

# Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions

REZA CURTMOLA\*    JUAN GARAY†    SENY KAMARA‡    RAFAIL OSTROVSKY§

## 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 review existing security definitions, pointing out their shortcomings, and propose two new stronger definitions which we prove equivalent. 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 [KBC<sup>+</sup>00, ABC<sup>+</sup>02, MMGC02] 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 [SWP00, Goh03, BC04, BdCOP04, CM05]; 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 [ISA02]. In addition, it can allow services such as Google Desktop [GD] to offer valuable features (e.g., the ability of searching a client’s data across several computers) without sacrificing the client’s privacy.

Searchable encryption can be achieved securely in its full generality using the work of Ostrovsky and Goldreich on *oblivious RAMs* [Ost90, GO96]. While oblivious RAMs hide all information (including the access pattern) from a remote and potentially malicious server, this comes at the cost of a logarithmic number of rounds of interaction for each read and write. In the same paper, they show a 2-round solution, but with considerably larger square-root overhead. Therefore, the previously mentioned work on searchable encryption tries to achieve more efficient solutions (typically in one or two rounds) by weakening the privacy guarantees (e.g., revealing the access pattern).

We start by examining the definition of what it means to completely reveal the user’s access and search patterns (precise definitions below) while “hiding everything else,” and show that the

---

\*Department of Computer Science, Johns Hopkins University. [crix@cs.jhu.edu](mailto:crix@cs.jhu.edu). This work was partly done while the author was at Bell Labs.

†Bell Labs – Lucent Technologies. [garay@research.bell-labs.com](mailto:garay@research.bell-labs.com)

‡Department of Computer Science, Johns Hopkins University. [seny@cs.jhu.edu](mailto:seny@cs.jhu.edu). Supported by a Bell Labs Graduate Research Fellowship.

§Department of Computer Science, UCLA. [rafail@cs.ucla.edu](mailto:rafail@cs.ucla.edu).

existing security definitions have several important limitations. Additionally, we show that the current definitions only achieve what we call *non-adaptive* SSE security, while the more natural usage of searchable encryption calls for *adaptive* security (a notion that we make precise in Section 3). We propose new security definitions for both the non-adaptive and adaptive cases, and present efficient constructions for both based on any one-way function.

Our first construction is the most efficient non-adaptive SSE scheme to date in terms of computation on the server, and incurs a minimal (i.e., constant) cost for the user. Our second construction achieves adaptive security, which was not previously achieved by any constant-round solution. (Later on we perform a detailed comparison between our constructions and previous work—see Table 1.)

We also extend the problem of SSE to the multi-user setting, where a client wishes to allow an authorized group of users to search through its document collection.

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 searches.

**On different models for private search** In recent years, there has been some confusion regarding three distinct models for searching with privacy: 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.

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 [Ost90, GO96].

In the setting of searching 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 [BdCOP04, ABC<sup>+</sup>05]. 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, Ostrovsky and Persiano [BdCOP04] reveals the user’s access pattern, recently Boneh, Kushilevitz, Ostrovsky and Skeith [BKOS07] 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 than the best private-key solutions.

Recently, Bellare, Boldyreva and O’Neill [BBO06] introduced the notion of asymmetric *efficiently searchable encryption* (ESE) and proposed three constructions in the random oracle model. Unlike PEKS, asymmetric 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.

In single-database private information retrieval, (or PIR) introduced by Kushilevitz and Ostrovsky [KO97], they show how a user can retrieve data from a server containing *unencrypted* data without revealing the access pattern and with total communication less than the data size. This was extended to keyword searching, including searching on streaming data [OS05]. 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 [IKOS04, IKOS06], 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.

**Related work.** We already mentioned the work on oblivious RAMs [GO96]. In an effort to reduce the round complexity associated with oblivious RAMs, Song, Wagner and Perrig [SWP00] 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 [Goh03] and of Chang and Mitzenmacher [CM05], 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 [Blo70] 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 shortcomings in more detail in Section 3 and Appendix B

As mentioned above, encryption with keyword search has also been considered in the public-key setting [BdCOP04, ABC<sup>+</sup>05], 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. Asymmetric ESE [BBO06] achieves comparable efficiency, but at the price of providing weaker security guarantees. 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 asymmetric ESE, with the added benefit of allowing the owner to revoke search privileges.

Naturally, SSE can also be viewed as an instance of secure two-party/multi-party computation [Yao82, GMW87, BOGW88]. However, the weakening and refinement of the privacy requirement (more on this below) as well as efficiency considerations (e.g., [KO04]), mandate a specialized treatment of the problem, both at the definitional and construction levels.<sup>1</sup>

**Subtleties of SSE security definitions.** So far, establishing correct security definitions for searchable encryption has been elusive. Clearly, as we have discussed, one could use the general definitions from oblivious RAMs, but subsequent work (including ours) examines if more efficient schemes can be achieved by revealing *some* information. The first difficulty seems to be in correctly capturing this intuition in a security definition. In the literature, this has typically been characterized as the requirement that nothing be leaked beyond the outcome of a search (e.g., [SWP00, CM05]); however, a more accurate description of the security guarantees in previous work is that nothing is leaked beyond the outcome and the *pattern* of a search, where the pattern of a search is any information that can be derived from knowing whether two searches were performed for the same word or not.

---

<sup>1</sup>Indeed, some of the results we show—equivalence of SSE security definitions (Section 3)—are known not to hold for the general secure multi-party computation case.

The second issue seems to be in appropriately capturing the adversary’s power. In fact, while Song *et al.* attempt to prove their construction secure, the definition implicitly used in their work is that of a classical encryption scheme, where the adversary is not allowed to perform searches. This was partly rectified by Goh [Goh03], whose definition of secure indexes (IND2-CKA) allows the adversary to access the index and trapdoor oracles. While much work on searchable encryption uses IND2-CKA as a security definition [GSW04, PKL04, BKM05], we note that it was never intended as such. In fact, Goh notes that IND2-CKA does not explicitly require trapdoors to be secure, which is an important requirement for any searchable encryption scheme. To remedy this, one might be tempted to introduce a second definition that exclusively guarantees the semantic security of trapdoors. One would then prove a construction secure under both IND2-CKA, and the new definition. While this might seem like a workable (though cumbersome) idea, the straightforward approach of requiring trapdoors to be indistinguishable does not work. In fact, as we show in Appendix B, SSE schemes can be built with trapdoors that, taken independently, leak no partial information about the word being queried, but when combined with an index allow an adversary to recover the entire word. This illustrates that the security of indexes and the security of trapdoors are intrinsically linked.

This is indeed the approach taken by Chang and Mitzenmacher [CM05], who propose a simulation-based definition that aims to guarantee privacy for indexes *and* trapdoors. Similarly to the classical definition of semantic security for encryption [GM84], they require that anything that can be computed from the index and the trapdoors for various queries, can be computed from the search outcome of those queries. However, while the intuition seems correct, in the case of searchable encryption one must also take care in describing *how* the search queries are generated. In particular, whether they can be made adaptively (i.e., after seeing the outcome of previous queries) or non-adaptively (i.e., without seeing the outcome of any queries). This distinction is important because it leads to security definitions that achieve drastically different privacy guarantees. Indeed, while non-adaptive definitions only guarantee security to clients who generate all their queries at once, adaptive definitions guarantee privacy even to clients who generate queries as a function of previous search outcomes. Unfortunately, as we discuss in Section 3, the definition presented in [CM05] is not only *non-adaptive*, but can be trivially satisfied by any SSE scheme, even one that is insecure.

**Our results.** We now summarize the contributions of this work.

1. We review existing security definitions for searchable encryption, including IND2-CKA [Goh03] and the simulation-based definition in [CM05], and highlight their shortcomings. Specifically, we point out that IND2-CKA is not an adequate notion of security for SSE and then highlight (and fix) technical issues with Chang and Mitzenmacher’s simulation-based definition. We address both these issues by proposing new indistinguishability and simulation-based definitions that provide security for both indexes and trapdoors, and show their equivalence.
2. 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 can 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 [SWP00, Goh03, CM05], these definitions only guarantee security for users that perform all their searches *at once*. We address this by introducing indistinguishability and simulation-based definitions in the adaptive setting and show that they are equivalent.
3. We present two constructions which we prove secure under the 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 on the server that is

Properties	[Ost90, GO96]	[Ost90, GO96]-light	[SWP00]	[Goh03]	[CM05]	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 document 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.

proportional to the actual number of documents that contain the queried word, requires constant storage on the client, and linear (in the size of the document collection) storage on the server. While the construction in [Goh03] also performs searches in one round, it can induce false positives, which is not the case for our construction. Additionally, all the constructions in [Goh03, CM05] require the server to perform an amount of work proportional to 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 by Goh’s construction). 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 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 [CM05]. We point out an optimization which lowers the communication size per query from linear to logarithmic in the number of updates.

**4. 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 [BEG<sup>+</sup>91] and universal arguments [BG02] 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, and delay this extension to the full version

of the paper.

## 2 Preliminaries

Let  $\Delta = \{w_1, \dots, w_d\}$  be a dictionary of  $d$  words, and  $2^\Delta$  be the set of all possible documents. Further, let  $\mathcal{D} \subseteq 2^\Delta$  be a collection of  $n$  documents  $\mathcal{D} = (D_1, \dots, D_n)$  and  $2^{2^\Delta}$  be the set of all possible document collections. Let  $\text{id}(D)$  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  $\mathcal{D}(w)$  the lexicographically ordered list consisting of the identifiers of all documents in  $\mathcal{D}$  that contain the word  $w$ . We sometimes refer to  $\mathcal{D}(w)$  as the *outcome of a search* for  $w$  and to the sequence  $(\mathcal{D}(w_1), \dots, \mathcal{D}(w_n))$  as the *access pattern* of a client. We also define the *search pattern* of a client as any information that can be derived from knowing whether two arbitrary searches were performed for the same word or not.

We write  $x \stackrel{R}{\leftarrow} \mathcal{X}$  to represent an element  $x$  being sampled from a distribution  $\mathcal{X}$  and  $x \stackrel{R}{\leftarrow} X$  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  $\parallel$  to mean string concatenation. We call a function  $\nu : \mathbb{N} \rightarrow \mathbb{N}$  negligible if for every polynomial  $p(\cdot)$  and all sufficiently large  $k$ ,  $\nu(k) < \frac{1}{p(k)}$ .

**Model.** The participants in a single-user searchable encryption scheme include a user that wishes to store an encrypted document collection  $\mathcal{D} = (D_1, \dots, D_n)$  on an honest-but-curious server  $S$ , while preserving the ability to search through them. We note that while we choose, for ease of exposition, to limit searches to be over documents, any SSE scheme can be trivially extended to search over lists of arbitrary keywords associated with the documents.

The participants in a multi-user searchable encryption scheme include a trusted owner  $O$ , an honest-but-curious server  $S$ , and a set of users  $\mathbb{N}$ .  $O$  owns  $\mathcal{D}$  and wants to grant and revoke searching privileges to a subset of users in  $\mathbb{N}$ . We let  $\mathbb{G} \subseteq \mathbb{N}$  be the set of users allowed to search. We assume that currently non-revoked users behave honestly.

More formally, throughout this paper, (authorized) users are modeled as probabilistic polynomial-time Turing machines, while adversaries are modeled as non-uniform Turing machines. On the other hand, simulators are uniform Turing machines, but they might become non-uniform by running a non-uniform adversary as a subroutine.

**Basic primitives.** A symmetric encryption scheme is a set of three polynomial-time algorithms  $(\mathcal{G}, \mathcal{E}, \mathcal{D})$  such that  $\mathcal{G}$  takes a security parameter  $k$  in unary and returns a secret key  $K$ ;  $\mathcal{E}$  takes a key  $K$  and an  $n$ -bit message  $m$  and returns a ciphertext  $c$ ;  $\mathcal{D}$  takes a key  $K$  and a ciphertext  $c$  and returns  $m$  if  $K$  was the key under which  $c$  was produced. Informally, a symmetric encryption scheme is considered secure if the ciphertexts it outputs do not leak any partial information about the plaintext even to an adversary that can adaptively query an encryption and a decryption oracle. We recall the formal definition of security for symmetric encryption schemes in Appendix A.

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. (Also in Appendix A.)

## 3 Revisiting Searchable Symmetric Encryption Definitions

We begin by reviewing the formal definition of a SSE scheme.

**Definition 3.1** (Searchable Symmetric Encryption Scheme (SSE)). *A SSE scheme is a collection of four polynomial-time algorithms (Keygen, BuildIndex, Trapdoor, Search) such that:*

$\text{Keygen}(1^k)$  is a probabilistic key generation algorithm that is run by the user to setup the scheme. It takes a security parameter  $k$ , and returns a secret key  $K$  such that the length of  $K$  is polynomially bounded in  $k$ .

$\text{BuildIndex}(K, \mathcal{D})$  is a (possibly probabilistic) algorithm run by the user to generate indexes. It takes a secret key  $K$  and a polynomially bounded in  $k$  document collection  $\mathcal{D}$  as inputs, and returns an index  $\mathcal{I}$  such that the length of  $\mathcal{I}$  is polynomially bounded in  $k$ .

$\text{Trapdoor}(K, w)$  is run by the user to generate a trapdoor for a given word. It takes a secret key  $K$  and a word  $w$  as inputs, and returns a trapdoor  $T_w$ .

$\text{Search}(\mathcal{I}, T_w)$  is run by the server  $S$  in order to search for the documents in  $\mathcal{D}$  that contain word  $w$ . It takes an index  $\mathcal{I}$  for a collection  $\mathcal{D}$  and a trapdoor  $T_w$  for word  $w$  as inputs, and returns  $\mathcal{D}(w)$ , the set of identifiers of documents containing  $w$ .

**A correct intuition.** While security for searchable encryption is typically characterized as the requirement that nothing be leaked beyond the outcome of a search (i.e., the identifiers of the documents returned from a search), we are not aware of any previous work on SSE that satisfies this intuition. In fact, with the exception of oblivious RAMs, all the constructions in the literature leak, in addition to the search outcomes, the user’s search pattern. This is clearly the case for the schemes presented in [SWP00, Goh03, CM05] since their trapdoors are deterministic. Therefore, a more accurate characterization of the security notion achieved (or rather, sought) for SSE is that nothing should be leaked beyond the outcome and the *pattern* of a sequence of searches.

Having clarified our intuition, it remains to precisely describe our adversarial model. SSE schemes are typically used in the following manner: the client generates an index from its document collection, stores the index and the encrypted documents on the server, and finally, performs various search queries. Here, it is important to note that the user may or may not generate its word queries depending on 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 query generation is important because it gives rise to definitions that achieve different privacy guarantees: non-adaptive definitions can only guarantee security to clients who generate their queries in one batch, while adaptive definitions can guarantee privacy even to clients who generate queries 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 schemes: indistinguishability against chosen-keyword attacks (IND2-CKA), introduced by Goh [Goh03]<sup>2</sup>, and a simulation-based definition introduced by Chang and Mitzenmacher [CM05].<sup>3</sup>

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

We remark that Goh’s work addresses a larger problem than searchable encryption, namely that of secure indexes, which are secure data structures that have many uses, only one of which is searchable encryption. And as Goh remarks (*cf.* Note 1, p. 5 of [Goh03]), 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 the semantic security of trapdoors, this cannot be done in a straightforward manner. Indeed, first proving

---

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

<sup>3</sup>We note that, unlike the latter and our own definitions (see below), IND2-CKA applies to indexes that are built for individual documents, as opposed to indexes built from entire document collections.

that an SSE scheme is IND2-CKA and then proving that its trapdoors are semantically secure *does not* imply that an adversary cannot recover the word being queried. As we show in Appendix B, SSE schemes can be built with trapdoors that, taken independently, leak no partial information about the word being queried, but when combined with an index allow an adversary to recover the word. This illustrates that the security of indexes and the security of trapdoors are inherently linked.

Regarding existing simulation-based definitions, in [CM05] Chang and Mitzenmacher provide a definition of security for SSE that is intended to be stronger than IND2-CKA in the sense that it requires a scheme to output 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 (see Appendix B).

**Our security definitions.** We now address the above issues. Before stating our definitions for SSE, we introduce three auxiliary notions which we will make use of.

Above, we mentioned our (and previous work’s) willingness to let the outcome and the pattern of a sequence of searches be known to the adversary (i.e., the server) in order to achieve greater efficiency. This can be more formally specified as follows. First, we note that an interaction between the client and the server will be determined by a document collection and a set of words that the client wishes to search for (and that we wish to hide from the adversary); we call an instantiation of such an interaction a *history*.

**Definition 3.2** (History). *Let  $\Delta$  be a dictionary. A history  $H_q$ ,  $H_q \in 2^{2^\Delta} \times \Delta^q$ , is an interaction between a client and a server over  $q$  queries. The partial history  $H_q^t \in 2^{2^\Delta} \times \Delta^t$  of a given history  $H_q = (\mathcal{D}, w_1, \dots, w_q)$ , is the sequence  $H_q^t = (\mathcal{D}, w_1, \dots, w_t)$ , where  $t \leq q$ .*

Given a history, we refer to what the adversary actually gets to “see” during an interaction as the history’s *view*. In particular, the view will consist of the index (of the document collection) and the trapdoors (of the queried words). It will also contain some additional common information, such as the number of documents in the collection and their ciphertexts). However (if done properly) the view (i.e., the index and the trapdoors) should not reveal any information about the history (i.e., the documents and the queried words) besides the outcome and the pattern of the searches (i.e., the information we are willing to leak). Let  $\mathcal{I}_{\mathcal{D}}$  be the index for  $\mathcal{D}$  generated under secret key  $K$ , and  $T_i$ ,  $1 \leq i \leq q$ , be the trapdoors for the words queried in  $H_q$ .

**Definition 3.3** (View). *Let  $\mathcal{D}$  be a collection of  $n$  documents and  $H_q = (\mathcal{D}, w_1, \dots, w_q)$  be a history over  $q$  queries. An adversary’s view of  $H_q$  under secret key  $K$  is defined as  $V_K(H_q) = (\text{id}(D_1), \dots, \text{id}(D_n), \mathcal{E}(D_1), \dots, \mathcal{E}(D_n), \mathcal{I}_{\mathcal{D}}, T_1, \dots, T_q)$ . The partial view  $V_K^t(H_q)$  of a history  $H_q$  under secret key  $K$  is the sequence  $V_K^t(H_q) = (\text{id}(D_1), \dots, \text{id}(D_n), \mathcal{E}(D_1), \dots, \mathcal{E}(D_n), \mathcal{I}_{\mathcal{D}}, T_1, \dots, T_t)$ , where  $t \leq q$ .*

We note that  $K$  in  $V_K(H_q)$  and  $V_K^t(H_q)$  only refers to the secret key for the SSE scheme and not to the keys used to encrypt the documents.

This leads to the notion of the *trace* of an interaction/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 query word in the history (i.e., the outcome of each search), and information that describes which trapdoors in the view correspond to the same underlying words in the history (i.e., the pattern of the searches). According to our intuitive formulation of security, this should be no more than the outcome of a search and the user’s search pattern. 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.<sup>4</sup> The trace also includes the user’s search pattern. More formally, a user’s search pattern  $\Pi_q$  can be

---

<sup>4</sup>On the other hand, if we wish not to disclose the size of the documents, this can be trivially achieved by “padding” each plaintext document such that all documents have a fixed size and omitting the document sizes from the trace.



thought of as a symmetric binary matrix where  $\Pi_q[i, j] = 1$  if  $w_i = w_j$ , and  $\Pi_q[i, j] = 0$  otherwise, for  $1 \leq i, j \leq q$ .

**Definition 3.4** (Trace). *Let  $\mathcal{D}$  be a collection of  $n$  documents and  $H_q = (\mathcal{D}, w_1, \dots, w_q)$  be a history over  $q$  queries. The trace of  $H_q$  is the sequence  $\text{Tr}(H_q) = (\text{id}(D_1), \dots, \text{id}(D_n), |D_1|, \dots, |D_n|, \mathcal{D}(w_1), \dots, \mathcal{D}(w_q), \Pi_q)$ .*

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

**Definition 3.5** (Non-Adaptive Indistinguishability Security for SSE). *A SSE scheme is secure in the sense of non-adaptive indistinguishability if for all  $q \in \mathbb{N}$ , for all (non-uniform) probabilistic polynomial-time adversaries  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ , for all polynomials  $p$  and all sufficiently large  $k$ :*

$$\Pr \left[ \begin{array}{l} b' = b : K \leftarrow \text{Keygen}(1^k); (H_0, H_1, \text{state}) \leftarrow \mathcal{A}_1; \\ b \stackrel{R}{\leftarrow} \{0, 1\}; b' \leftarrow \mathcal{A}_2(V_K(H_b), \text{state}) \end{array} \right] < \frac{1}{2} + \frac{1}{p(k)}$$

where  $(H_0, H_1)$  are histories over  $q$  queries such that  $\text{Tr}(H_0) = \text{Tr}(H_1)$ , where **state** is a polynomially bounded string that captures  $\mathcal{A}_1$ 's state, and the probability is taken over the internal coins of **Keygen**,  $\mathcal{A}$ , and the underlying **BuildIndex** algorithm.

**Definition 3.6** (Non-Adaptive Semantic Security for SSE). *A SSE scheme is non-adaptively semantically secure if for all  $q \in \mathbb{N}$  and for all (non-uniform) probabilistic polynomial-time adversaries  $\mathcal{A}$ , there exists a (non-uniform) probabilistic polynomial-time algorithm (the simulator)  $\mathcal{S}$  such that for all traces  $\text{Tr}_q$  of length  $q$ , all polynomially samplable distributions  $\mathcal{H}_q$  over  $\{H_q \in 2^{2^\Delta} \times \Delta^q : \text{Tr}(H_q) = \text{Tr}_q\}$  (i.e., the set of histories with trace  $\text{Tr}_q$ ), all functions  $f : \{0, 1\}^m \rightarrow \{0, 1\}^{\ell(m)}$  (where  $m = |H_q|$  and  $\ell(m) = \text{poly}(m)$ ), all polynomials  $p$  and sufficiently large  $k$ :*

$$|\Pr[\mathcal{A}(V_K(H_q)) = f(H_q)] - \Pr[\mathcal{S}(\text{Tr}(H_q)) = f(H_q)]| < \frac{1}{p(k)},$$

where  $H_q \stackrel{R}{\leftarrow} \mathcal{H}_q$ ,  $K \leftarrow \text{Keygen}(1^k)$ , and the probabilities are taken over  $\mathcal{H}_q$  and the internal coins of **Keygen**,  $\mathcal{A}$ ,  $\mathcal{S}$  and the underlying **BuildIndex** algorithm.

We now prove that our two definitions of security for non-adaptive adversaries are equivalent. For this, we follow the approach of Goldwasser and Micali [GM84] and Goldreich [Gol93].

**Theorem 3.7.** *Non-adaptive indistinguishability security of SSE is equivalent to non-adaptive semantic security of SSE.*

*Proof.* (Non-adaptive semantic security  $\Rightarrow$  non-adaptive indistinguishability security.) We show that given a non-uniform adversary  $\mathcal{A}$  that breaks the security of a SSE scheme in the sense of non-adaptive indistinguishability, we can construct a non-uniform adversary  $\mathcal{B}$  that breaks the security of the scheme in the sense of non-adaptive semantic security.

If a SSE scheme (**Keygen**, **BuildIndex**, **Trapdoor**, **Search**) is non-adaptively distinguishable, then there exists a value  $q \in \mathbb{N}$ , a non-uniform probabilistic polynomial-time adversary  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$  and two histories over  $q$  queries  $(H_0, H_1)$  where  $\text{Tr}(H_0) = \text{Tr}(H_1)$ , such that  $\mathcal{A}$  can distinguish with non-negligible probability over  $\frac{1}{2}$  any view that results from a history  $H_b \stackrel{R}{\leftarrow} \{H_0, H_1\}$  and a key  $K \leftarrow \text{Keygen}(1^k)$ .

Therefore, given a view  $V_K(H_b)$  such that  $H_b \stackrel{R}{\leftarrow} \{H_0, H_1\}$  and that  $K \leftarrow \text{Keygen}(1^k)$ ,  $\mathcal{B}$  can use  $\mathcal{A}$  to compute  $f$ , where  $f$  is the identity function, on  $H_b$  as follows. It begins by running  $\mathcal{A}_1$ , which outputs  $(H_0, H_1, \text{state})$ .  $\mathcal{B}$  then runs  $\mathcal{A}_2(V_K(H_b), \text{state})$  and outputs  $H_0$  if  $b = 0$  and  $H_1$  if  $b = 1$ . From  $\mathcal{B}$ 's description, it follows that  $\mathcal{B}$  will output  $f(H_b) = H_b$  with probability equal to  $\mathcal{A}$ 's distinguishing probability which, by assumption, is non-negligibly greater than  $\frac{1}{2}$ . On the other hand, no algorithm can guess  $f(H_b)$  with probability larger than  $\frac{1}{2}$  when only given  $\text{Tr}(H_b)$ , since  $\text{Tr}(H_0) = \text{Tr}(H_1)$ .

(Non-adaptive indistinguishability security  $\Rightarrow$  non-adaptive semantic security.) We show that given a non-uniform adversary  $\mathcal{A}$  that breaks the security of a SSE scheme in the sense of non-adaptive semantic security, we can construct a non-uniform adversary  $\mathcal{B}$  that breaks the security of the scheme in the sense of non-adaptive indistinguishability.

If a SSE scheme is not non-adaptively semantically secure, then there exists a value  $q \in \mathbb{N}$  and a non-uniform probabilistic polynomial-time adversary  $\mathcal{A}$  such that for all simulators  $\mathcal{S}$  there exists a trace  $\text{Tr}_q$  of length  $q$ , a distribution  $\mathcal{H}_q$  over  $\{H \in 2^{2^\Delta} \times \Delta^q : \text{Tr}(H) = \text{Tr}_q\}$ , a function  $f : \{0, 1\}^m \rightarrow \{0, 1\}^{\ell(m)}$  (where  $\ell(m) = \text{poly}(m)$ ), and a polynomial  $p$  such that for infinitely many  $k$ 's

$$|\Pr[\mathcal{A}(V_K(H)) = f(H)] - \Pr[\mathcal{S}(\text{Tr}(H)) = f(H)]| \geq \frac{1}{p(k)}, \quad (1)$$

where  $H \stackrel{R}{\leftarrow} \mathcal{H}_q$  and  $K \leftarrow \text{Keygen}(1^k)$ . From this, it follows that there exists at least two histories  $(H_0, H_1)$  in  $\mathcal{H}_q$ 's support such that  $f(H_0) \neq f(H_1)$ —otherwise the simulator that always outputs the value  $v = f(H)$ , outputs  $f(H)$  with probability equal to  $\mathcal{A}$ 's. The indistinguishability adversary  $\mathcal{B}$  runs in two stages  $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ , where  $\mathcal{B}_1$  finds a triple  $(H_0, H_1, v)$ , and  $\mathcal{B}_2$  attempts to recover the challenge bit  $b$  with probability non-negligibly greater than  $\frac{1}{2}$ .

$\mathcal{B}_1$  begins by sampling two histories  $(H_0, H_1)$  from  $\mathcal{H}_q$ . We claim that with non-negligible probability there exists a value  $v$  in  $f$ 's range such that

$$|\Pr[\mathcal{A}(V_K(H_0)) = v] - \Pr[\mathcal{A}(V_K(H_1)) = v]| \geq \frac{1}{2 \cdot p(k)}, \quad (2)$$

where  $K \leftarrow \text{Keygen}(1^k)$ . To see why, consider the simulator  $\mathcal{S}^*$  that on input  $\text{Tr}(H_0)$  generates a key  $K \leftarrow \text{Keygen}(1^k)$ , samples a history  $H_1 \stackrel{R}{\leftarrow} \mathcal{H}_q$  and runs  $\mathcal{A}$  on  $V_K(H_1)$ . Since our initial assumption about  $\mathcal{A}$  states that Equation (1) must hold against all simulators, it must hold against  $\mathcal{S}^*$ . And according to  $\mathcal{S}^*$ 's definition, this means that

$$|\Pr[\mathcal{A}(V_K(H_0)) = f(H_0)] - \Pr[\mathcal{A}(V_K(H_1)) = f(H_0)]| \geq \frac{1}{p(k)}, \quad (3)$$

where  $H_0 \stackrel{R}{\leftarrow} \mathcal{H}_q$ ,  $H_1 \stackrel{R}{\leftarrow} \mathcal{H}_q$ ,  $K \leftarrow \text{Keygen}(1^k)$ . If we define the set  $B_q$  as

$$B_q = \left\{ H \in \text{sup}(\mathcal{H}_q) : |\Pr[\mathcal{A}(V_K(H_0)) = f(H_0)] - \Pr[\mathcal{A}(V_K(H)) = f(H_0)]| \geq \frac{1}{2 \cdot p(k)} \right\},$$

then by an averaging argument,  $\Pr_{H \in \mathcal{H}_q}[H \in B_q] \geq \frac{1}{2 \cdot p(k)}$ . Defining  $C_q$  to be

$$C_q = \left\{ H \in \text{sup}(\mathcal{H}_q) : \exists v \in \{0, 1\}^{\ell(m)} \text{ s.t. } |\Pr[\mathcal{A}(V_K(H_0)) = v] - \Pr[\mathcal{A}(V_K(H)) = v]| \geq \frac{1}{2 \cdot p(k)} \right\},$$

it follows that  $\Pr_{H \in \mathcal{H}_q}[H \in C_q] \geq \frac{1}{2 \cdot p(k)}$  since  $B_q \subseteq C_q$ . This proves our claim.

Having found a pair  $(H_0, H_1)$  such that there exists a value  $v$  satisfying Equation (2) with non-negligible probability, it remains for  $\mathcal{B}_1$  to find  $v$ . To this end, for each value  $v$  in  $f$ 's range,  $\mathcal{B}_1$

will estimate  $\Pr[\mathcal{A}(V_K(H_0)) = v]$  and  $\Pr[\mathcal{A}(V_K(H_1)) = v]$ . It will then choose the  $v$  for which the absolute value of the difference of the estimates is non-negligible.

More precisely,  $\mathcal{B}_1$  begins by constructing  $t = \text{poly}(k)$  views from each history. This results in two sequences of  $t$  views:

$$U_0 = (V_{K_1}(H_0), \dots, V_{K_t}(H_0)),$$

generated from  $H_0$ , where  $K_i \leftarrow \text{Keygen}(1^k)$  for  $1 \leq i \leq t$ ; and

$$U_1 = (V_{K_{t+1}}(H_1), \dots, V_{K_{2t}}(H_1)),$$

generated from  $H_1$ , where  $K_i \leftarrow \text{Keygen}(1^k)$  for  $t+1 \leq i \leq 2t$ . It then runs  $\mathcal{A}$  on each element of  $U_0$  and  $U_1$ , resulting in two sequences of  $t$  values in  $f$ 's range

$$W_0 = (\mathcal{A}(V_{K_1}(H_0)), \dots, \mathcal{A}(V_{K_t}(H_0))),$$

and

$$W_1 = (\mathcal{A}(V_{K_{t+1}}(H_1)), \dots, \mathcal{A}(V_{K_{2t}}(H_1))).$$

For each unique value  $v$  in  $W_0$  and  $W_1$ ,  $\mathcal{B}_1$  computes

$$\pi_0(v) = \frac{|\{w \in W_0 : w = v\}|}{t}$$

and

$$\pi_1(v) = \frac{|\{w \in W_1 : w = v\}|}{t}$$

Notice that for a fixed  $v$ ,  $\pi_0(v)$  and  $\pi_1(v)$  are estimates of  $\Pr[\mathcal{A}(V_K(H_0)) = v]$  and  $\Pr[\mathcal{A}(V_K(H_1)) = v]$ , respectively, taken over  $t = \text{poly}(k)$  samples. And if  $t$  is a large enough polynomial in  $k$  they will approximate  $\Pr[\mathcal{A}(V_K(H_0)) = v]$  and  $\Pr[\mathcal{A}(V_K(H_1)) = v]$  with high probability. After estimating the probabilities,  $\mathcal{B}_1$  searches  $W_0$  and  $W_1$  for an element  $v$  such that

$$|\pi_0(v) - \pi_1(v)| \geq \frac{1}{2 \cdot p(k)}.$$

Since the  $\pi$ 's are good estimators, and since we know from our claim that with non-negligible probability there exists a  $v \in \{0, 1\}^{\ell(m)}$  such that  $|\Pr[\mathcal{A}(V_K(H_0)) = v] - \Pr[\mathcal{A}(V_K(H_1)) = v]| \geq \frac{1}{2 \cdot p(k)}$ , such a  $v$  will be found with high probability.  $\mathcal{B}_1$  then outputs the triplet  $(H_0, H_1, \text{state})$ , where  $\text{state} = v$ .

Finally, when given challenge  $V_K(H_b)$ ,  $\mathcal{B}_2$  runs  $\mathcal{A}(V_K(H_b))$  and outputs 0 if  $\mathcal{A}$  returns  $v$  and 1 otherwise. From  $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ 's description, it follows that:

$$\begin{aligned} \Pr[b' = b] &= \Pr[b' = b \mid b = 0] \cdot \Pr[b = 0] + \Pr[b' = b \mid b = 1] \cdot \Pr[b = 1] \\ &= \frac{1}{2} \cdot (\Pr[\mathcal{A}(V_K(H_0)) = v] + \Pr[\mathcal{A}(V_K(H_1)) \neq v]) \\ &= \frac{1}{2} \cdot (\Pr[\mathcal{A}(V_K(H_0)) = v] + 1 - \Pr[\mathcal{A}(V_K(H_1)) = v]) \\ &\geq \frac{1}{2} + \frac{1}{4 \cdot p(k)} \end{aligned}$$

from which the theorem follows. ■

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  $(\mathcal{D}_0, \mathcal{D}_1)$  (subject to some constraints which we describe below), and receives the index of one of the collections  $\mathcal{D}_b$ ; it then submits two words  $(w_0, w_1)$  and receives the trapdoor of one of the words  $w_b$ . This process goes on until the adversary has submitted  $q$  queries and is then challenged to output the value of  $b$ .

**Definition 3.8** (Adaptive Indistinguishability Security for SSE). *A SSE scheme is secure in the sense of adaptive indistinguishability if for all  $q \in \mathbb{N}$ , for all (non-uniform) probabilistic polynomial-time adversaries  $\mathcal{A} = (\mathcal{A}_1, \dots, \mathcal{A}_{q+2})$ , for all polynomials  $p$  and sufficiently large  $k$ :*

$$\Pr \left[ \begin{array}{l} b' = b : K \leftarrow \text{Keygen}(1^k); b \xleftarrow{R} \{0, 1\}; \\ (\mathcal{D}_0, \mathcal{D}_1, \text{state}_1) \leftarrow \mathcal{A}_1; \\ (w_{1,0}, w_{1,1}, \text{state}_2) \leftarrow \mathcal{A}_2(\mathcal{I}_b, \text{state}_1); \\ \dots; \\ (w_{q,0}, w_{q,1}, \text{state}_{q+1}) \leftarrow \mathcal{A}_{q+1}(\mathcal{I}_b, T_{w_{1,b}}, \dots, T_{w_{q-1,b}}, \text{state}_q); \\ b' \leftarrow \mathcal{A}_{q+2}(V_K(H_b), \text{state}_{q+1}) \end{array} \right] < \frac{1}{2} + \frac{1}{p(k)}$$

where  $H_0 = (\mathcal{D}_0, w_{1,0}, \dots, w_{q,0})$ ,  $H_1 = (\mathcal{D}_1, w_{1,1}, \dots, w_{q,1})$ , and  $\text{Tr}(H_0) = \text{Tr}(H_1)$ . Also,  $\text{state}_i$  ( $1 \leq i \leq q+1$ ) is a polynomially bounded string that captures  $\mathcal{A}_i$ 's state, and the probability is taken over the internal coins of  $\text{Keygen}$ ,  $\mathcal{A}$ , and the underlying  $\text{BuildIndex}$  algorithm.

We now present our simulation-based definition, which is similar to the non-adaptive definition, except that for all queries  $0 \leq t \leq q$ , we require the simulator, given only a partial trace of the history, to simulate the adversary on a partial view of the same history. More precisely, the challenger picks a key  $K$  and a history  $H_q$  over  $q$  queries, and for each query  $t$  between 0 and  $q$ , requires the simulator to simulate, given only a partial trace of  $H_q$ , an adversary  $\mathcal{A}$  that is given a partial view of  $H_q$  under key  $K$ . If the adversary can be simulated for each query under the same key  $K$ , then it follows that during each query of an interaction between a user and a server, the SSE scheme leaks no partial information about the elements of the history beyond the outcome of the searches and the user's search pattern.

**Definition 3.9** (Adaptive Semantic Security for SSE). *A SSE scheme is adaptively semantically secure if for all  $q \in \mathbb{N}$  and for all (non-uniform) probabilistic polynomial-time adversaries  $\mathcal{A}$ , there exists a (non-uniform) probabilistic polynomial-time algorithm (the simulator)  $\mathcal{S}$  such that for all traces  $\text{Tr}_q$  of length  $q$ , all polynomially samplable distributions  $\mathcal{H}_q$  over  $\{H_q \in 2^{2^\Delta} \times \Delta^q : \text{Tr}(H_q) = \text{Tr}_q\}$  (i.e., the set of histories with trace  $\text{Tr}_q$ ), all functions  $f : \{0, 1\}^m \rightarrow \{0, 1\}^{\ell(m)}$  (where  $m = |H_q|$  and  $\ell(m) = \text{poly}(m)$ ), all  $0 \leq t \leq q$  and all polynomials  $p$  and sufficiently large  $k$ :*

$$\left| \Pr [\mathcal{A}(V_K^t(H_q) = f(H_q^t))] - \Pr [\mathcal{S}(\text{Tr}(H_q^t)) = f(H_q^t)] \right| < \frac{1}{p(k)}$$

where  $H_q \xleftarrow{R} \mathcal{H}_q$ ,  $K \leftarrow \text{Keygen}(1^k)$ , and the probabilities are taken over  $\mathcal{H}_q$  and the internal coins of  $\text{Keygen}$ ,  $\mathcal{A}$ ,  $\mathcal{S}$  and the underlying  $\text{BuildIndex}$  algorithm.

**Theorem 3.10.** *Adaptive indistinguishability security of SSE is equivalent to adaptive semantic security of SSE.*

The proof of this theorem is similar to (albeit more tedious than) that of Theorem 3.7; it will appear in the full version of this paper.

## 4 Efficient and Secure Searchable Symmetric Encryption

In this section we present our efficient SSE constructions, and state their security in terms of the definitions presented in Section 3. We start by introducing some additional notation and the data structures used by the constructions. Let  $\Delta'$ ,  $\Delta' \subseteq \Delta$ , be the set of distinct words that exist in the document collection  $\mathcal{D}$ . We assume that words in  $\Delta$  can be represented using at most  $p$  bits. Also, recall that  $\mathcal{D}(w)$  is the set of identifiers of documents in  $\mathcal{D}$  that contain word  $w$  ordered in lexicographic order.

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

### 4.1 An efficient SSE construction

We first give an overview of our one-round non-adaptive SSE construction. We associate a single index  $\mathcal{I}$  with a document collection  $\mathcal{D}$ . The index  $\mathcal{I}$  consists of two data structures:

- an array  $\mathbf{A}$ , in which we store in encrypted form the set  $\mathcal{D}(w)$ , for each word  $w \in \Delta'$ , and
- a look-up table  $\mathbf{T}$ , which contains information that enables one to locate and decrypt the appropriate elements from  $\mathbf{A}$ , for each word  $w \in \Delta'$ .

We start with a collection of linked lists  $\mathbf{L}_i$ ,  $w_i \in \Delta'$ , where the nodes of each  $\mathbf{L}_i$  are the identifiers of documents in  $\mathcal{D}(w_i)$ . We then write in the array  $\mathbf{A}$  the nodes of all lists  $\mathbf{L}_i$ , “scrambled” in a random order and encrypted with randomly generated keys. Before encryption, the  $j$ -th node of  $\mathbf{L}_i$  is augmented with information about the index in  $\mathbf{A}$  of the  $(j + 1)$ -th node of  $\mathbf{L}_i$ , together with the key used to encrypt it. In this way, given the position (index) in  $\mathbf{A}$  and the decryption key for the first node of a list  $\mathbf{L}_i$ , the server will be able to locate and decrypt all the nodes in  $\mathbf{L}_i$ . Note that by storing in  $\mathbf{A}$  the nodes of all lists  $\mathbf{L}_i$  in a random order, the size of each  $\mathbf{L}_i$  is hidden.

We now build a look-up table  $\mathbf{T}$  that allows one to locate and decrypt the first element of each list  $\mathbf{L}_i$ . Each entry in  $\mathbf{T}$  corresponds to a word  $w_i \in \Delta$  and consists of a pair  $\langle \text{address}, \text{value} \rangle$ . The field **value** contains the index in  $\mathbf{A}$  and the decryption key for the first element of  $\mathbf{L}_i$ . **value** is itself encrypted using the output of a pseudo-random function. The other field, **address**, is simply used to locate an entry in  $\mathbf{T}$ . The look-up table  $\mathbf{T}$  is managed using *indirect addressing* (described below).

The user computes both  $\mathbf{A}$  and  $\mathbf{T}$  based on the un-encrypted  $\mathcal{D}$ , and stores them on the server together with the encrypted  $\mathcal{D}$ . When the user wants to retrieve the documents that contain word  $w_i$ , it computes the decryption key and the address for the corresponding entry in  $\mathbf{T}$  and sends them to the server. The server locates and decrypts the given entry of  $\mathbf{T}$ , and gets the index in  $\mathbf{A}$  and the decryption key for the first node of  $\mathbf{L}_i$ . Since each element of  $\mathbf{L}_i$  contains information about the next element of  $\mathbf{L}_i$ , the server can locate and decrypt all the nodes of  $\mathbf{L}_i$ , which gives the identifiers in  $\mathcal{D}(w_i)$ .

**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  $\mathbf{T}$  are tuples  $\langle \text{address}, \text{value} \rangle$ , in which the **address** field is used as a *virtual address* to locate the entry in  $\mathbf{T}$  that contains some **value** field. Given a parameter  $p$ , a virtual address is from a domain of exponential size (i.e., from  $\{0, 1\}^p$ ). However, the maximum number of entries in a look-up table will be polynomial in  $p$ , so the number of virtual addresses that are used can be approximated as  $\text{poly}(p)$ . If, for a look-up table  $\mathbf{T}$ , the **address** field is from  $\{0, 1\}^p$ , the **value** field is from  $\{0, 1\}^v$  and there are at most  $m$  entries in  $\mathbf{T}$ , then we say  $\mathbf{T}$  is a  $(\{0, 1\}^p \times \{0, 1\}^v \times m)$  look-up table.

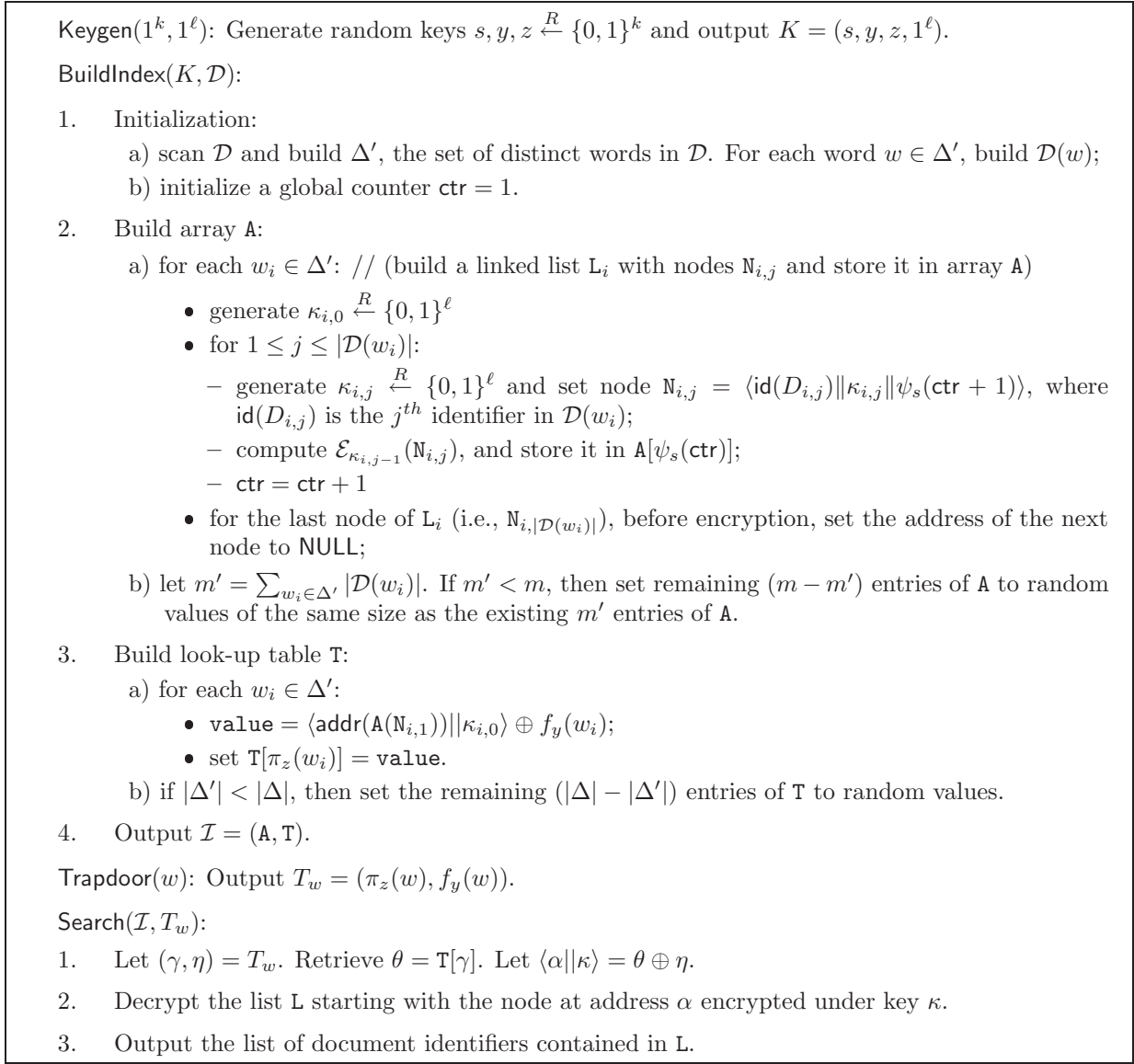


Figure 1: Efficient SSE construction (SSE-1)

Let **Addr** be the set of virtual addresses that are used for entries in a look-up table **T**. We can efficiently store **T** such that, when given a virtual address, it returns the associated **value** field. We achieve this by organizing the **Addr** set in a so-called *FKS dictionary* [FKS84], an efficient data structure for storage of sparse tables that requires  $O(|\text{Addr}|)$  ( $+o(|\text{Addr}|)$ ) storage and  $O(1)$  look-up time. In other words, given some virtual address **A**, we are able to tell if  $\mathbf{A} \in \text{Addr}$  and if so, return the associated **value** in constant look-up time. Addresses that are not in **Addr** are considered undefined.

**SSE-1 in detail.** We are now ready to proceed to the details of the construction. Let  $k, \ell$  be security parameters and let  $(\mathcal{G}, \mathcal{E}, \mathcal{D})$  be a semantically secure symmetric encryption scheme with  $\mathcal{E} : \{0, 1\}^\ell \times \{0, 1\}^r \rightarrow \{0, 1\}^r$ . In addition, we make use of one pseudo-random function  $f$  and two pseudo-random permutations  $\pi$  and  $\psi$  with the following parameters:

- $f : \{0, 1\}^k \times \{0, 1\}^p \rightarrow \{0, 1\}^{\ell + \log_2(m)}$ ;
- $\pi : \{0, 1\}^k \times \{0, 1\}^p \rightarrow \{0, 1\}^p$ ; and

$$- \psi : \{0, 1\}^k \times \{0, 1\}^{\log_2(m)} \rightarrow \{0, 1\}^{\log_2(m)}.$$

Let  $m$  be the total size of the plaintext document collection, expressed in *units*. A *unit* is the smallest possible size for a word (e.g. one byte).<sup>5</sup> Let  $\mathbf{A}$  be an array of size  $m$ . Let  $\mathbf{T}$  be a  $(\{0, 1\}^p \times \{0, 1\}^{\ell + \log_2(m)} \times |\Delta|)$  look-up table, managed using indirect addressing as described previously. Our construction SSE-1 = (Keygen, BuildIndex, Trapdoor, Search) is described in Fig. 1.

Consistent with our security definitions, SSE-1 reveals only the outcome and the pattern of a search, the total size of the encrypted document collection and the number of documents in  $\mathcal{D}$ . Recall that the array  $\mathbf{A}$  can be seen as a collection of linked lists  $L_i$ , where each  $L_i$  contains the identifiers of documents containing word  $w_i$ . Let  $m' = \sum_{w_i \in \Delta'} |L_i|$ . If, for all  $D_j \in \mathcal{D}$ , a word does not appear more than once in document  $D_j$ , it is clear that  $m = m'$ . If the size of  $\mathbf{A}$  is smaller than  $m$ , then the array  $\mathbf{A}$  reveals that at least one document in  $\mathcal{D}$  contains a word more than once. To avoid such leakage, we set the size of  $\mathbf{A}$  equal to  $m$  and fill the  $(m - m')$  remaining entries with random values. We follow the same line of reasoning for the look-up table  $\mathbf{T}$ , which has at least one entry for each distinct word in  $\mathcal{D}$ . To avoid revealing the number of distinct words in  $\mathcal{D}$ , we add additional  $(|\Delta| - |\Delta'|)$  entries in  $\mathbf{T}$ , filled with random values, such that the number of entries in  $\mathbf{T}$  is always equal to  $|\Delta|$ .

**Theorem 4.1.** *SSE-1 is a non-adaptively secure SSE scheme.*

*Proof.* For the proof, we follow the simulation-based approach of Definition 3.6. We describe a probabilistic polynomial-time simulator  $\mathcal{S}$  such that for all  $q \in \mathbb{N}$ , all probabilistic polynomial-time adversaries  $\mathcal{A}$ , all polynomially-bounded functions  $f$ , all distributions  $\mathcal{H}_q$ , given  $\text{Tr}(H_q)$ ,  $\mathcal{S}$  can simulate  $\mathcal{A}(V_K(H_q))$  with probability negligibly close to 1, where  $H_q \stackrel{R}{\leftarrow} \mathcal{H}_q$  and  $K \leftarrow \text{Keygen}(1^k)$ . More precisely, we show that  $\mathcal{S}(\text{Tr}(H_q))$  can generate a view  $V_q^*$  such that  $V_q^*$  is indistinguishable from  $V_K(H_q)$ . Recall that  $\text{Tr}(H_q) = \{1, \dots, n, |D_1|, \dots, |D_n|, \mathcal{D}(w_1), \dots, \mathcal{D}(w_q), \Pi_q\}$  and  $V_K(H_q) = \{1, \dots, n, \mathcal{E}(D_1), \dots, \mathcal{E}(D_n), \mathcal{I}_{\mathcal{D}}, T_{w_1}, \dots, T_{w_q}\}$ . Further, recall that the parameters of the pseudo-random function  $f$  and of the two pseudo-random permutations  $\pi, \psi$  used to instantiate the scheme are known to  $\mathcal{S}$ .

For  $q = 0$ ,  $\mathcal{S}$  builds the set  $V_0^* = \{1, \dots, n, e_1^*, \dots, e_n^*, \mathcal{I}^*\}$  such that  $e_i^* \stackrel{R}{\leftarrow} \{0, 1\}^{|D_i|}$  for  $1 \leq i \leq n$ ; and that  $\mathcal{I}^* = (\mathbf{A}^*, \mathbf{T}^*)$ , where  $\mathbf{A}^*$  and  $\mathbf{T}^*$  are generated as follows:

**Generating  $\mathbf{A}^*$ :** for  $1 \leq i \leq m$ ,  $\mathcal{S}$  generates a random string  $r_i \stackrel{R}{\leftarrow} \{0, 1\}^{\log_2(n) + \ell + \log_2(m)}$  and sets  $\mathbf{A}^*[i] = r_i$ .

**Generating  $\mathbf{T}^*$ :** for  $1 \leq i \leq |\Delta|$ ,  $\mathcal{S}$  generates pairs  $(a_i^*, c_i^*)$  such that  $a_i^* \stackrel{R}{\leftarrow} \{0, 1\}^p$ , where each  $a_i^*$  is distinct, and  $c_i^* \stackrel{R}{\leftarrow} \{0, 1\}^{\ell + \log_2(m)}$ . It then sets  $\mathbf{T}^*[a_i^*] = c_i^*$ .

We claim that no probabilistic polynomial-time adversary  $\mathcal{A}$  can distinguish between  $V_0^*$  and  $V_K(H_0) = \{1, \dots, n, \mathcal{E}(D_1), \dots, \mathcal{E}(D_n), \mathcal{I}_{\mathcal{D}}\}$  for any  $H_0$  (sampled from any distribution  $\mathcal{H}_0$ ) and  $K \leftarrow \text{Keygen}(1^k)$ , otherwise, by a standard hybrid argument,  $\mathcal{A}$  could distinguish between at least one of the elements of  $V_0^*$  and its corresponding element in  $V_K(H_0)$ . And we show that this is impossible since each element of  $V_0^*$  is indistinguishable from its corresponding element in  $V_K(H_0)$ . Since, clearly, the document identifiers in  $V_0^*$  and  $V_K(H_0)$  are indistinguishable, we consider only the other elements.

**Encrypted documents:** for  $1 \leq i \leq n$ ,  $\mathcal{E}(D_i)$  is indistinguishable from a random string  $e_i^* \stackrel{R}{\leftarrow} \{0, 1\}^r$  since  $(\mathcal{G}, \mathcal{E}, \mathcal{D})$  is semantically secure.

**Index:** Recall that  $\mathcal{I}_{\mathcal{D}} = (\mathbf{A}, \mathbf{T})$  and that  $\mathcal{I}^* = (\mathbf{A}^*, \mathbf{T}^*)$ . Since  $\mathbf{A}$  consists of  $m'$  semantically secure ciphertexts and  $m - m'$  random strings of the same size, and  $\mathbf{A}^*$  consists of  $m$  random strings of

<sup>5</sup>If the documents are not encrypted with a length preserving encryption scheme or if they are compressed before encryption, then  $m$  is the maximum between the total size of the plaintext  $\mathcal{D}$  and the total size of the encrypted  $\mathcal{D}$ .

the same size, it follows that each element in  $\mathbf{A}^*$  will be indistinguishable from its counterpart in  $\mathbf{A}$ .

Now recall that  $\mathbf{T}$  consists of  $|\Delta'|$  ciphertexts,  $c_i$ , generated by XOR-ing a message with the output of  $f$ , and of  $|\Delta - \Delta'|$  random values of size  $\ell + \log_2(m)$ .  $\mathbf{T}^*$ , on the other hand, consists of  $|\Delta|$  random string of the same size. It follows then, that each element of  $\mathbf{T}$  and  $\mathbf{T}^*$  are indistinguishable, otherwise one could distinguish between the output of  $f$  and a random string of size  $\ell + \log_2(m)$ .

For  $q > 0$ ,  $\mathcal{S}$  constructs  $V_q^* = \{1, \dots, n, e_1^*, \dots, e_n^*, \mathcal{I}^*, T_1^*, \dots, T_q^*\}$ , such that  $(e_1^*, \dots, e_n^*)$  are random values (as in the case of  $q = 0$ ) and that  $\mathcal{I}^* = (\mathbf{A}^*, \mathbf{T}^*)$ , where  $\mathbf{A}^*$  and  $\mathbf{T}^*$  are generated as follows:

**Generating  $\mathbf{A}^*$ :** To build  $\mathbf{A}^*$ ,  $\mathcal{S}$  runs step 2 of the `BuildIndex` algorithm of SSE-1 on the sets  $(\mathcal{D}(w_1), \dots, \mathcal{D}(w_q))$  (which it knows from its trace) with  $|\Delta'| = q$  and using different random strings of size  $\log_2(m)$  instead of  $\psi(\text{ctr})$ .

**Generating  $\mathbf{T}^*$ :** for  $1 \leq i \leq q$ ,  $\mathcal{S}$  generates random values  $\beta_i^* \xleftarrow{R} \{0, 1\}^{\ell + \log_2(m)}$  and  $a_i^* \xleftarrow{R} \{0, 1\}^p$ , and sets  $\mathbf{T}^*[a_i^*] = \langle \text{addr}_{\mathbf{A}^*}(N_{i,1}) \parallel \kappa_{i,0} \rangle \oplus \beta_i^*$ . It then inserts dummy entries into the remaining entries of  $\mathbf{T}^*$ . So in other words,  $\mathcal{S}$  runs step 3 of the `BuildIndex` algorithm of SSE-1 with  $|\Delta'| = q$ , using  $\mathbf{A}^*$  instead of  $\mathbf{A}$ , and using  $\beta_i^*$  and  $a_i^*$  instead of  $f_y(w_i)$  and  $\pi_z(w_i)$ , respectively.

**Generating  $T_i^*$ :**  $\mathcal{S}$  sets  $T_i^* = (a_i^*, \beta_i^*)$

The correctness of searching on  $\mathcal{I}^*$  using trapdoors  $T_i^*$  follows trivially. We again claim that no probabilistic polynomial-time adversary  $\mathcal{A}$  can distinguish between  $V_q^*$  and  $V(H_q) = \{1, \dots, n, \mathcal{E}(D_1), \dots, \mathcal{E}(D_n), \mathcal{I}_{\mathcal{D}}, T_1, \dots, T_q\}$  for any  $H_q$ , otherwise, by a standard hybrid argument,  $\mathcal{A}$  could distinguish between at least one of the elements of  $V_q^*$  and its corresponding element in  $V(H_q)$ . We show that this is impossible since each element of  $V_q^*$  is indistinguishable from its corresponding element in  $V(H_q)$ :

**Encrypted documents:** for  $1 \leq i \leq n$ ,  $\mathcal{E}(D_i)$  is indistinguishable from a random string  $e_i^* \xleftarrow{R} \{0, 1\}^r$  since  $(\mathcal{G}, \mathcal{E}, \mathcal{D})$  is semantically secure.

**Index:** Recall that  $\mathcal{I}_{\mathcal{D}} = (\mathbf{A}, \mathbf{T})$  and that  $\mathcal{I}^* = (\mathbf{A}^*, \mathbf{T}^*)$ . Since  $\mathbf{A}$  consists of  $m'$  semantically secure ciphertexts and  $m - m'$  random strings of the same size, while  $\mathbf{A}^*$  consists of  $q$  semantically secure ciphertexts and  $m - q$  random strings of the same size, it follows that each element in  $\mathbf{A}^*$  will be indistinguishable from its counterpart in  $\mathbf{A}$ , otherwise one could either distinguish between two semantically secure ciphertexts of the same size, or between a semantically secure ciphertext and a random string of the same size.

Now recall that  $\mathbf{T}$  consists of  $|\Delta'|$  ciphertexts,  $c_i$ , generated by XOR-ing a message with the output of  $f$ , and of  $|\Delta - \Delta'|$  random values of size  $\ell + \log_2(m)$ .  $\mathbf{T}^*$ , on the other hand, consists of  $q$  ciphertexts generated by XOR-ing a message with a random string  $\beta_i^*$  of length  $\ell + \log_2(m)$ , and  $|\Delta| - q$  random strings of the same length. It follows that each element of  $\mathbf{T}$  and  $\mathbf{T}^*$  are indistinguishable, otherwise one could distinguish between the output of  $f$  and a random string.

**Trapdoors:** for  $1 \leq j \leq q$ ,  $T_j$  is indistinguishable from  $T_j^*$ , otherwise one could distinguish between the output of  $\pi$  and a random string of size  $p$  or between the output of  $f$  and a random string of size  $\ell + \log_2(m)$ . ■

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  $O(m)$  on the server; more specifically, in



addition to the encrypted  $\mathcal{D}$ , the server stores the index  $\mathcal{I}$ , which has size  $O(m)$ , and the look-up table  $\mathbf{T}$ , which has size  $O(|\Delta|)$ . Since the size of encrypted  $\mathcal{D}$  is  $O(m)$ , 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 word  $w$ , the server spends time proportional to  $|\mathcal{D}(w)|$ .

## 4.2 Adaptive SSE security

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, costs are the same—see Table 1.)

The difficulty of proving our SSE-1 construction secure against an adaptive adversary stems from the difficulty of creating in advance a view for the adversary that would be consistent with future (unknown) queries. Given the intricate structure of the SSE-1 construction, with each word having a corresponding linked list whose nodes are stored encrypted and in a random order, building an appropriate index is quite challenging. We circumvent this problem as follows.

For a given word  $w$  and a given integer  $j$ , we derive a label for  $w$  by concatenating  $w$  with  $j$  ( $j$  is first converted to a string of characters). For example, if  $w$  is “coin” and  $j$  is 1, then  $w||j$  is “coin1”. We define the *family* of a word  $w \in \Delta'$  to be the set of labels  $F_w = \{w||j : 1 \leq j \leq |\mathcal{D}(w)|\}$ . For example, if the word “coin” appears in three documents, then  $F_w = \{\text{“coin1”}, \text{“coin2”}, \text{“coin3”}\}$ . Note that the maximum size of a word’s family is  $n$ . Now, for each word  $w \in \Delta'$ , we choose not to keep a list of nodes with the identifiers in  $\mathcal{D}(w)$ , but instead to simply derive the family  $F_w$  of  $w$ , and insert the elements of  $F_w$  into the index. Searching for  $w$  becomes equivalent with searching for all the labels in  $w$ ’s family. Since each label in  $w$ ’s family will appear in only one document, a search for it “reveals” only one entry in the index. Translated to the proof, this will allow the simulator to easily construct a view for the adversary that is indistinguishable from a real view.

We now give an overview of the SSE-2 construction. We associate with the document collection  $\mathcal{D}$  an index  $\mathcal{I}$ , that consists of a look-up table  $\mathbf{T}$ . For each label in a word  $w$ ’s family, we add an entry in  $\mathbf{T}$ , whose `value` field is the identifier of the document that contains an instance of  $w$ . In order to hide the number of distinct words in each document, we make it look like all documents contain the same number of distinct words: We achieve this by “padding” the look-up table  $\mathbf{T}$  such that the identifier of each document appears in the same number of entries. The search for a word  $w$  is slightly different than for the SSE-1 construction: a user needs to search for all the labels in  $w$ ’s family (there can be at most  $n$  labels).

Recall that  $\Delta'$  is the set of distinct words that exist in  $\mathcal{D}$ . Let  $\mathbf{max}$  be the maximum number of distinct words that can be fit in the largest document in  $\mathcal{D}$  (assuming the minimum size for a word is one byte). We give next an algorithm to determine  $\mathbf{max}$ , given  $\mathbf{MAX}$ , the size (in bytes) of the largest document in  $\mathcal{D}$ . In step 1 we try to fit the maximum number of distinct 1-byte words; there are  $2^8$  such words, which gives a total size of 256 bytes ( $2^8 \cdot 1$  bytes). If  $\mathbf{MAX} > 256$ , then we continue to step 2. In step 2 we try to fit the maximum number of distinct 2-byte words; there are  $2^{16}$  such words, which gives a total size of 131328 bytes ( $2^8 \cdot 1 + 2^{16} \cdot 2$  bytes). In general, in step  $i$  we try to fit the maximum number of distinct  $i$ -byte words, which is  $2^{8 \cdot i}$ . We continue similarly until step  $i$  when  $\mathbf{MAX}$  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 words as possible in a document of size  $\mathbf{MAX}$ . For example, when the largest document in  $\mathcal{D}$  has size  $\mathbf{MAX} = 1$  MByte, we can fit at most  $\mathbf{max} = 355349$  distinct words ( $2^8$  distinct 1-byte words +  $2^{16}$  distinct 2-byte words + 289557 distinct 3-byte words). Note that  $\mathbf{max}$  cannot be larger than  $|\Delta|$ ; thus, if we get a value for  $\mathbf{max}$  (using the previously described algorithm) that is larger than  $|\Delta|$ , then we set  $\mathbf{max} = |\Delta|$ .

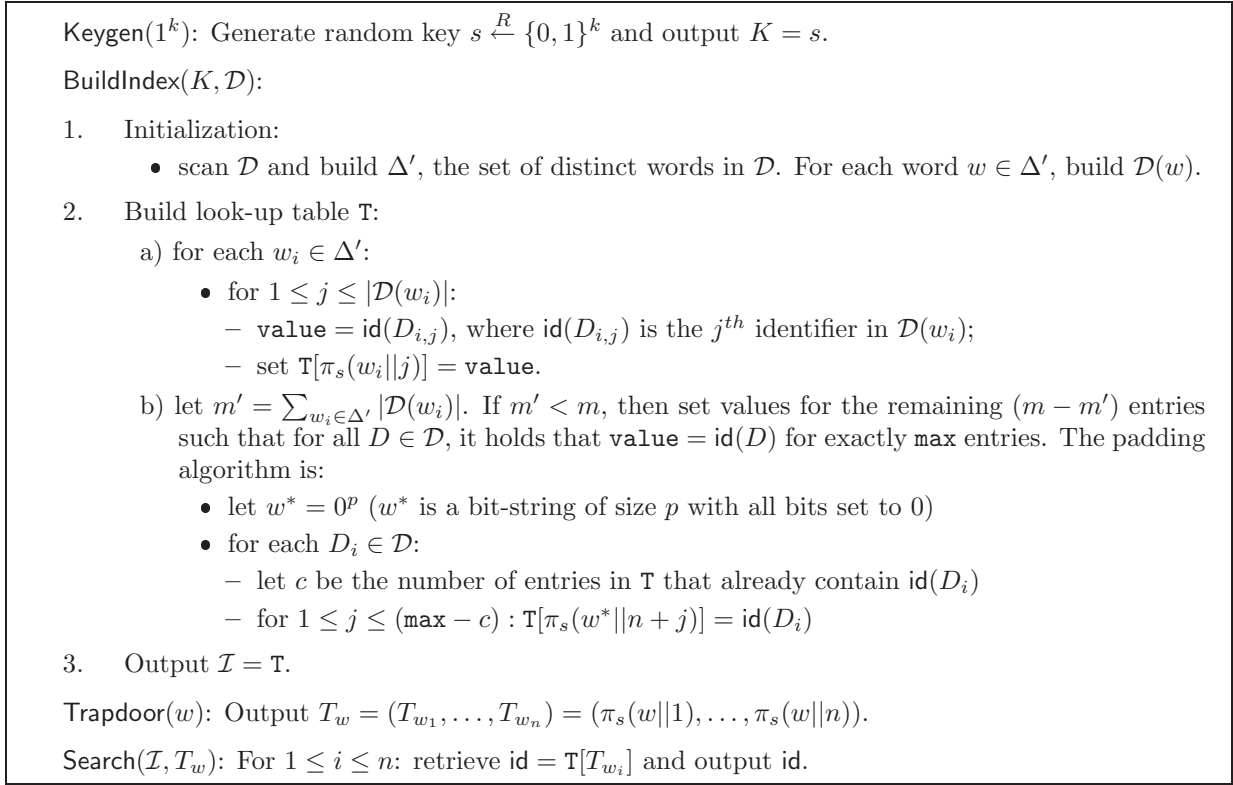


Figure 2: Adaptively secure SSE construction (SSE-2)

Let  $k$  be a security parameter and  $m = \mathbf{max} \cdot n$ , where  $n$  is the number of documents in  $\mathcal{D}$ . Recall that words in  $\Delta$  can be represented using at most  $p$  bits. We use a pseudo-random permutation  $\pi : \{0, 1\}^k \times \{0, 1\}^{p+\log_2(n+\mathbf{max})} \rightarrow \{0, 1\}^{p+\log_2(n+\mathbf{max})} \times \{0, 1\}^{\log_2(n)} \times m$  look-up table, managed using indirect addressing. The construction SSE-2 is described in Fig. 2.

**Theorem 4.2.** *SSE-2 is an adaptively secure SSE scheme.*

*Proof.* The idea behind the proof is to describe a simulator that can simulate the partial view of an adversary given only the trace of a partial history. Following the simulation-based approach of Definition 3.9, we describe a probabilistic polynomial-time simulator  $\mathcal{S}$  such that for all  $q \in \mathbb{N}$ , all probabilistic polynomial-time adversaries  $\mathcal{A}$ , all polynomially-bounded functions  $f$ , all distributions  $\mathcal{H}_q$ , given  $\text{Tr}(H_q^t)$ ,  $\mathcal{S}$  can simulate  $\mathcal{A}(V_K^t(H_q))$  for all  $0 \leq t \leq q$  with probability negligibly close to 1, where  $H_q \xleftarrow{R} \mathcal{H}_q$  and  $K \leftarrow \text{Keygen}(1^k)$ . More precisely, we show that, for all  $0 \leq t \leq q$ ,  $\mathcal{S}(\text{Tr}(H_q^t))$  can generate a view  $(V_q^t)^*$  such that  $(V_q^t)^*$  is indistinguishable from  $V_K^t(H_q)$ .

In the SSE-2 construction, for each word, the labels in the word's family are inserted into the index and each label appears in only one document. Searching for a word consists of searching for  $n$  labels in its family. Before proceeding further, we recall that the parameters of the pseudo-random permutation  $\pi$  used to instantiate the scheme are known to  $\mathcal{S}$ . Also, recall the notions of a view and of a trace (and their partial versions) in the context of the SSE-2 construction:

- $V_K(H_q) = (1, \dots, n, \mathcal{E}(D_1), \dots, \mathcal{E}(D_n), \mathcal{I}_{\mathcal{D}}, T_1, \dots, T_q)$
- $V_K^t(H_q) = (1, \dots, n, \mathcal{E}(D_1), \dots, \mathcal{E}(D_n), \mathcal{I}_{\mathcal{D}}, T_1, \dots, T_t)$ , where  $0 \leq t \leq q$
- $\text{Tr}(H_q) = (1, \dots, n, |D_1|, \dots, |D_n|, \mathcal{D}(w_1), \dots, \mathcal{D}(w_q), \Pi_q)$ , where  $\mathcal{D}(w_i) = (\mathcal{D}(w_i||1), \dots, \mathcal{D}(w_i||n))$ , for  $0 \leq i \leq q$

- $Tr(H_q^t) = (1, \dots, n, |D_1|, \dots, |D_n|, \mathcal{D}(w_1), \dots, \mathcal{D}(w_t), \Pi_t)$ , where  $\mathcal{D}(w_i) = (\mathcal{D}(w_i||1), \dots, \mathcal{D}(w_i||n))$ , for  $0 \leq i \leq t$  and  $0 \leq t \leq q$

For a given  $q$ , the simulator must commit to an index before any queries are made, i.e. at time  $t = 0$  the simulator generates an index  $\mathcal{I}^*$  that will be included in all partial views  $(V_q^t)^*$  used to simulate  $\mathcal{A}$ , for all  $0 \leq t \leq q$ . Note that although at time  $t = 0$  the simulator has no knowledge about future queries, the index  $\mathcal{I}^*$  must be indistinguishable from a real index  $\mathcal{I}_{\mathcal{D}}$  in  $V_K^t(H_q)$ , for all  $0 \leq t \leq q$ . Also, for all  $0 \leq t \leq q$ , the simulator includes in the partial view  $(V_q^t)^*$  the document identifiers (which it knows from the trace of the partial history  $Tr(H_q^0)$ ) and ciphertexts obtained by generating random values (of size known from  $Tr(H_q^0)$ ). It follows trivially that the identifiers and ciphertexts in  $(V_q^t)^*$  are indistinguishable from those in  $V_K^t(H_q)$ , for all  $0 \leq t \leq q$ . It then remains to show how  $\mathcal{S}$  constructs the other elements in the view, namely the index and the trapdoors. Note that, from having access to the trace, the simulator knows the size of the largest document in  $\mathcal{D}$ , and can thus compute  $\mathbf{max}$  and  $m = \mathbf{max} \cdot n$ .

For  $t = 0$ , the simulator's trace on the partial history  $Tr(H_q^0)$  contains among other things, the identifiers of each document in the collection.  $\mathcal{S}$  constructs (and includes in  $(V_q^0)^*$ ) the index  $\mathcal{I}^*$  as a  $(\{0, 1\}^{p+\log_2(n+\mathbf{max})} \times \{0, 1\}^{\log_2(n)} \times m)$  look-up table  $\mathbf{T}^*$ , where  $m = \mathbf{max} \cdot n$  and  $\mathbf{T}^*$  contains  $\mathbf{max}$  copies of each document's identifier inserted at random locations.  $\mathcal{S}$  keeps a copy of  $\mathcal{I}^*$  in order to be able to simulate future partial views for  $1 \leq t \leq q$ . Given the algorithm used to construct a real index  $\mathcal{I}$  included in the partial view  $V_K^0(H_q)$ , it is clear that  $\mathcal{I}^*$  is indistinguishable from  $\mathcal{I}$ , otherwise one could distinguish between the output of  $\pi$  and a random string of size  $p + \log_2(n + \mathbf{max})$ . Thus,  $(V_q^0)^*$  is indistinguishable from  $V_K^0(H_q)$ .

For  $1 \leq t \leq q$ , the simulator includes in the partial view  $(V_q^t)^*$  the index  $\mathcal{I}^*$  which was computed for  $t = 0$  and which was established above to be indistinguishable from a real index  $\mathcal{I}$  in a partial view  $V_K^t(H_q)$ . Recall that  $\mathcal{I}^*$  consists of a look-up table  $\mathbf{T}^*$  and that  $Tr(H_q^t)$  contains the search pattern matrix  $\Pi_t$  for the  $t$  queries in  $Tr(H_q^t)$ . We describe how  $\mathcal{S}$  constructs the trapdoors  $(T_1^*, \dots, T_t^*)$  included in  $(V_q^t)^*$ .  $\mathcal{S}$  reuses the trapdoors  $(T_1^*, \dots, T_{t-1}^*)$  that were included in  $(V_q^{t-1})^*$  (We assume that  $\mathcal{S}$  remembers  $(V_q^{t-1})^*$  and can reuse the trapdoors in it; alternatively,  $\mathcal{S}$  can reconstruct these trapdoors from  $Tr(H_q^{t-1})$ , one by one in the same manner we will show how to construct  $T_t^*$  from  $\mathcal{D}(w_t)$  and  $\Pi_t$ ).

To construct  $T_t^*$ ,  $\mathcal{S}$  first checks if  $H_q^{t-1}$  contains  $w_t$  (by checking if  $\Pi_t[t, j] = 1$  for any  $1 \leq j \leq t-1$ ). If this check is negative, then  $\mathcal{S}$  uses the knowledge from  $Tr(H_q^t)$  about  $\mathcal{D}(w_t)$ , namely that  $\mathcal{D}(w_t) = (\mathcal{D}(w_t||1), \dots, \mathcal{D}(w_t||n))$ . Note that each  $\mathcal{D}(w_t||i)$ , for  $1 \leq i \leq n$ , contains only one document identifier, to which we refer as  $\text{id}(D_{t,i})$ . For  $1 \leq i \leq n$ ,  $\mathcal{S}$  randomly picks an address  $\mathbf{addr}_i$  from  $\mathbf{T}^*$  such that  $\mathbf{T}^*[\mathbf{addr}_i] = \text{id}(D_{t,i})$ , making sure that all  $\mathbf{addr}_i$  are pairwise distinct, and constructs trapdoor  $T_t^* = (\mathbf{addr}_1, \dots, \mathbf{addr}_n)$ . Also,  $\mathcal{S}$  remembers the association between  $T_t^*$  and  $w_t$ . Otherwise, if  $H_q^{t-1}$  contains  $w_t$ , then  $\mathcal{S}$  retrieves the trapdoor associated with  $w_t$  and assigns it to  $T_t^*$ . This ensures that if  $H_q^t$  contains repeated words, then the corresponding trapdoors included in  $(V_q^t)^*$  are identical.

It's easy to see that the trapdoors  $(T_1^*, \dots, T_t^*)$  in  $(V_q^t)^*$  are indistinguishable from the trapdoors  $(T_1, \dots, T_t)$  in  $V_K^t(H_q)$ , otherwise one could distinguish between the output of  $\pi$  and a random string of size  $p + \log_2(n + \mathbf{max})$ . Thus,  $(V_q^t)^*$  is indistinguishable from  $V_K^t(H_q)$ , for all  $0 \leq t \leq q$ . ■

Just like SSE-1, SSE-2 requires for each query one round of communication and an amount of computation on the server proportional with the number of documents that match the query (i.e.,  $O(|\mathcal{D}(w)|)$ ). Similarly, the storage and computational demands on the user are  $O(1)$ . The communication is equal to  $n$  and the storage on the server is increased by a factor of  $\mathbf{max}$  when compared to

SSE-1. We note that the communication cost can be reduced if in each entry of  $T$  corresponding to an element in some word  $w$ 's family, we also store  $|\mathcal{D}(w)|$  in encrypted form. In this way, after searching for a label in  $w$ 's family, the user will know  $|\mathcal{D}(w)|$  and can derive  $F_w$ . The user can then send in a single round all the trapdoors corresponding to the remaining labels in  $w$ 's family.

### 4.3 Secure updates

We allow for secure updates to the document collection in the sense defined by Chang and Mitzenmacher [CM05]: each time the user adds a new set  $\zeta$  of encrypted documents,  $\zeta$  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 [CM05], the user sends a pseudo-random seed for each document collection, which implies that queries have size  $O(u)$ . We propose a solution that achieves better bounds for the size of queries (namely queries have size  $O(\log u)$ ) 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.

When the user performs an update, i.e. submits a set  $\zeta^a$  of new documents, the server checks if there exists (from previous updates) a document collection  $\zeta^b$ , such that  $|\zeta^b| \leq |\zeta^a|$ . If so, then the server sends  $\zeta^b$  back to the user, and the user combines  $\zeta^a$  and  $\zeta^b$  into a single collection  $\zeta^c$  with  $|\zeta^a| + |\zeta^b|$  documents for which it re-computes the index. The server stores the combined document collection  $\zeta^c$  and its index  $\mathcal{I}_c$ , and deletes the composing document collections  $\zeta^a, \zeta^b$  and their indexes  $\mathcal{I}_a, \mathcal{I}_b$ . Note that  $\zeta^c$  and its index  $\mathcal{I}_c$  will not reveal anything more than what was already revealed by the  $\zeta^a, \zeta^b$  and their indexes  $\mathcal{I}_a, \mathcal{I}_b$ , 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 the Hamming weight of  $f(u)$ . 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 [CM05]).

## 5 Multi-User Searchable Encryption

In this section we consider the natural extension to the SSE setting where a user owns a document collection, but an arbitrary group of users can submit queries to search his collection. A familiar question arises in this new setting, that of managing access privileges, but 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 [FN94] (BE) scheme. Let  $\mathbf{N}$  denote the set of all possible users, and  $\mathbf{G} \subseteq \mathbf{N}$  the set of users that are currently authorized to search.

**Definition 5.1** (Multi-User Searchable Symmetric Encryption Scheme). *A multi-user SSE scheme is a collection of six polynomial-time algorithms  $\mathbf{M}\text{-SSE} = (\mathbf{M}\text{Keygen}, \mathbf{M}\text{BuildIndex}, \text{AddUser}, \text{RevokeUser}, \mathbf{M}\text{Trapdoor}, \mathbf{M}\text{Search})$  such that:*

$\mathbf{M}\text{Keygen}(1^k)$  is a probabilistic key generation algorithm that is run by the owner  $O$  to setup the scheme.

It takes a security parameter  $k$ , and returns an owner secret key,  $K_O$ .

$\mathbf{M}\text{BuildIndex}(K_O, \mathcal{D})$  is run by  $O$  to construct indexes. It takes the owner's secret key  $K_O$  and a document collection  $\mathcal{D}$  as inputs, and returns an index  $\mathcal{I}$ .

$\text{AddUser}(K_O, U)$  is run by  $O$  whenever it wishes to add a user to the group  $G$ . It takes the owner's secret key  $K_O$  and a user  $U$  as inputs, and returns  $U$ 's secret key,  $K_U$ .

$\text{RevokeUser}(K_O, U)$  is run by  $O$  whenever it wishes to revoke a user from  $G$ . It takes the owner's secret key  $K_O$  and a user  $U$  as inputs, and revokes the user's searching privileges.

$\text{MTrapdoor}(K_U, w)$  is run by a user (including  $O$ ) in order to generate a trapdoor for a given word. It takes a user  $U$ 's secret key  $K_U$  and a word  $w$  as inputs, and returns a trapdoor  $T_{U,w}$ .

$\text{MSearch}(\mathcal{I}_D, T_{U,w})$  is run by the server  $\mathcal{S}$  in order to search for the documents in  $\mathcal{D}$  that contain word  $w$ . It takes the index  $\mathcal{I}_D$  for collection  $\mathcal{D}$  and the trapdoor  $T_{U,w}$  for word  $w$  as inputs, and returns  $\mathcal{D}(w)$  if user  $U \in G$  and  $\perp$  if user  $U \notin G$ .

We briefly discuss here notions of security that a multi-user SSE scheme should achieve. It should be clear that the (semantic) security of a multi-user scheme can be reduced to the semantic security of the underlying single-user scheme. The reason is that in the multi-user case, just like in the single-user case, we are only concerned with providing security against the server. One distinct property in this new setting is that of *revocation*, which essentially states that a revoked user no longer be able to perform searches on the owner's documents.

**Definition 5.2** (Revocation). *A multi-user SSE scheme  $\text{M-SSE} = (\text{MKeygen}, \text{MBuildIndex}, \text{AddUser}, \text{RevokeUser}, \text{MTrapdoor}, \text{MSearch})$  achieves revocation if for all probabilistic polynomial-time adversaries  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ , all polynomials  $p$  and sufficiently large  $k$ :*

$$\Pr \left[ \begin{array}{l} T' = \text{MTrapdoor}(K_U, w) : K_O \leftarrow \text{MKeygen}(1^k); K_A \leftarrow \text{AddUser}(K_O, \mathcal{A}); \\ \text{state} \leftarrow \mathcal{A}_1^{\text{MBuildIndex}(K_O, \cdot)}(K_A); \text{RevokeUser}(K_O, \mathcal{A}); \\ (w, T') \leftarrow \mathcal{A}_2^{\text{MTrapdoor}(K_O, \cdot)}(\text{state}); \end{array} \right] < \frac{1}{p(k)}$$

where the probability is taken over  $\text{MKeygen}$ ,  $\mathcal{A}$  and  $\text{MBuildIndex}$ 's internal coins.

In this definition, the adversary runs in two stages,  $\mathcal{A}_1$  and  $\mathcal{A}_2$ , representing its status as an authorized user and then as a revoked user. We use *state* to represent the knowledge gained by  $\mathcal{A}_1$ , which is then being passed to  $\mathcal{A}_2$ . The adversary  $\mathcal{A}$  wins the game if it is able to generate a valid trapdoor  $T_{\mathcal{A},w}$  for some word  $w$ , after it has been revoked.

Our construction makes use of a single-user SSE scheme and a broadcast encryption (BE) scheme. Recall that in BE, a center encrypts a message  $m$  to a group  $G$  of privileged users who are allowed to access the message. The group  $G$  can be dynamically changing, as users can be added to or removed from  $G$ . Although the encrypted message can be received by a larger set  $N$  of receivers, only the users in  $G$  can recover the message. When a user joins the system, it receives a set of secrets, referred to as *long-lived* secrets. The long-lived secrets are distinct for each user. Given an encrypted message, the long-lived secrets allow a user to decrypt it only if the user was non-revoked at the time the message was encrypted. We use off-the-shelf BE as a building block in our multi-user secure index construction in order to efficiently manage user revocation.

We point out that users receive their long-lived secrets for the BE scheme only when they are given search authorization. As such, a user  $U$  that has not joined the system yet can run the  $\text{MTrapdoor}$  algorithm and retrieve  $\mathcal{E}_{N \setminus R}^{\text{BE}}(r)$  from the server. However, since  $U$  does not know its long-lived secrets, it will not be able to recover  $r$ . Similarly, when a revoked user  $U$  retrieves  $\mathcal{E}_{N \setminus R}^{\text{BE}}(r)$  from the server, it cannot recover  $r$  because  $U \in R$ . Moreover, even though a revoked user which has been re-authorized to search could recover (old) values of  $r$  used while being revoked, these values are no longer of interest. The fact that backward secrecy is not needed for the BE scheme makes the  $\text{AddUser}$  algorithm more efficient, since it does not require the owner to send a message to the server.

We now provide an overview of the construction. In order to retrieve the documents that contain the word  $w$ , an authorized user  $u$  computes a regular single-user trapdoor  $T_{U,w}$ , but applies on it a

<p><b>MKeygen</b>(<math>1^k, 1^\ell</math>): let <math>K^s \leftarrow \text{Keygen}(1^k, 1^\ell)</math> and <math>r \xleftarrow{R} \{0, 1\}^k</math>. Output <math>K_{\mathcal{O}} = (K^s, r)</math>.</p> <p><b>MBuildIndex</b>(<math>K_{\mathcal{O}}, \mathcal{D}</math>): run <math>\mathcal{I}^s \leftarrow \text{BuildIndex}(K^s, \mathcal{D})</math>. Initialize the BE scheme. Set <math>R = \{\emptyset\}</math>. Send <math>r</math> and <math>\mathcal{E}_{\mathbf{N}}^{\text{BE}}(r)</math> to the server. Output <math>\mathcal{I}^m = \mathcal{I}^s</math>.</p> <p><b>AddUser</b>(<math>K_{\mathcal{O}}, U</math>): send <math>K_U = (K^s, r)</math> to user <math>U</math>, where <math>r</math> is the current key used for <math>\phi</math>. Also send to <math>U</math> the long-lived secrets needed for the BE scheme.</p> <p><b>RevokeUser</b>(<math>K_{\mathcal{O}}, U</math>): <math>R = R \cup \{U\}</math>. Pick a new key <math>r' \xleftarrow{R} \{0, 1\}^k</math> and send <math>r'</math> and <math>\mathcal{E}_{\mathbf{N} \setminus R}^{\text{BE}}(r')</math> to <math>\mathcal{S}</math>. <math>\mathcal{S}</math> overwrites the old values of <math>r</math> and <math>\mathcal{E}_{\mathbf{N} \setminus R \cup \{U\}}^{\text{BE}}(r)</math> with <math>r'</math> and <math>\mathcal{E}_{\mathbf{N} \setminus R}^{\text{BE}}(r')</math>, respectively.</p> <p><b>MTrapdoor</b>(<math>K_U, w</math>): let <math>T_w^s \leftarrow \text{Trapdoor}(K^s, w)</math>. Retrieve <math>\mathcal{E}_{\mathbf{N} \setminus R}^{\text{BE}}(r)</math> from <math>\mathcal{S}</math> and use the long-lived BE secrets to recover <math>r</math>. Output <math>T_{U,w}^m = \phi_r(T_w^s)</math>.</p> <p><b>MSearch</b>(<math>\mathcal{I}^m, T_{U,w}^m</math>): recover <math>T_w^s = \phi_r^{-1}(T_{U,w}^m)</math>. Run <math>\text{Search}(\mathcal{I}^m, T_w^s)</math> and return its output.</p>
---

Figure 3: Multi-user SSE construction (M-SSE)

pseudo-random permutation  $\phi$  keyed with a secret key  $r$  before sending it to the server. The server, upon receiving  $\phi_r(T_{U,w})$ , recovers the trapdoor by computing  $T_{U,w} = \phi_r^{-1}(\phi_r(T_{U,w}))$ . The key  $r$  currently used for  $\phi$  is only known by the owner, by the set of currently authorized users and by the server. Each time a user is revoked, the owner picks a new  $r$  and stores it on the server encrypted such that only non-revoked users can decrypt it. Broadcast encryption provides an efficient method to distribute  $r$  to the set of non-revoked users. The server will use the new  $r$  to compute  $\phi_r^{-1}$  for subsequent queries. Revoked users cannot recover the current  $r$  and, with overwhelming probability, their queries will not yield a valid trapdoor after the server applies  $\phi_r^{-1}$ .

When the owner  $O$  of a document collection  $\mathcal{D}$  gives a user  $U$  permission to search through  $\mathcal{D}$ , it sends to  $U$  all the secret information needed to perform searches in a single-user context<sup>6</sup>. The extra layer given by the pseudo-random permutation  $\phi$ , together with the guarantees offered by the BE scheme and the assumption that the server is honest-but-curious, is what prevents users from performing successful searches once they are revoked.

Next we describe the multi-user SSE construction in detail. Let  $\text{SSE} = (\text{Keygen}, \text{BuildIndex}, \text{Trapdoor}, \text{Search})$  be a single-user SSE scheme and  $\text{BE} = (\mathcal{G}^{\text{BE}}, \mathcal{E}_{\mathbf{G}}^{\text{BE}}, \mathcal{D}_{\mathbf{G}}^{\text{BE}})$  be a broadcast encryption scheme. We require a standard security notion for the BE scheme, namely that it provide revocation-scheme security against a coalition of all revoked users and that its key assignment algorithm satisfies key indistinguishability. Recall that we let  $\mathbf{N}$  denote the set of all users, and  $\mathbf{G} \subseteq \mathbf{N}$  the set of users (currently) authorized to search; let  $\mathbf{R}$  denote the set of revoked users. Let  $\phi$  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. For example, if the single user SSE scheme is instantiated with our SSE-1 construction, then  $\phi$  will be defined as  $\phi : \{0, 1\}^k \times \{0, 1\}^{p+\log_2(m)+\ell} \rightarrow \{0, 1\}^{p+\log_2(m)+\ell}$ .  $\phi$  can be constructed using the techniques proposed by Black and Rogaway [BR02], which describe how to build pseudo-random permutations over domains of arbitrary size. Our multi-user construction M-SSE = (MKeygen, MBuildIndex, AddUser, RevokeUser, MTrapdoor, MSearch) is described in Fig. 3.

Our multi-user construction is very efficient on the server side: 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 “heavier” authentication protocol would be required.

<sup>6</sup>Note that  $O$  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  $\mathcal{D}$ . This guarantees that only  $O$  can perform updates to  $\mathcal{D}$ .

## Acknowledgements

The authors thank Fabian Monrose for helpful discussions at early stages of this work. The first author was at Bell Labs during part of this work. The third author is 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

- [ABC<sup>+</sup>02] A. Adya, W. Bolosky, M. Castro, R. Chaiken, G. Cermak, J. Douceur, J. Howell, J. Lorch, M. Theimer, and R. Wattenhofer. Farsite: Federated, available, and reliable storage for an incompletely trusted environment. In *Proceedings of the 5th SOSDI*, December 2002.
- [ABC<sup>+</sup>05] 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 *CRYPTO 2005*, volume 3621 of *LNCS*, pages 205–222. Springer, 2005.
- [BBO06] M. Bellare, A. Boldyreva, and A. O’Neill. Efficiently-searchable and deterministic asymmetric encryption. Cryptology ePrint archive, June 2006. report 2006/186, <http://eprint.iacr.org/2006/186>.
- [BC04] S.M. Bellovin and W.R. Cheswick. Privacy-enhanced searches using encrypted Bloom filters. Technical Report 2004/022, IACR ePrint Cryptography Archive, 2004.
- [BdCOP04] D. Boneh, G. di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Proc. EUROCRYPT 04*, pages 506–522, 2004.
- [BEG<sup>+</sup>91] 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*, pages 90–99, 1991.
- [BG02] B. Barak and O. Goldreich. Universal arguments and their applications. In *IEEE Conference on Computational Complexity*, pages 194–203, 2002.
- [BKM05] L. Ballard, S. Kamara, and F. Monrose. Achieving efficient conjunctive keyword searches over encrypted data. In *Proceedings of the Seventh International Conference on Information and Communication Security (ICICS 2005)*, pages 414–426, 2005.
- [BKOS07] D. Boneh, E. Kushilevitz, R. Ostrovsky, and W. Skeith. Public-key encryption that allows PIR queries. In *Proc. of CRYPTO ’07*, August 2007.
- [Blo70] B. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.
- [BOGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for fault-tolerant distributed computing. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 1–10, Chicago, 1988. ACM.
- [BR02] J. Black and P. Rogaway. Ciphers with arbitrary finite domains. In *Proc. of CT-RSA*, volume 2271 of *LNCS*, pages 114–130. Springer-Verlag, 2002.
- [CM05] Y. C. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *Applied Cryptography and Network Security Conference (ACNS)*, 2005.
- [FKS84] M.L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with  $O(1)$  worst case access time. *J. ACM*, 31(3):538–544, 1984.
- [FN94] A. Fiat and M. Naor. Broadcast encryption. In Douglas R. Stinson, editor, *Proc. CRYPTO 93*, volume 773 of *Lecture Notes in Computer Science*, pages 480–491. Springer-Verlag, 1994.
- [GD] Google Desktop. <http://desktop.google.com>.
- [GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *JCSS*, 28(2):270–299, April 1984.

- [GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to prove all NP statements in Zero-Knowledge and a methodology of cryptographic protocol design. In A. M. Odlyzko, editor, *Proc. CRYPTO 86*, volume 263 of *LNCS*, pages 171–185. Springer-Verlag, 1987.
- [GO96] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM*, 43(3):431–473, 1996.
- [Goh03] E-J. Goh. Secure indexes. Technical Report 2003/216, IACR ePrint Cryptography Archive, 2003. See <http://eprint.iacr.org/2003/216>.
- [Gol93] Oded Goldreich. A uniform complexity treatment of encryption and zero-knowledge. *Journal of Cryptology*, 6(1):21–53, 1993.
- [GSW04] 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)*, volume 3089 of *LNCS*, pages 31–45. Springer-Verlag, 2004.
- [IKOS04] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Batch codes and their applications. In *36th Annual ACM Symposium on Theory of Computing (STOC '04)*, pages 262–271. ACM, 2004.
- [IKOS06] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Cryptography from anonymity. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS '06)*. IEEE, 2006.
- [ISA02] Privacy with Security. DARPA Information Science and Technology (ISAT) Study Group (<http://www.cs.berkeley.edu/~tygar/papers/ISAT-final-briefing.pdf>), December 2002.
- [KBC<sup>+</sup>00] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS '00*. ACM, November 2000.
- [KO97] Eyal Kushilevitz and Rafail Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *IEEE Symposium on Foundations of Computer Science*, pages 364–373, 1997.
- [KO04] J. Katz and R. Ostrovsky. Round-optimal secure two-party computation. In *Advances in Cryptology - CRYPTO 2004*, volume 3152 of *LNCS*, pages 335–354. Springer, 2004.
- [MMGC02] A. Muthitacharoen, R. Morris, T.M. Gil, and B. Chen. Ivy: A read/write peer-to-peer file system. In *Proceedings of the 5th SOSDI*, December 2002.
- [OS05] R. Ostrovsky and W. Skeith. Private searching on streaming data. In *Advances in Cryptology - CRYPTO '05*, volume 3621 of *Lecture Notes in Computer Science*, pages 223–240. Springer, 2005.
- [Ost90] R. Ostrovsky. Software protection and simulations on oblivious RAMs. In *Proceedings of 22nd Annual ACM Symposium on Theory of Computing*, 1990. MIT Ph.D. Thesis, 1992.
- [PKL04] D.J. Park, K. Kim, and P.J. Lee. Public key encryption with conjunctive field keyword search. In *5th International Workshop WISA 2004*, volume 3325 of *LNCS*, pages 73–86. Springer, 2004.
- [SWP00] D. Song, D. Wagner, and A. Perrig. Practical techniques for searching on encrypted data. In *Proceedings of 2000 IEEE Symposium on Security and Privacy*, pages 44–55, May 2000.
- [Yao82] A. C. Yao. Protocols for secure computations. In *Proc. 23rd IEEE Symp. on Foundations of Comp. Science*, pages 160–164, Chicago, 1982. IEEE.

## A Basic Primitives

**Definition A.1** (Semantic Security against Adaptive Chosen-Ciphertext Attacks). *We say that a symmetric encryption scheme is semantically secure against adaptive chosen-ciphertext attacks (IND-CCA2) if for all probabilistic polynomial-time adversaries  $\mathcal{A}$ , all polynomials  $p$  and sufficiently large  $k$ :*

$$\Pr \left[ \begin{array}{l} b' = b : K \leftarrow \mathcal{G}(1^k); (m_0, m_1) \leftarrow \mathcal{A}^{\mathcal{E}_K(\cdot), \mathcal{D}_K(\cdot)}; \\ b \xleftarrow{R} \{0, 1\}; c \leftarrow \mathcal{E}_K(m_b); b' \leftarrow \mathcal{A}^{\mathcal{E}_K(\cdot), \mathcal{D}_K(\cdot)}(c) \end{array} \right] < \frac{1}{p(k)}$$



with the restriction that  $|m_0| = |m_1|$ , that  $\mathcal{A}$  cannot query its decryption oracle with  $c$ , and where the probability is taken over  $\mathcal{G}$ ,  $\mathcal{E}$  and  $\mathcal{A}$ 's internal coins.

**Definition A.2** (Pseudo-random Functions and Permutations). *Let  $\mathcal{F}$  be the set of all functions from  $\{0, 1\}^n$  to  $\{0, 1\}^m$ . A function  $f : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^m$  is pseudo-random if it is computable in polynomial time and if for all probabilistic polynomial-time adversaries  $\mathcal{A}$ , all polynomials  $p$  and sufficiently large  $k$ :*

$$\left| \Pr \left[ \mathcal{A}^{f_{K(\cdot)}} = 1 : K \xleftarrow{R} \{0, 1\}^k \right] - \Pr \left[ \mathcal{A}^{g(\cdot)} = 1 : g \xleftarrow{R} \mathcal{F} \right] \right| < \frac{1}{p(k)}$$

where the probabilities are taken over the internal coins of  $\mathcal{A}$  and the choice of  $K$  and  $g$ . If  $f$  is bijective then it is a pseudo-random permutation.

## B Limitations of Previous SSE Definitions

To date, two definitions of security have been used for SSE schemes: indistinguishability against chosen-keyword attacks (IND2-CKA), introduced by Goh [Goh03], and a simulation-based definition introduced by Chang and Mitzenmacher [CM05]. We point out their shortcomings in turn.

**Indistinguishability-based SSE definitions.** Intuitively, the notion of security that IND2-CKA tries to achieve can be described as follows: given access to a set of indexes, the adversary (i.e., the server) is not able to learn any partial information about the underlying document that he cannot learn from using a trapdoor that was given to him by the client, and this holds even against adversaries that can trick the client into generating indexes and trapdoors for documents and keywords of its choice (i.e., chosen-keyword attacks).

**Definition B.1** (IND2-CKA [Goh03]). *A SSE scheme (Keygen, BuildIndex, Trapdoor, Search) is IND2-CKA if for all probabilistic polynomial-time adversaries  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ , all polynomials  $p$  and sufficiently large  $k$ :*

$$\Pr \left[ \begin{array}{l} b' = b : K \leftarrow \text{Keygen}(1^k); (D_0, D_1, \text{state}) \leftarrow \mathcal{A}_1^{\text{BuildIndex}(K, \cdot), \text{Trapdoor}(K, \cdot)}; \\ b \xleftarrow{R} \{0, 1\}; \mathcal{I}_b \leftarrow \text{BuildIndex}(K, \mathcal{D}_b); \\ b' \leftarrow \mathcal{A}_2^{\text{BuildIndex}(K, \cdot), \text{Trapdoor}(K, \cdot)}(\mathcal{I}_b, D_0, D_1, \text{state}) \end{array} \right] < \frac{1}{2} + \frac{1}{p(k)}$$

with the restriction that  $\mathcal{A}_1$  must choose  $(D_0, D_1)$  such that  $\text{Search}(\mathcal{I}_0, T) = \text{Search}(\mathcal{I}_1, T)$  for all  $T$  generated using its Trapdoor oracle and the restriction that  $\mathcal{A}_2$  cannot query its Trapdoor oracle on any word in  $(D_0 \cup D_1) \setminus (D_0 \cap D_1)$ . Also,  $\text{state}$  is a polynomially bounded string that captures  $\mathcal{A}_1$ 's state, and the probability is taken over Keygen,  $\mathcal{A}$  and BuildIndex's internal coins.

We remark that Goh's work addresses a larger problem than searchable encryption, namely that of secure indexes, which are secure data structures that have many uses, only one of which is searchable encryption. And as Goh remarks (*cf.* Note 1, p. 5 of [Goh03]), IND2-CKA does not explicitly require that trapdoors be secure since this is not a requirement for all applications of secure indexes. Therefore, IND2-CKA is clearly not an appropriate security definition for searchable encryption.

In addition, we point out that although one might be tempted to remedy the situation by introducing a second definition to guarantee the semantic security of trapdoors, this cannot be done in a straightforward manner. Indeed, first proving that an SSE scheme is IND2-CKA and then proving that its trapdoors are semantically secure *does not* imply that an adversary cannot recover the word being queried. In fact, as we now show, SSE schemes can be built with trapdoors that, taken independently, leak no partial information about the word being queried, but when combined with an index

allow an adversary to recover the entire word. This illustrates that the security of indexes and the security of trapdoors are inherently linked.

We now give an SSE construction, SSE-0, that generates indexes that can be proven IND2-CKA, and trapdoors that can be proven semantically secure (in the sense that they leak no partial information about the underlying word), but that when taken together enable an adversary to recover the entire word. Before we describe the construction, we note that 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 3.1, so that we could use the original formulations of IND2-CKA (or IND-CKA). In particular, this means that to both build and search indexes, one must run the `BuildIndex` and the `Search` algorithms on each document in a collection  $\mathcal{D} = (D_1, \dots, D_n)$ .

Let  $\Delta = (w_1, \dots, w_d)$  be a dictionary of  $d$  words; we assume, without loss of generality, that each word is encoded as a bit string of length  $\lambda$ . The construction also uses two pseudo-random permutations  $\pi$  and  $\psi$  with the following parameters:  $\pi : \{0, 1\}^k \times \{0, 1\}^{\lambda+\ell} \rightarrow \{0, 1\}^{\lambda+\ell}$  and  $\psi : \{0, 1\}^k \times \{0, 1\}^\lambda \rightarrow \{0, 1\}^\lambda$ . (For simplicity, we let  $\pi(s, x) \stackrel{\text{def}}{=} \pi_s(x)$ ; similarly for  $\psi$ .) Let SSE-0 = (Keygen, BuildIndex, Trapdoor, Test) be a SSE scheme defined as follows:

**Keygen**( $1^k$ ): Generate two random keys  $s, z \stackrel{R}{\leftarrow} \{0, 1\}^k$  and set  $K = (s, z)$ .

**BuildIndex**( $K, \Delta, D_i$ ):

1. Instantiate an array  $\mathbf{A}_i$  of  $|\Delta|$  elements;<sup>7</sup>
2. for each word  $w_j \in D_i$ :
  - (a) run `Trapdoor`( $K, w_j$ ) and keep  $(\pi_s(w_j||i), \psi_z(w_j))$ ;
  - (b) store  $\pi_s(w_j||i) \oplus (w_j||0^\ell)$  in  $\mathbf{A}_i[\psi_z(w_j)]$ ;
3. output  $\mathcal{I}_i = \mathbf{A}_i$ .

**Trapdoor**( $K, w$ ): Output  $T_w = (\pi_s(w||1), \dots, \pi_s(w||n), \psi_z(w))$

**Search**( $\mathcal{I}_i, T_w$ ): Decrypt element by computing  $\mathbf{A}_i[\psi_z(w)] \oplus \pi_s(w||i)$ . If the last  $\ell$  bits of the result are equal to 0, then output `true`.

**Theorem B.2.** *If  $\pi_s$  is a pseudo-random permutation, then SSE-0 is IND2-CKA.*

*Proof.* We show that given an adversary  $\mathcal{A}$  that breaks SSE-0 in the IND2-CKA sense, we can construct an adversary  $\mathcal{B}$  that can distinguish whether a permutation  $\pi$  is random or pseudo-random.

Let  $\psi : \{0, 1\}^k \times \{0, 1\}^\lambda \rightarrow \{0, 1\}^\lambda$  be a pseudo-random permutation.  $\mathcal{B}$  is given oracle access to  $\pi$  and begins by generating a random value  $z \stackrel{R}{\leftarrow} \{0, 1\}^k$ . It then starts simulating  $\mathcal{A}_1$ , and answers its oracle queries as follows: for each `Trapdoor` query  $w$ ,  $\mathcal{B}$  answers with  $T_w = (\pi(w||1), \dots, \pi(w||n), \psi_z(w))$  by querying its own oracle and by using the pseudo-random permutation  $\psi_z$ ; for each `BuildIndex` query  $D$ ,  $\mathcal{B}$  runs the `BuildIndex` algorithm using its oracle and  $\psi_z$  (as the two underlying random permutations), and returns  $\mathcal{I}_D$ . After polynomially many queries,  $\mathcal{A}_1$  outputs two documents  $(D_0, D_1)$  such that `Search`( $D_0, T$ ) = `Search`( $D_1, T$ ) for all  $T$  returned by its oracle queries.  $\mathcal{B}$  then chooses a bit  $b \stackrel{R}{\leftarrow} \{0, 1\}$  and constructs an index  $I_b$  from document  $D_b$ , again by running `BuildIndex` and using its oracle  $\pi$  and  $\psi_z$  as the underlying random permutations.  $\mathcal{B}$  then returns  $I_b$  to  $\mathcal{A}_2$  and answers  $\mathcal{A}_2$ 's `Trapdoor` and `BuildIndex` queries as it did before and under the same restrictions. Finally, after polynomially many queries,  $\mathcal{A}_2$  outputs a bit  $b'$  which represents its guess as to which document

<sup>7</sup>We assume that  $\mathbf{A}_i$  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 Sect. 4.1 for an efficient way to achieve this.

was used to generate  $I_b$ . If  $b' = b$ , then  $\mathcal{B}$  answers its own challenge with 0, indicating that  $\pi$  is a pseudo-random permutation; otherwise it outputs 1, indicating that  $\pi$  is a random permutation.

It follows from  $\mathcal{B}$ 's description that  $\mathcal{B}$  runs in polynomial time, so it remains to show that its advantage in distinguishing  $\pi$  is non-negligible. From Definition A.2 and the fact that  $\mathcal{B}$  outputs 0 if and only if  $\mathcal{A}$  outputs  $b' = b$ , we know that:

$$\left| \Pr \left[ \mathcal{B}^{\pi(\cdot)} = 0 \right] - \Pr \left[ \mathcal{B}^{\pi_s(\cdot)} = 0 \right] \right| = \left| \Pr \left[ b' = b \mid \pi \xleftarrow{R} \Pi \right] - \Pr \left[ b' = b \mid s \xleftarrow{R} \{0, 1\}^k \right] \right|$$

We claim that

$$\Pr \left[ b' = b \mid \pi \xleftarrow{R} \Pi \right] = \frac{1}{2} \tag{4}$$

and that

$$\Pr \left[ b' = b \mid s \xleftarrow{R} \{0, 1\}^k \right] \geq \frac{1}{2} + \frac{1}{p(k)} \tag{5}$$

where  $p$  is some arbitrary polynomial.

Since the indexes include the unique ID's of the underlying document, if  $\pi$  is a pseudo-random permutation, then two indexes will be independent of each other to any probabilistic polynomial-time adversary, even if they are built from the same documents. This implies that  $\mathcal{A}$  cannot learn anything about its challenge index  $I_b$  from its BuildIndex queries. We can therefore restrict our analysis to  $\mathcal{A}$ 's distinguishing probability when only given its challenge.

We begin by establishing Equation (4). Since  $\pi$  is a random permutation,  $\mathcal{A}$  will output  $b' = b$  with probability exactly  $\frac{1}{2}$ . To see why, consider two documents  $D_0 = (w_0)$  and  $D_1 = (w_1)$  with one word each and suppose, for contradiction, that  $\mathcal{A}$  outputs  $b' = b$  with probability larger than  $\frac{1}{2}$ . This means that when given  $I_b = \pi(w_b || b) \oplus (w_b || 0^\ell)$ ,  $\mathcal{A}$  can determine whether  $w_b = w_0$  or  $w_b = w_1$  with probability larger than  $\frac{1}{2}$ . This is a contradiction, since it would imply that  $\mathcal{A}$  can “break” a one-time pad.

We now establish Equation (5). If  $\pi$  is instantiated with a pseudo-random permutation, then  $\mathcal{B}$  simulates the IND2-CKA game perfectly and, by assumption,  $\mathcal{A}$  outputs  $b' = b$  with non-negligible probability over  $\frac{1}{2}$ . ■

Notice that SSE-0's trapdoors do *not* leak any partial information about the underlying word since the trapdoors are generated using pseudo-random permutations. However, any adversary with access to a trapdoor  $T_w = (\pi_s(w || 1), \dots, \pi_s(w || n), \psi_z(w))$  and a set of indexes  $(\mathbf{A}_1, \dots, \mathbf{A}_n)$  can recover  $w$  by simply computing  $\mathbf{A}_i[\psi_z(w)] \oplus \pi_s(w || i)$  for all  $1 \leq i \leq n$ .

**Simulation-based SSE definitions.** In [CM05], Chang and Mitzenmacher provide a definition of security for SSE that is intended to be stronger than IND2-CKA in the sense that it requires a scheme to output 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** ([CM05]). *For all  $q \in \mathbb{N}$ , for all probabilistic polynomial-time adversaries  $\mathcal{A}$ , all sets  $\{\mathcal{D}, w_1, \dots, w_{q-1}\}$ , all functions  $f$ , there exists a probabilistic polynomial-time algorithm (simulator)  $\mathcal{S}$  such that for all polynomials  $p$  and sufficiently large  $k$ :*

$$\left| \frac{\Pr [\mathcal{A}(C_q) = f(\{\mathcal{D}, w_1, \dots, w_q\})]}{\Pr [\mathcal{S}(\{\mathcal{E}(\mathcal{D}), \mathcal{D}(w_1), \dots, \mathcal{D}(w_q)\}) = f(\{\mathcal{D}, w_1, \dots, w_q\})]} \right| < \frac{1}{p(k)}$$

where  $C_q$  is the entire communication the server receives up to the  $q^{\text{th}}$  query<sup>8</sup> to and  $k$  is the security parameter used for the index and trapdoor generation.

Note that the order of the quantifiers in the definition imply that the algorithm  $\mathcal{S}$  can depend on a set  $\{\mathcal{D}, w_1, \dots, w_{q-1}\}$ . This means that for any  $q$  and any set  $\{\mathcal{D}, w_1, \dots, w_{q-1}\}$ , there will *always* exist a simulator that can compute some function  $f$  of  $\{\mathcal{D}, w_1, \dots, w_{q-1}\}$ , given only  $\{\mathcal{E}(\mathcal{D}), \mathcal{D}(w_1), \dots, \mathcal{D}(w_{q-1})\}$ . This can be 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  $\{\mathcal{D}, w_1, \dots, w_{q-1}\}$  the inequality in Definition B.3 holds; or, as we do in our own definition (see Definition 3.6), by requiring that the inequality hold over all distributions over the set of all possible sets of the form  $\{\mathcal{D}, w_1, \dots, w_{q-1}\}$ .

As mentioned in Section 1, Definition B.3 is inherently non-adaptive. Again, 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  $\mathcal{D}(w_1)$ , at time  $t = 2$  performs a search for word  $w_2$  and receives the set of documents  $\mathcal{D}(w_2)$ , and so on until  $q$  searches are performed (i.e., until  $t = q$ ). Our intuition about secure searchable encryption clearly tells us that at time  $t = 0$ , the adversary (i.e., the server) should not be able to recover any partial information about the documents from the index. Similarly, at time  $t = 1$ , the adversary should not be able to recover any partial information about the documents and  $w_1$  from the index and the trapdoor for  $w_1$ . 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.

Returning to Definition B.3, notice that for a given  $q \in \mathbb{N}$ , the simulator is required to simulate  $\mathcal{A}(C_q)$  (where  $C_q$  is the entire communication between the user and the server over  $q$  queries) 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. However, we cannot conclude that the index can be simulated at time  $t = 0$  given only the encrypted documents; or whether the 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 [CM05], where for each  $q \in \mathbb{N}$ , the simulator is allowed to generate a different index (when  $q \geq 0$ ) and different trapdoors (when  $q \geq 1$ ).

---

<sup>8</sup>While the original definition found in [CM05] defines  $C_q$  to be the entire communication the server receives *before* the  $q^{\text{th}}$  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.