A secure index scheme is a tuple of four polynomial-time algorithms $\mathsf{SI} = (\mathsf{Gen}, \mathsf{Index}, \mathsf{Trpdr}, \mathsf{Search})$ that work as follows. Gen is a probabilistic algorithm that takes as input a security parameter k and outputs a key K. Index is a probabilistic algorithm that takes as input a key K and a document collection $\mathbf{D}$ and outputs a secure index I. Trpdr is a deterministic algorithm that takes as input a key K and a keyword w and outputs a trapdoor t. Search is a deterministic algorithm that takes as input an index I and a trapdoor t and outputs a set X of document identifiers.

To date, two security definitions have been used for secure index schemes: indistinguishability against chosen-keyword attacks (IND2-CKA) from [23] and a simulation-based definition introduced in [18].

Game-based definitions. Intuitively, the security guarantee that IND2-CKA provides can be described as follows: given access to a set of indexes, the adversary (i.e., the server) cannot learn any partial information about the underlying documents beyond what he can learn from using a trapdoor that was given to him by the client, and this holds even against adversaries that can convince the client to generate indexes and trapdoors for documents and keywords chosen by the adversary (i.e., chosen keyword attacks).

In the following definition, we use $\ominus$ to denote the symmetric difference between two sets A and B: $A \ominus B = (A \cup B) \setminus (A \cap B)$.

Definition B.1 (IND2-CKA [23])**.** *Let* $\mathsf{SI} = (\mathsf{Gen}, \mathsf{Index}, \mathsf{Trpdr}, \mathsf{Search})$ *be a secure index scheme,* Δ *be a dictionary,* $\mathcal{A}$ *be an adversary and consider the following probabilistic experiment* $\mathbf{CKA}_{\mathsf{SI},\mathcal{A}}(k)$*:*

1. $\mathcal{A}$ *generates a collection of* n *documents* $\mathbf{D} = (D_1, \ldots, D_n)$ *from* Δ*.*
2. *the challenger generates a key* $K \leftarrow \mathsf{Gen}(1^k)$ *and indexes* $(I_1, \ldots, I_n)$ *such that* $I_i \leftarrow \mathsf{Index}_K(D_i)$
3. *given* $(I_1, \ldots, I_n)$ *and oracle access to* $\mathsf{Trpdr}_K(\cdot)$*,* $\mathcal{A}$ *outputs two documents* D_0^* *and* D_1^* *such that* $D_0^* \in \mathbf{D}$*,* $D_1^* \subseteq \Delta$*, and* $|D_0^* \setminus D_1^*| \neq 0$ *and* $|D_1^* \setminus D_0^*| \neq 0$*. In addition, we require that* $\mathcal{A}$ *does not query its trapdoor oracle on any word in* $D_0^* \ominus D_1^*$*.*
4. *the challenger chooses a bit* b *uniformly at random and computes* $I_b \leftarrow \mathsf{Index}_K(D_b)$*.*
5. *given* I_b *and oracle access to* $\mathsf{Trpdr}_K(\cdot)$*,* $\mathcal{A}$ *outputs a bit* b'*. Here, again,* $\mathcal{A}$ *cannot query its oracle on any word in* $D_0^* \ominus D_1^*$*.*
6. *the output of the experiment is* 1 *if* $b' = b$ *and* 0 *otherwise.*

We say that SI *is IND2-CKA secure if for all polynomial-size adversaries* $\mathcal{A}$*,*

$$\Pr\left[\,\mathbf{CKA}_{\mathsf{SI},\mathcal{A}}(k) = 1\,\right] \leq \frac{1}{2} + \mathsf{negl}(k),$$

where the probability is over the choice of b *and the coins of* Gen *and* Enc*.*

As Goh remarks (*cf.* Note 1, p. 5 of [23]), IND2-CKA does not explicitly require that trapdoors be secure since this is not a requirement for all applications of secure indexes. It follows then that the notion of IND2-CKA is not strong enough to guarantee that an index can be safely used to build a SSE scheme. To remedy the situation, one might be tempted to require that a secure index be IND2-CKA and that its trapdoors not leak any partial information about the keywords.

We point out, however, that this cannot be done in a straightforward manner. Indeed, we give an explicit construction of an IND2-CKA index with "secure" trapdoors that cannot yield a secure SSE scheme.

Before we describe the construction, we briefly discuss two of its characteristics. First, it is defined to operate on documents, as opposed to document *collections*. We chose to define it this way, as opposed to defining it according to Definition 4.1, so that we could use the original formulations of IND2-CKA (or IND-CKA). In particular, this means that build an index one must run the Index and algorithm on each document D_i in a collection $\mathbf{D} = (D_1, \ldots, D_n)$. Similarly, to search one must run the Search algorithm on each index I_i in the collection $(I_1, \ldots, I_n)$. Second, the construction is *stateful*, which means that the Index and Trpdr algorithms are extended to take as input and output a state st.

Recall that $\Delta = (w_1, \ldots, w_d)$ is a dictionary of d words; we assume, without loss of generality, that each word is encoded as a bit string of length ℓ. The construction uses a pseudo-random permutation $\pi : \{0,1\}^k \times \{0,1\}^{\ell+k} \to \{0,1\}^{\ell+k}$ and a function $H : \Delta \to \mathbb{Z}_d$ that maps a word in Δ to its position in the dictionary (e.g., the third word in Δ is mapped to 3). Let $\mathsf{SI} = (\mathsf{Gen}, \mathsf{Index}, \mathsf{Trpdr}, \mathsf{Search})$ be the secure index scheme defined as follows:

$\mathsf{Gen}(1^k)$: generate a random key $K \xleftarrow{\$} \{0,1\}^k$.

$\mathsf{Index}(K, st, D)$:

1. Instantiate an array $\mathtt{A}$ of d elements[6]
2. set $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$
3. for each word $w \in \delta(D)$:
   (a) compute $r \leftarrow \pi_K(w||\mathsf{ctr})$ and $z \leftarrow H(w)$
   (b) store $r \oplus (w||0^k)$ in $\mathtt{A}[z]$;
4. fill in the empty locations of $\mathtt{A}$ with random strings of length $\ell + k$;
5. output $\mathtt{A}$ as the index I and ctr as st.

$\mathsf{Trpdr}(K, st, w)$: output $t_w = (\pi_K(w||1), \ldots, \pi_K(w||\mathsf{ctr}))$.

$\mathsf{Search}(I_i, t_w)$:

1. parse t_w as $(r_1, \ldots, r_{\mathsf{ctr}})$
2. for $0 \leq j \leq |\mathtt{A}| - 1$:
   (a) decrypt the j^{th} element of $\mathtt{A}$ by computing $v \leftarrow \mathtt{A}[j] \oplus r_i$
   (b) output 1 if the last k bits of v are equal to 0, otherwise continue;
3. output 0.

Theorem B.2. *If π is a pseudo-random permutation, then SI is IND2-CKA.*

Proof. We show that if there exists a polynomial-size adversary $\mathcal{A}$ that wins in a $\mathbf{CKA}_{\mathsf{SI},\mathcal{A}}(k)$ experiment with non-negligible probability over 1/2, then there exists a polynomial-size adversary $\mathcal{B}$ that distinguishes whether a permutation π is random or pseudo-random.

$\mathcal{B}$ begins by simulating $\mathcal{A}$ as follows. It initializes a counter ctr to 0 and, given a document collection $\mathbf{D} = (D_1, \ldots, D_n)$ from $\mathcal{A}$, it returns a set of indexes $(I_1, \ldots, I_n)$ such that I_i is the result of running the Index algorithm with document D_i, counter ctr and where the PRP is replaced with oracle queries to π. For any trapdoor query w from $\mathcal{A}$, $\mathcal{B}$ returns $t = (\pi(w||1), \ldots, \pi(w||\mathsf{ctr}))$.

[6] We assume that $\mathtt{A}$ is "augmented" with an indirect addressing capability, namely, the ability to map $|\Delta|$ values from an exponential-size domain into its entries. See the construction in Section 5.1 for an efficient way to achieve this.

After polynomially many queries, $\mathcal{A}$ outputs two documents D_0^* and D_1^* subject to the following restrictions: $D_0^* \in \mathbf{D}$, $D_1^* \subseteq \Delta$, $|D_0^* \setminus D_1^*| \neq 0$ and $|D_1^* \setminus D_0^*| \neq 0$; and no word in $D_0^* \ominus D_1^*$ was used as a trapdoor query.

$\mathcal{B}$ then samples a bit b uniformly at random and constructs an index I_b as above. It returns I_b to $\mathcal{A}_2$ and answers its remaining Trpdr queries as before. After polynomially many queries, $\mathcal{A}$ outputs a bit b' and if $b' = b$ then $\mathcal{B}$ answers its own challenge indicating that π is a pseudo-random permutation; otherwise it indicates that π is a random permutation.

Clearly $\mathcal{B}$ is polynomial-size since $\mathcal{A}$ is. Notice that if π is a random permutation then whether $b = 0$ or $b = 1$, the index returned to $\mathcal{A}_2$ is a d-element array filled with $(\ell + k)$-bit random strings. Similarly, notice that since $\mathcal{A}$ is only allowed to query on keywords in $D_0^* \cap D_1^*$, the trapdoors returned by $\mathcal{B}$ are the same whether $b = 0$ or $b = 1$. It follows then that the probability that $\mathcal{A}$ succeeds in outputting $b' = b$ is at most $1/2$. On the other hand, if π is a pseudo-random permutation then $\mathcal{A}$'s view while being simulated is exactly the view it would have during a $\mathbf{CKA}_{\mathsf{SI},\mathcal{A}}(k)$ experiment. Therefore, by our initial assumption, $\mathcal{A}_2$ will succeed with non-negligible probability over $1/2$. It follows then that $\mathcal{B}$ will succeed in distinguishing whether π is random or pseudo-random with non-negligible probability.

■

Notice that while SI's trapdoors do not leak any information about the underlying keyword (since the trapdoors are generated using a pseudo-random permutation), the Search algorithm leaks the entire keyword. Clearly then, SI cannot be used as a secure SSE scheme.

Simulation-based SSE definitions. In [18] a simulation-based security definition for SSE is proposed that is intended to be stronger than IND2-CKA in the sense that it requires a scheme to have secure trapdoors. Unfortunately, it turns out that this definition can be trivially satisfied by any SSE scheme, even one that is insecure.

Definition B.3 ([18]). *For all $q \in \mathbb{N}$, for all* PPT *adversaries $\mathcal{A}$, all sets H composed of a document collection $\mathbf{D}$ and q keywords $(w_1, \ldots, w_q)$, and all functions f, there exists a* PPT *algorithm (simulator) $\mathcal{S}$ such that*

$$|\Pr[\mathcal{A}(C_q) = f(H)] - \Pr[\mathcal{S}(\{\mathcal{E}(\mathbf{D}), \mathbf{D}(w_1), \ldots, \mathbf{D}(w_q)\}) = f(H)]| \leq \mathsf{negl}(k),$$

where C_q is the entire communication the server receives up to the q^{th} query[7] *, $\mathcal{E}(\mathbf{D})$ is the encryption of the document collection (either as a single ciphertext or n ciphertexts), and k is the security parameter.*

Note that the order of the quantifiers in the definition imply that the algorithm $\mathcal{S}$ can depend on H. This means that for any q and any H, there will *always* exist a simulator that can satisfy the definition. This issue can be easily corrected in one of two ways: either by changing the order of the quantifiers and requiring that for all $q \in \mathbb{N}$, for all adversaries, for all functions, there exists a simulator such that for all sets H, the inequality in Definition B.3 holds; or by requiring that the inequality hold over all distributions over the set $2^{\Delta} \times \Delta^q$.

As mentioned in Section 1, Definition B.3 is inherently non-adaptive. Consider the natural way of using searchable encryption, where at time $t = 0$ a user submits an index to the server, then at time $t = 1$ performs a search for word w_1 and receives the set of documents $\mathbf{D}(w_1)$, at time $t = 2$ performs a search for word w_2 and receives the set of documents $\mathbf{D}(w_2)$, and so on until q searches are performed (i.e., until $t = q$). Our intuition about secure searchable encryption clearly tells us that at

[7] While the original definition found in [18] defines C_q to be the entire communication the server receives *before* the q^{th} query, we define it differently in order to stay consistent with the rest of our paper. Note that this in no way changes the meaning of the definition.

time $t = 0$ the adversary (i.e., the server) should not be able to learn any partial information about the documents from the index (beyond, perhaps, the number of documents it contains). Similarly, at time $t = 1$ the adversary should not be able to learn any partial information about the documents and w_1 from the index and the trapdoor for w_1 beyond what it can learn from $\mathbf{D}(w_1)$. More generally, at time $t = i$, where $1 \leq i \leq q$, the adversary should not be able to recover any partial information about the documents and words w_1 through w_i from the index and the corresponding trapdoors beyond what it can learn from the trace of the history.

Returning to Definition B.3, notice that for a fixed $q \in \mathbb{N}$, the simulator is required to simulate $\mathcal{A}(C_q)$ when only given the encrypted documents and the search outcomes of the q queries. But even if we are able to describe such a simulator, the only conclusion we can draw is that the *entire* communication C_q leaks nothing beyond the outcome of the q queries. We cannot, however, conclude that the index can be simulated at time $t = 0$ given only the encrypted documents; or that the index and trapdoor for w_1 can be simulated at time $t = 1$ given only the encrypted documents and $\mathcal{D}(w_1)$. We note that the fact that Definition B.3 holds for all $q \in \mathbb{N}$, does not imply the previous statements since, for each different q, the underlying algorithms used to generate the elements of C_q (i.e., the encryption scheme and the SSE scheme) might be used under a different secret key. Indeed, this assumption is implicit in the security proofs of the two constructions presented in [18], where for each $q \in \mathbb{N}$, the simulator is allowed to generate a different index (when $q \geq 0$) and different trapdoors (when $q \geq 1$).