Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions *

Reza Curtmola
Department of Computer Science
Johns Hopkins University
Baltimore, MD
crix@cs.jhu.edu

Juan Garay
Bell Labs – Lucent Technologies
Murray Hill, NJ
garay@research.bell-labs.com

Seny Kamara
Department of Computer Science
Johns Hopkins University
Baltimore, MD
seny@cs.jhu.edu

Rafail Ostrovsky
Department of Computer Science and
Department of Mathematics
UCLA
Los Angeles, CA
rafail@cs.ucla.edu

ABSTRACT

Searchable symmetric encryption (SSE) allows a party to outsource the storage of its data to another party (a server) in a private manner, while maintaining the ability to selectively search over it. This problem has been the focus of active research in recent years. In this paper we show two solutions to SSE that simultaneously enjoy the following properties:

1. Both solutions are more efficient than all previous constant-round schemes. In particular, the work performed by the server per returned document is *constant* as opposed to linear in the size of the data.
2. Both solutions enjoy stronger security guarantees than previous constant-round schemes. In fact, we point out subtle but serious problems with previous notions of security for SSE, and show how to design constructions which avoid these pitfalls. Further, our second solution also achieves what we call *adaptive* SSE security, where queries to the server can be chosen adaptively (by the adversary) during the execution of the search; this notion is both important in practice and has not been previously considered.

Surprisingly, despite being more secure and more efficient, our SSE schemes are remarkably simple. We consider the simplicity of both solutions as an important step towards the deployment of SSE technologies.

As an additional contribution, we also consider *multi-user* SSE. All prior work on SSE studied 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 the multi-user setting, and present an efficient construction that achieves better performance than simply using access control mechanisms.

Categories and Subject Descriptors

E.3 [**Data Encryption**]; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval.

General Terms

Algorithms, Security, Theory.

Keywords

Searchable symmetric encryption, multi-user searchable encryption, security definitions.

1. INTRODUCTION

Private-key storage outsourcing 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 [27, 16, 7, 10, 12]; 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 [2]. In addition, it can allow services such as Google Desktop [1] 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 software protection based on *oblivious RAMs* [24, 18]. While oblivious RAMs hide all information about the RAM use

*An extended version of the paper is available as IACR ePrint report 2006/210 [13].

CCS'06, October 30–November 3, 2006, Alexandria, Virginia, USA.

from a remote and potentially malicious server with a poly-logarithmic overhead in all parameters (including computation and communication), 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 achieves 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 reveal the user's access and search patterns (precise definitions below) while "hiding everything else," and show that the 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 [24, 18].

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 [10, 3]. Anyone with access to the public key can add words to the index, but only the owner of the private key can generate "trapdoors" to test for the occurrence of a keyword. Although the original work on public-key encryption with keyword search (PEKS) by Boneh, di Crescenzo, Ostrosvky and Persiano [10] reveals the user's access pattern, recently Boneh, Kushilevitz, Ostvrosky and Skeith [11] have shown how to build a public-key encryption scheme that hides even the access pattern. This construction, however, has an overhead in search time that is proportional to the square root of the database size, which is far less efficient then the best private-key solutions.

Recently, Bellare, Boldyreva and O'Neill [6] 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 [23], they show how a user can retrieve data from a server containing *unencrypted* data without revealing the access pattern and with total communication less then the data size. This was extended to keyword searching, including searching on streaming data [25]. 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 [21, 22], 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 software protection and oblivious RAMs by Goldreich and Ostrovsky [18]. In an effort to reduce the round complexity associated with oblivious RAMs, Song, Wagner and Perrig [27] 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 [16] and of Chang and Mitzenmacher [12], 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, for "chosen-keyword attack," and the slightly stronger

IND2-CKA), and puts forth a construction based on Bloom filters [8] 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 do not leak any information about the words being queried. We discuss these security definitions and their shortcomings in more detail in Section 3.

As mentioned above, encryption with keyword search has also been considered in the public-key setting [10, 3], 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 [6] 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.

Our results. We now summarize our contributions.

1. We review existing security definitions for searchable encryption, including IND2-CKA [16] and the simulation-based definition in [12], 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 of 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 [27, 16, 12], 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 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 [16] also performs searches in one round, it can induce false positives, which is not the case for our construction. Additionally, all the constructions in [16, 12] 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 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 [12]. We point out an optimization which lowers the communication size and the server's computation per query from linear to logarithmic in the number of updates (see full version [13]).

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 [9] and universal arguments [5] 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 postpone this extension to the full version.

Due to space limitations, full-fledged security definitions, security proofs and extensions are presented in the full version of the paper [13].

2. PRELIMINARIES

Let $\Delta = \{w_1, \ldots, w_d\}$ be a dictionary of d words, and 2^{Δ} be the set of all possible documents. Further, let $\mathcal{D} \subseteq 2^{\Delta}$ be a collection of n documents $\mathcal{D} = (D_1, \ldots, D_n)$ and $2^{2^{\Delta}}$ be the set of all possible document collections. Let $\mathsf{id}(D)$ be the identifier of document D, where the identifier can be any string that uniquely identifies a document, such as

Properties	[24, 18]	[24, 18]-light	[27]	[16]	[12]	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.

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 $(\mathcal{D}(w_1), \ldots, \mathcal{D}(w_n))$ 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 $x \leftarrow \mathcal{X}$ to represent an element x being sampled from a distribution $\mathcal{X}$ and $x \xleftarrow{R} X$ 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 $||$ 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 $\mathcal{D} = (D_1, \ldots, D_n)$ 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 N. O owns the document collection $\mathcal{D}$ and wants to grant and revoke searching privileges to a subset of users in N. We let $\mathsf{G} \subseteq \mathsf{N}$ be the set of users allowed to search. We assume that currently non-revoked users behave honestly. The honest-but-curious server S is a party that follows the protocol specification correctly, but may try to analyze the messages received during the protocol in order to learn additional information.

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.

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.

3. REVISITING SSE DEFINITIONS

We begin by reviewing the definition of a SSE scheme.

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

$\mathsf{Keygen}(1^k)$ *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.*

$\mathsf{BuildIndex}(K, \mathcal{D})$ *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.*

$\mathsf{Trapdoor}(K, w)$ *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.*

$\mathsf{Search}(\mathcal{I}, T_w)$ *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. 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 as a formal security definition. In the literature, 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), however we are not aware of any previous work on SSE that satisfies this intuition. In fact, with the exception of oblivious RAMs, all previous constructions leak, in addition to the search outcomes, the user's search pattern. This is clearly the case for the schemes presented in [27, 16, 12] 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, 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.

Limitations of previous SSE definitions. The second issue seems to be in appropriately capturing the adversary's power. While Song, Wagner and Perrig proved 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 who proposed the notion of indistinguishability against chosen-keyword attacks (IND2-CKA) in [16][1]. 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 [13] [2].

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 though much work on searchable encryption uses IND2-CKA as a security definition [20, 26, 4], we note that it was never intended as such. This is simply because, as Goh remarks (*cf.* Note 1, p. 5 of [16]), IND2-CKA does not explicitly require that trapdoors be secure since this is not a requirement for all applications of secure indexes.

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 reasonable (though cumbersome) idea, the straightforward approach of requiring trapdoors to be indistinguishable does not work. In fact, as we show in [13], 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.

Chang and Mitzenmacher propose a simulation-based definition that aims to guarantee privacy for indexes *and* trapdoors [12]. Similarly to the classical definition of semantic security for encryption [19], 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 show in [13], the definition presented in [12] is not only *non-adaptive*, but can be trivially satisfied by any SSE scheme, even one that is insecure.

Our security definitions. We now address the above issues. For ease of readability, in this section we present our approach at a somewhat informal level, but a more rigorous treatment can be found in the full version of the paper [13].

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*. 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). 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).

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 3.2. (Non-Adaptive Indistinguishability Security for SSE—informal version) *A SSE scheme is secure in the sense of* non-adaptive indistinguishability *if for any two adversarially constructed histories with equal length and trace, no (probabilistic polynomial-time) adversary can distinguish the view of one from the view of the other with probability non-negligibly better than* $\frac{1}{2}$.

Second, for each history the adversary generates, we give him the ability to choose the word queries as a function of the index and the trapdoors corresponding to the document collection and the previous queries it chose. More precisely,

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

[2]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.

for each history, the adversary must choose a document collection and multiple query words. So in this version of our definition, after he chooses a document collection, he will receive its corresponding index *before* he chooses his first query word. And he will then receive that query word's trapdoor *before* he chooses his next query word, and so on. What this implies is that for the two histories he constructs, he can choose query words as a function of the index and his previous query words' trapdoors. Intuitively, this could enable the adversary to perform more sophisticated attacks than in the previous case. We call such histories "(adversarially) adaptively constructed" (a formal specification of this process is described in [13]).

DEFINITION 3.3. (ADAPTIVE INDISTINGUISHABILITY SECURITY FOR SSE—INFORMAL VERSION) *A SSE scheme is secure in the sense of* adaptive indistinguishability *if for any two adaptively-constructed histories with equal length and trace, no (probabilistic polynomial-time) adversary can distinguish the view of one history from the view of the other with probability non-negligibly better than* $\frac{1}{2}$.

An alternative approach to security definitions is the so-called semantic security or "simulation-based" approach [19, 17]. At a high level, in such an approach the security guarantee is provided by the existence, for all adversaries, of a polynomial-time algorithm (the *simulator*) which, being given very little information (in our case, a history's trace), is able to compute whatever the adversary is able to compute from the given information (in our case, the history's view). In [13], we also present simulation-based definitions for SSE (overcoming the shortcomings of the simulation-based definition in [12]), both for the non-adaptive and adaptive settings, and, moreover, we are able to prove:

THEOREM 3.4. *Non-adaptive (respectively, adaptive) indistinguishability security of SSE is equivalent to non-adaptive (resp., adaptive) semantic security of SSE.*

We remark that the existence of such an equivalence proof typically vouches for the soundness of the definitions presented herein. Further, it allows us to state that an SSE scheme is simply non-adaptively (resp., adaptively) secure, without any reference to the proof methodology.

4. EFFICIENT AND SECURE SSE

In this section we present our efficient SSE constructions, and state their security in terms of the definitions presented in Section 3 (the security proofs are presented in the full version [13]). We start by introducing some additional notation and the data structures used by the constructions. Let Δ', $\Delta' \subseteq \Delta$, be the set of distinct words that exist in the document collection $\mathcal{D}$. We assume that words in Δ 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 $\mathtt{A}$, we refer to the element at address i in $\mathtt{A}$ as $\mathtt{A}[i]$, and to the address of element x relative to $\mathtt{A}$ as $\mathsf{addr}(\mathtt{A}(x))$. So if $\mathtt{A}[i] = x$, then $\mathsf{addr}(\mathtt{A}(x)) = i$. In addition, a linked list $\mathtt{L}$, stored in an array $\mathtt{A}$, is a set of nodes $\mathtt{N}_i = \langle v_i; \mathsf{addr}(\mathtt{A}(\mathtt{N}_{i+1}))\rangle$, where $1 \leq i \leq |\mathtt{L}|$, v_i is an arbitrary string and $\mathsf{addr}(\mathtt{A}(\mathtt{N}_{i+1}))$ is the memory address of the next node in the list.

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

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

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

The user computes both $\mathtt{A}$ and $\mathtt{T}$ 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 $\mathtt{T}$ and sends them to the server. The server locates and decrypts the given entry of $\mathtt{T}$, and gets the index in $\mathtt{A}$ and the decryption key for the first node of $\mathtt{L}_i$. Since each element of $\mathtt{L}_i$ contains information about the next element of $\mathtt{L}_i$, the server can locate and decrypt all the nodes of $\mathtt{L}_i$, 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 $\mathtt{T}$ are tuples <address,value>, in which the `address` field is used is used as a *virtual address* to locate the entry in $\mathtt{T}$ 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 $\mathsf{poly}(p)$. If, for a lookup-up table $\mathtt{T}$, the `address` field is from $\{0,1\}^p$, the `value` field is from $\{0,1\}^v$ and there are at most m entries in $\mathtt{T}$, then we say $\mathtt{T}$ is a $(\{0,1\}^p \times \{0,1\}^v \times m)$ look-up table.

Let Addr be the set of virtual addresses that are used for entries in a look-up table $\mathtt{T}$. We can efficiently store $\mathtt{T}$ such that, when given a virtual address, it returns the associated `value` field. We achieve this by organizing the Addr set in a so-called *FKS dictionary* [15], an efficient data structure for

$\mathsf{Keygen}(1^k, 1^\ell)$: **Generate random keys** $s, y, z \xleftarrow{R} \{0,1\}^k$ **and output** $K = (s, y, z, 1^\ell)$.

$\mathsf{BuildIndex}(K, \mathcal{D})$:

1. **Initialization:**
   a) **scan** $\mathcal{D}$ **and build** Δ'**, the set of distinct words in** $\mathcal{D}$**. For each word** $w \in \Delta'$**, build** $\mathcal{D}(w)$**;**
   b) **initialize a global counter** $\mathsf{ctr} = 1$.
2. **Build array** $\mathtt{A}$**:**
   a) **for each** $w_i \in \Delta'$**: // (build a linked list** $\mathtt{L}_i$ **with nodes** $\mathtt{N}_{i,j}$ **and store it in array** $\mathtt{A}$**)**
   - **generate** $\kappa_{i,0} \xleftarrow{R} \{0,1\}^\ell$
   - **for** $1 \le j \le |\mathcal{D}(w_i)|$**:**
     - **generate** $\kappa_{i,j} \xleftarrow{R} \{0,1\}^\ell$ **and set node** $\mathtt{N}_{i,j} = \langle \mathsf{id}(D_{i,j}) \| \kappa_{i,j} \| \psi_s(\mathsf{ctr}+1) \rangle$**, where** $\mathsf{id}(D_{i,j})$ **is the** j^{th} **identifier in** $\mathcal{D}(w_i)$**;**
     - **compute** $\mathcal{E}_{\kappa_{i,j-1}}(\mathtt{N}_{i,j})$**, and store it in** $\mathtt{A}[\psi_s(\mathsf{ctr})]$**;**
     - $\mathsf{ctr} = \mathsf{ctr} + 1$
   - **for the last node of** $\mathtt{L}_i$ **(i.e.,** $\mathtt{N}_{i,|\mathcal{D}(w_i)|}$**), before encryption, set the address of the next node to** NULL**;**

   b) **let** $m' = \sum_{w_i \in \Delta'} |\mathcal{D}(w_i)|$**. If** $m' < m$**, then set remaining** $(m - m')$ **entries of** $\mathtt{A}$ **to random values of the same size as the existing** m' **entries of** $\mathtt{A}$**.**
3. **Build look-up table** $\mathtt{T}$**:**
   a) **for each** $w_i \in \Delta'$**:**
   - $\mathtt{value} = \langle \mathsf{addr}(\mathtt{A}(\mathtt{N}_{i,1})) || \kappa_{i,0} \rangle \oplus f_y(w_i)$**;**
   - **set** $\mathtt{T}[\pi_z(w_i)] = \mathtt{value}$.

   b) **if** $|\Delta'| < |\Delta|$**, then set the remaining** $(|\Delta| - |\Delta'|)$ **entries of** $\mathtt{T}$ **to random values.**
4. **Output** $\mathcal{I} = (\mathtt{A}, \mathtt{T})$.

$\mathsf{Trapdoor}(w)$: **Output** $T_w = (\pi_z(w), f_y(w))$.

$\mathsf{Search}(\mathcal{I}, T_w)$:

1. **Let** $(\gamma, \eta) = T_w$**. Retrieve** $\theta = \mathtt{T}[\gamma]$**. Let** $\langle \alpha || \kappa \rangle = \theta \oplus \eta$.
2. **Decrypt the list** $\mathtt{L}$ **starting with the node at address** α **encrypted under key** κ.
3. **Output the list of document identifiers contained in** $\mathtt{L}$.

Figure 1: Efficient SSE construction (SSE-1)

storage of sparse tables that requires $O(|\mathsf{Addr}|)$ $(+o(|\mathsf{Addr}|))$ storage and $O(1)$ look-up time. In other words, given some virtual address A, we are able to tell if A $\in$ 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, ℓ be security parameters and let $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ be a semantically secure symmetric encryption scheme with $\mathcal{E} : \{0,1\}^\ell \times \{0,1\}^r \to \{0,1\}^r$. In addition, we make use of one pseudo-random function f and two pseudo-random permutations π and ψ with the following parameters:

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

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).[3] Let $\mathtt{A}$ be an array of size m. Let $\mathtt{T}$ 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 $\mathtt{A}$ can be seen as a collection of linked lists $\mathtt{L}_i$, where each $\mathtt{L}_i$ contains the identifiers of documents containing word w_i. Let $m' = \sum_{w_i \in \Delta'} |\mathtt{L}_i|$. 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 $\mathtt{A}$ is smaller than m, then the array $\mathtt{A}$ reveals that at least one document in $\mathcal{D}$ contains a word more than once. To avoid such leakage, we set the size of $\mathtt{A}$ 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 $\mathtt{T}$, 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 $\mathtt{T}$, filled with random values, such that the number of entries in $\mathtt{T}$ is always equal to $|\Delta|$.

In the full version of the paper [13], we show:

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

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

[3] 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 $\mathcal{D}$.

$\mathsf{Keygen}(1^k)$: **Generate random key** $s \xleftarrow{R} \{0,1\}^k$ **and output** $K = s$.

$\mathsf{BuildIndex}(K, \mathcal{D})$:

1. **Initialization:**
   - **scan** $\mathcal{D}$ **and build** Δ'**, the set of distinct words in** $\mathcal{D}$**. For each word** $w \in \Delta'$**, build** $\mathcal{D}(w)$.
2. **Build look-up table** $\mathtt{T}$**:**
   a) **for each** $w_i \in \Delta'$**:**
      - **for** $1 \le j \le |\mathcal{D}(w_i)|$**:**
        - $\mathtt{value} = \mathsf{id}(D_{i,j})$**, where** $\mathsf{id}(D_{i,j})$ **is the** j^{th} **identifier in** $\mathcal{D}(w_i)$**;**
        - **set** $\mathtt{T}[\pi_s(w_i||j)] = \mathtt{value}$.

   b) **let** $m' = \sum_{w_i \in \Delta'} |\mathcal{D}(w_i)|$**. If** $m' < m$**, then set values for the remaining** $(m - m')$ **entries such that for all** $D \in \mathcal{D}$**, it holds that** $\mathtt{value} = \mathsf{id}(D)$ **for exactly** $\mathtt{max}$ **entries. Also, set the** $\mathtt{address}$ **field of these remaining entries to random values.**
3. **Output** $\mathcal{I} = \mathtt{T}$.

$\mathsf{Trapdoor}(w)$: **Output** $T_w = (T_{w_1}, \ldots, T_{w_{\mathtt{max}}}) = (\pi_s(w||1), \ldots, \pi_s(w||\mathtt{max}))$.

$\mathsf{Search}(\mathcal{I}, T_w)$: **For** $1 \le i \le \mathtt{max}$**: retrieve** $\mathsf{id} = \mathtt{T}[T_{w_i}]$ **and output** id.

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

the index $\mathcal{I}$, which contains the array $\mathtt{A}$ of size $O(m)$ and the look-up table $\mathtt{T}$ of 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 was 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 $\mathsf{F}_w = \{w||j : 1 \le j \le |\mathcal{D}(w)|\}$. For example, if the word "coin" appears in three documents, then $\mathsf{F}_w = \{$"coin1", "coin2", "coin3"$\}$. 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 $\mathtt{T}$. For each label in a word w's family, we add an entry in $\mathtt{T}$, whose $\mathtt{value}$ 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 have to "pad" the look-up table $\mathtt{T}$ 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.

Let k be a security parameter. We use a pseudo-random permutation $\pi : \{0,1\}^k \times \{0,1\}^p \to \{0,1\}^p$. Recall that a *unit* is the smallest possible size for a word (e.g. one byte). Also, recall that Δ' is the set of distinct words that exist in $\mathcal{D}$. Let $\mathtt{max}$ be the size of the largest plaintext document in $\mathcal{D}$, expressed in units. Let $m = \mathtt{max} \cdot n$, where n is the number of documents in $\mathcal{D}$. Let $\mathtt{T}$ be a $(\{0,1\}^p \times \{0,1\}^{\log_2(n)} \times m)$ look-up table, managed using indirect addressing. The construction SSE-2 is described in Fig. 2. In [13] we prove:

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

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 $\mathtt{max}$ and the storage on the server is increased by a factor of $\mathtt{max}$ when compared to SSE-1. We note that the communication cost can be reduced if in each entry of $\mathtt{T}$ 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.

5. MULTI-USER SSE

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,

$\mathsf{MKeygen}(1^k, 1^\ell)$: **let** $K^s \leftarrow \mathsf{Keygen}(1^k, 1^\ell)$ **and** $r \xleftarrow{R} \{0,1\}^k$. **Output** $K_{\mathcal{O}} = (K^s, r)$.

$\mathsf{MBuildIndex}(K_{\mathcal{O}}, \mathcal{D})$: **run** $\mathcal{I}^s \leftarrow \mathsf{BuildIndex}(K^s, \mathcal{D})$. **Initialize the** BE **scheme. Set** $\mathsf{R} = \{\varnothing\}$. **Send** r **and** $\mathcal{E}^{\mathsf{BE}}_{\mathsf{N}}(r)$ **to the server. Output** $\mathcal{I}^m = \mathcal{I}^s$.

$\mathsf{AddUser}(K_{\mathcal{O}}, U)$: **send** $K_U = (K^s, r)$ **to user** U, **where** r **is the current key used for** ϕ. **Also send to** U **the long-lived secrets needed for the** BE **scheme.**

$\mathsf{RevokeUser}(K_{\mathcal{O}}, U)$: $\mathsf{R} = \mathsf{R} \cup \{U\}$. **Pick a new key** $r' \xleftarrow{R} \{0,1\}^k$ **and send** r' **and** $\mathcal{E}^{\mathsf{BE}}_{\mathsf{N}\setminus\mathsf{R}}(r')$ **to** $\mathcal{S}$. $\mathcal{S}$ **overwrites the old values of** r **and** $\mathcal{E}^{\mathsf{BE}}_{\mathsf{N}\setminus\mathsf{R}\cup\{U\}}(r)$ **with** r' **and** $\mathcal{E}^{\mathsf{BE}}_{\mathsf{N}\setminus\mathsf{R}}(r')$, **respectively.**

$\mathsf{MTrapdoor}(K_U, w)$: **let** $T^s_w \leftarrow \mathsf{Trapdoor}(K^s, w)$. **Retrieve** $\mathcal{E}^{\mathsf{BE}}_{\mathsf{N}\setminus\mathsf{R}}(r)$ **from** $\mathcal{S}$ **and use the long-lived BE secrets to recover** r. **Output** $T^m_{U,w} = \phi_r(T^s_w)$.

$\mathsf{MSearch}(\mathcal{I}^m, T^m_{U,w})$: **recover** $T^s_w = \phi_r^{-1}(T^m_{U,w})$; **let** $T^s_w = (\gamma, \eta)$. **If** γ **is a valid virtual address, then run** $\mathsf{Search}(\mathcal{I}^m, T^s_w)$ **and return its output. Otherwise, return** $\perp$.

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

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 (BE) scheme [14]. Let N denote the set of all possible users, and $\mathsf{G} \subseteq \mathsf{N}$ 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* $\mathsf{M\text{-}SSE} = (\mathsf{MKeygen}, \mathsf{MBuildIndex}, \mathsf{AddUser}, \mathsf{RevokeUser}, \mathsf{MTrapdoor}, \mathsf{MSearch})$ *such that:*

- $\mathsf{MKeygen}(1^k)$ *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.
- $\mathsf{MBuildIndex}(K_O, \mathcal{D})$ *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* $\mathcal{I}$.
- $\mathsf{AddUser}(K_O, U)$ *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.
- $\mathsf{RevokeUser}(K_O, U)$ *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.*
- $\mathsf{MTrapdoor}(K_U, w)$ *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* $T_{U,w}$.
- $\mathsf{MSearch}(\mathcal{I}_{\mathcal{D}}, T_{U,w})$ *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* $\mathcal{I}_{\mathcal{D}}$ *for collection* $\mathcal{D}$ *and the trapdoor* $T_{U,w}$ *for word* w *as inputs, and returns* $\mathcal{D}(w)$ *if user* $U \in \mathsf{G}$ *and* $\perp$ *if user* $U \notin \mathsf{G}$.

We briefly discuss 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. A formal specification of this property is given in the full version of the paper [13].

Definition 5.2 (Correctness). *Let* $\mathcal{D}$ *be a document collection and* $\mathcal{I}_{\mathcal{D}}$ *be its corresponding index. We say that a multi-user SSE scheme,* $\mathsf{M\text{-}SSE} = (\mathsf{MKeygen}, \mathsf{MBuildIndex}, \mathsf{AddUser}, \mathsf{RevokeUser}, \mathsf{MTrapdoor}, \mathsf{MSearch})$, *is* correct *if*

$$\Pr\left[\,\mathsf{MSearch}(\mathcal{I}_{\mathcal{D}}, T_{U,w}) = \mathcal{D}(w) : U \in \mathsf{G}\,\right] = 1.$$

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 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 pseudo-random permutation ϕ 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 ϕ 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 ϕ_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 ϕ_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 [4]. The extra layer given by the pseudo-random permutation ϕ, 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 $\mathsf{SSE} = (\mathsf{Keygen}, \mathsf{BuildIndex}, \mathsf{Trapdoor}, \mathsf{Search})$ be a single-user SSE scheme and $\mathsf{BE} = (\mathcal{G}^{\mathsf{BE}}, \mathcal{E}^{\mathsf{BE}}_{\mathsf{G}}, \mathcal{D}^{\mathsf{BE}}_{\mathsf{G}})$ be a broadcast encryption scheme. Though our MSSE construction is general and can be instantiated with any single-user SSE scheme, for ease of exposition, we describe it using our SSE-1 construction. 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 N denote the set of all users, and $\mathsf{G} \subseteq \mathsf{N}$ the set of users (currently) authorized to search; let R denote the set of revoked users. Let ϕ be a pseudo-random permutation such that $\phi : \{0,1\}^k \times \{0,1\}^{p+\log_2(m)+\ell} \rightarrow \{0,1\}^{p+\log_2(m)+\ell}$. Our multi-user construction, M-SSE, 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. Refer to [13] for further details.

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.

[4]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 $\mathcal{D}$.

6. REFERENCES

[1] Google Desktop. `http://desktop.google.com`.

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

[3] 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.

[4] 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.

[5] B. Barak and O. Goldreich. Universal arguments and their applications. In *IEEE Conference on Computational Complexity*, pages 194–203, 2002.

[6] 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`.

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

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

[9] 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.

[10] D. Boneh, G. di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Proc. EUROCRYPT 04*, pages 506–522, 2004.

[11] D. Boneh, E. Kushilevitz, R. Ostrovsky, and W. Skeith. Public-key encryption that allows PIR queries. Unpublished Manuscript, August 2006.

[12] Y. C. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *Applied Cryptography and Network Security Conference*, 2005.

[13] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. Cryptology ePrint archive, June 2006. report 2006/210, `http://eprint.iacr.org/2006/210`.

[14] A. Fiat and M. Naor. Broadcast encryption. In D. R. Stinson, editor, *Proc. CRYPTO 93*, volume 773 of *Lecture Notes in Computer Science*, pages 480–491. Springer-Verlag, 1994.

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

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

[17] O. Goldreich. *Foundations of Cryptography*. Cambridge University Press, 2001.

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

[19] S. Goldwasser and S. Micali. Probabilistic encryption. *JCSS*, 28(2):270–299, Apr. 1984.

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.

[25] 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.

[26] D. Park, K. Kim, and P. Lee. Public key encryption with conjunctive field keyword search. In *5th International Workshop WISA 2004*, volume 3325 of *LNCS*, pages 73–86. Springer, 2004.

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