The Locality of Searchable Symmetric Encryption

David Cash[1] and Stefano Tessaro[2]

[1] Department of Computer Science, Rutgers University, Piscataway, NJ 08855, USA
david.cash@cs.rutgers.edu
[2] Department of Computer Science, University of California, Santa Barbara
tessaro@cs.ucsb.edu

Abstract. This paper proves a lower bound on the trade-off between server storage size and the locality of memory accesses in searchable symmetric encryption (SSE). Namely, when encrypting an index of N identifier/keyword pairs, the encrypted index must have size $\omega(N)$ *or* the scheme must perform searching with $\omega(1)$ non-contiguous reads to memory *or* the scheme must read many more bits than is necessary to compute the results. Recent implementations have shown that non-locality of server memory accesses create a throughput-bottleneck on very large databases. Our lower bound shows that this is due to the security notion and not a defect of the constructions. An upper bound is also given in the form of a new SSE construction with an $O(N \log N)$ size encrypted index that performs $O(\log N)$ reads during a search.

Keywords. Symmetric Encryption, Lower Bound

1 Introduction

Searchable symmetric encryption (SSE) [24, 15, 13] enables a client to encrypt an index of record/keyword pairs and later issue tokens allowing an untrusted server to retrieve the (identifiers of) all records matching a keyword. SSE aims to hide statistics about the index to the greatest extent possible while maintaining practical efficiency for large indexes like email repositories or United States census data. These schemes employ only fast symmetric primitives and recent implementations [10] have shown that, in contrast to most applications of advanced cryptography, cryptographic processing like encryption is not the bottleneck for scaling. Instead, lower-level issues dealing with memory layouts required by the schemes are the limiting factor for large indexes.

This work studies how the security definitions for SSE inherently hamper scaling for large indexes. It proves an unconditional lower bound on the trade-off between server storage space and the *spatial locality* of its accesses to the encrypted index during a search. At a high level, the bound says that, for an index with N pairs, any secure SSE must either pad the encrypted index to an impractical (super-linear, $\omega(N)$) size *or* perform searching in a very non-local way (with $\omega(1)$ contiguous accesses or by reading far more bits than is necessary). Either of these options is likely to incur a large slow-down over a properly designed plaintext searching system with an $O(N)$-size index that can search with $O(1)$ contiguous accesses.

The issue of locality in SSE surfaced in recent works [13, 10] where implementations showed that the non-local use of external storage was a main bottleneck preventing scaling to large indexes. The only works with a highly local access pattern generated very large (roughly $O(N^2)$) encrypted databases that also prevented scaling. This paper explains this dichotomy of padding versus spatial locality by proving it is an unavoidable consequence of the SSE security definition. As more cryptographic applications are developed for securely outsourcing large amounts of data (while maintaining either authenticity or secrecy), lower-level issues like locality may become more relevant. While in some contexts (like secure multiparty computation) it is clear that the entire input must be touched during computation, this work appears to be the first to study of the effect of security on locality in detail.

The lower bound suggests the question of a matching upper bound. We give a new scheme with an $O(N \log N)$ size encrypted index and $O(\log N)$ locality via a different padding strategy, which compares to a scheme with a $O(N^2)$ size encrypted index and $O(1)$ locality.[3] This scheme may not be competitive with prior highly-optimized implementations, but it serves as intermediate point in the trade-off curve implied by the lower bound. The interesting question of closing the gap is left open.

Scheme	Leakage	EDB Size	Locality	Read Efficiency
CGKO'06-1 [13]	m, N	$O(N+m)$	$O(t_w)$	$O(1)$
CGKO'06-2 [13]	$M \cdot n$	$O(Mn)$	$O(t_w)$	$O(1)$
CK'10 [12]	m, n, M	$O(Mn)$	$O(1)$	$O(1)$
LSDHJ'10 [25]	m, n	$O(mn)$	$O(t_w)$	$O(1)$
KO'12 [20]	n, M	$O(Mn)$	$O(t_w)$	$O(1)$
KPR'12 [19]	m, N	$O(N+m)$	$O(t_w)$	$O(1)$
KP'13 [18]	m, n	$O(mn)$	$O(t_w \log n)$	$O(n \log n)$
CJJKRS'13 [10]	N	$O(N)$	$O(t_w)$	$O(1)$
This paper: Scheme	N	$O(N \log N)$	$O(\log N)$	$O(1)$
This paper: Lower Bound	Any from above	$\omega(N)$	$O(1)$	$O(1)$

Fig. 1. Comparison of some SSE schemes. Legend: Leakage is leakage from EDB only, and all schemes also leak search results and access pattern. n = total # of unique identifiers, $N = \sum_w |\mathsf{DB}(w)|$, m = total # of unique keywords, $M = \max_w |\mathsf{DB}(w)|$, $t_w = |\mathsf{DB}(w)|$ for the query w. For [12] we mean the scheme in Section 5.2 of the full version there. The lower bound is achieved with $\alpha = 0$ in Theorem 8. If a scheme support updates or more advanced searches then we consider a simplified static version for keyword searching as formalized in Section 2. Differences in security (simulation versus indistinguishability, adaptivity) are ignored here but explain why some schemes appear to be strictly worse than others.

[3] There are various ways to achieve a smaller index (see Figure 1), but these will achieve a slightly different notion of security.

SSE AND LOCALITY. Let us describe the issue in more detail, starting with SSE and its security goals. An input to an SSE scheme, denoted DB, is essentially an index associating with each keyword w a set of identifiers (bit strings) $\mathsf{DB}(w) = \{\mathrm{id}_1, \cdots, \mathrm{id}_{t_w}\}$, where the number t_w can vary. "Searching" means retrieving those identifiers, given w. An SSE scheme is a system for storing and retrieving these sets while hiding statistics about the identifier sets matching un-searched keywords such as their number, size, the size of their intersections, and so on. Security is formally parameterized with a leakage function $\mathcal{L}$ that describes an upper bound on what a server learns. One example of good leakage is $N = \sum_w |\mathsf{DB}(w)|$, along with the identifier sets $\mathsf{DB}(w)$ for each keyword that is searched for. Other statistics like $\max_w |\mathsf{DB}(w)|$ and the number of unique keywords are also usually considered acceptable leakage. In any case, the defined leakage is *all* the server should learn, so plaintext keywords, identifiers, and anything else other than the output of $\mathcal{L}$ must be hidden.

A scheme of Curtmola et al. [13] forms the basis for most subsequent SSE schemes. This scheme leaks only N and the number of unique keywords by placing all N of the identifier/keyword pairs in random order into a large array (along with some auxiliary tables) and enabling retrieval with encrypted linked lists that could be opened for the server. Searching for w requires walking through $|\mathsf{DB}(w)|$ pseudorandom locations in this large array. When the array is stored on disk, it means that each retrieved pair requires a disk read at a random location. Since each identifier is on the order of several bytes but the disk block size is now often 4KB, this searching will sacrifice throughput and latency compared to a plaintext search system which can store the identifiers together in sectors, and moreover in contiguous sectors which can be read together more efficiently without additional seeking. Naive modifications, like packing several identifiers from a single $\mathsf{DB}(w)$ set into one sector, render the scheme insecure for the leakage function they consider.

One work [12] addressed locality by enlarging the index to $\omega(N)$ size. This scheme pads every set $\mathsf{DB}(w)$ to size $\max_w |\mathsf{DB}(w)|$ and then store these padded sets in the own contiguous rows. For very large indexes (with billions of pairs as in [10]), even doubling the plaintext size may be unreasonable, so these works do not appear to scale for realistic datasets where the padding will be large.

RESULTS. In the sections that follow a precise model is given for measuring and comparing the memory usage of SSE schemes. The parameters of locality, read-efficiency, and read disjointness are defined and discussed in relation to lower bounds. Briefly, *locality* is the number of non-contiguous memory accesses made by the server, *read-efficiency* is the number of bits read beyond the minimum necessary, and a scheme has *α-overlapping reads* if reads for different searches overlap in at most α bits. In Figure 1 we compare the leakage, locality, and read efficiency of prior SSE constructions. (For read efficiency, the number listed is a multiplicative factor over the binary encoding of the identifiers matching the query.)

This paper's primary results are summarized below. For the following two theorems, let $\mathcal{L}$ be any of the leakage functions in Figure 1 (or any function efficiently computable from them). Below we write $\mathsf{BinEnc}(\mathsf{DB})$ to mean an encoding on DB as a binary string formed by concatening the lists of identifiers matching each keyword. See Sections 2 and 3 for definitions.

Theorem 1. *If Π is an $\mathcal{L}$-IND-secure SSE scheme with locality r as well as α-overlapping reads, then Π has $\omega\left(\frac{|\mathsf{BinEnc}(\mathsf{DB})|}{r\cdot(\alpha+2)}\right)$ server storage.*

We remark that a very weak read efficiency requirement is implicit in the condition on overlapping reads, and all existing schemes have highly non-overlapping reads.

Theorem 2. *Assuming one-way functions exist, there exists an $\mathcal{L}$-IND-secure SSE scheme with locality $O(\log N)$, $O(1)$ read efficiency and $O(N \log N)$ storage.*

The bulk of the paper is spent proving the first theorem. We now start with an intuitive sketch of how one might prove a weak lower bound. See Section 4 for a detailed sketch of the actual proof, which is more complicated.

Lower bound approach. Intuitively, if a scheme is very local, then after some searching the server can look at what is *not* read after several searches and infer statistics about what has not been opened. In particular, if one of the sets $\mathsf{DB}(w)$ is very large, then good locality means there is a very large region of the encrypted index that will not be touched by other searches, and the server will notice that this happens after several searches with small number of results.

The lower bound develops this intuition, but requires further ideas to achieve a lower bound of $\omega(N)$ on the server storage. For now let us sketch how one shows the server must approximately double the size of a plaintext index if it is to be *perfectly* local and read-efficient, meaning it processes a search by reading exactly the required number of bits from a single contiguous section of EDB, and moreover that the reads for all searches are disjoint. This seems highly restrictive, but later will we be able to weaken all of these assumptions to realistic versions.

Now suppose we have a perfectly local SSE scheme. Consider two index inputs, DB_0 and DB_1, where DB_0 consists of N keywords each matching a single unique document and DB_1 consists of 2 keywords matching a single unique document and a third keyword matching $N-2$ documents. If two random keywords matching single documents are searched for then the server learns which locations of the encrypted index are read in order to respond. If DB_0 was encrypted, then pigeon-hole argument shows that with constant probability, there is no remaining contiguous interval large enough to contain the bits that would be read for the third keyword (it is here that we use an assumed bound on the server storage). This is diagrammed in the top part of Figure 2, where when the red regions are read there is no longer space between them for a larger interval. This is in contrast to the case when DB_1 is encrypted, because after observing the two small reads, a perfectly local scheme there will always be a contiguous unread region large enough to hold the $N-2$ identifiers for the third keyword.

The full lower bound is an extension of this idea to consider a family of indexes with result sets of several sizes. Later it is argued that the technique above is limited to showing a factor 2 overhead in server storage, and that the complexity of the main attack seems necessary. We also address several extensions, such as when the server does not perform a single contiguous read but up to $O(1)$ reads, the leakage function parameter varies, and the reads are allowed to partially overlap.

Related work on secure searching. Following the initial work of [24] that suggested searchable encryption, Curtmola et al. [13] formalized the version of SSE that we consider in this paper. Subsequently SSE schemes were given with different efficiency properties [15, 11, 12], support for data updates [19, 18], authenticity [20] and support more advanced searches [10]. These improvements are rthogonal to the lower bound, which applies to these schemes when used for basic (non-dynamic, non-authenticated) SSE.

The problem of searching on encrypted data can be addressed in several ways using generic multiparty computation protocols, oblivious RAM schemes [16] or fully homomorphic encryption [14]. These approaches achieve slightly levels of functionality and different notions of security, meaning that the lower bound does not seem to apply. Order-preserving encryption [6, 7] takes a different approach to searching that achieves high efficiency for rich queries but is less secure than SSE. Implementations that use order-preserving encryption, notably CryptDB [22], inherit these properties. Our lower bound does not apply to them.

There is also a line of work on searching on *public-key* ciphertexts. Public-key encryption with keyword search [9, 17, 1, 2] In these schemes and subsequent work, the server performing the search by testing each encrypted record individually, resulting in a scheme that is trivial from the point of view of the lower bound. The line of work on deterministic public-key encryption [4, 8, 5] enables fast searching but achieves different, weaker security meaning our lower bound does not apply.

Related work on locality. Algorithmic performance with data stored on disk has been studied extensively in *external memory models* (c.f. [26, 23, 3]). These models usually consider block-oriented devices with varying degrees of precision (e.g., including modeling parallelism, drive geometry, memory hierarchies, caching, locality of blocks, etc.). Typically one measures the external memory efficiency of an algorithm by counting the number of blocks it accesses, and a wide array of techniques have been developed to optimize disk utilization at the algorithmic level.

Interestingly, matching lower and upper bounds are known for many natural problems like, e.g., dictionary retrieval, sorting, range searching – see, e.g., Chapter 6 of [26]. Our lower bound is fundamentally different from these results. There, one can give an information theoretic argument that a certain number of disk accesses are necessary in the worst case, with a flavor similar to the classic $O(n \log n)$ comparison-based sorting lower bound. Our lower bound, however, will proceed by showing that any SSE scheme that meets a certain level of efficiency will be *insecure* (rather than *incorrect* as in traditional external memory lower bounds). That is, our lower bound comes in the form of an attack. Due to the nature of our lower bound we opt for an extremely simplified version of locality and leave its adaptation to fine-grained external memory models to future work.

We are not aware of any prior similar lower bounds on cryptographic primitives, other than the folklore observation that security forces many primitives to touch every bit their inputs (e.g., homomorphic encryption [14], multiparty computation).

Organization. Preliminaries and definitions are recalled in Section 2. New definitions relating to locality are given and discussed in Section 3. The lower bound is stated and proved in Section 4, and the upper bound is in Section 5.

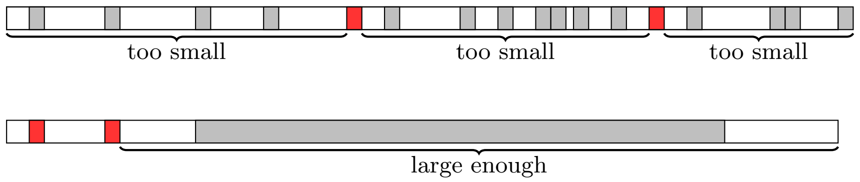

Fig. 2. Intuition for a basic lower bound.

2 Preliminaries

Throughout this paper the security parameter is denoted λ and all algorithms (and adversaries) are assumed to run in time polynomial in λ. We write $[n]$ for the set $\{1, \ldots, n\}$. For a vector $\mathbf{v}$ we write $|\mathbf{v}|$ for the dimension (length) of $\mathbf{v}$ and for $i \in [|\mathbf{v}|]$ we write $\mathbf{v}[i]$ for the i-th component of $\mathbf{v}$. For a bitstring s, we write $s[a, b]$ for the substring starting with the bit in position a and ending in position b.

Databases and SSE schemes. An index (or database) $\mathsf{DB} = (\mathrm{id}_i, \mathsf{W}_i)_{i=1}^n$ is a list of identifier/keyword-set pairs, where each $\mathrm{id}_i \in \{0,1\}^\lambda$ and each W_i is a set of bitstrings. When the DB under consideration is clear, we will write $\mathsf{W} = \bigcup_{i=1}^n \mathsf{W}_i$. For a keyword $w \in \mathsf{W}$, we write $\mathsf{DB}(w)$ for $\{\mathrm{id}_i : w \in \mathsf{W}_i\}$. We will always use $N = \sum_{w \in \mathsf{W}} |\mathsf{DB}(w)| = \sum_{i=1}^n |\mathsf{W}_i|$ to mean the total number of keyword/identifier pairs in DB, n to mean the number of unique identifiers, and $m = |\mathsf{W}|$ to mean the number of unique keywords.

A *searchable symmetric encryption (SSE) scheme* Π consists of algorithms $(\mathsf{KeyGen}, \mathsf{EDBSetup}, \mathsf{TokGen}, \mathsf{Search})$ that satisfy the following syntax. The key generation algorithm KeyGen takes as input the security parameter and outputs a key K. The algorithm $\mathsf{EDBSetup}$ takes as input a key K and a database DB and outputs an encrypted database EDB. The token generation protocol takes as input a string w and key K and outputs a token τ. Finally, the searching algorithm Search takes as input τ and EDB and outputs as set L of results.

We note that formalization of an SSE scheme does not model the storage of actual document payloads, but only of metadata encoded in a keyword index.

This simplifies the definition and makes it modular, but some care must be taken when combining an SSE scheme with a document storage scheme (see e.g. [13] for an example of how to store the payloads).

An SSE scheme is *correct* if the natural usage returns the correct results for the keyword being searched (i.e., $\mathsf{DB}(w)$), except with negligible probability. Formally, for every database DB, consider an experiment where $K \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$ and $\mathsf{EDB} \xleftarrow{\$} \mathsf{EDBSetup}(K, \mathsf{DB})$ are initially sampled. Then, an attacker learns EDB and can issue adaptive queries w_i, which are answered by first generating a token $\tau_i \leftarrow \mathsf{TokGen}(K, w_i)$ and then returning it together with $S_i \leftarrow \mathsf{Search}(\tau_i, \mathsf{EDB})$ to the attacker. The scheme is *correct* if for all polynomial-time attackers, $S_i = \mathsf{DB}(w_i)$ for all i, except with negligible probability.

We say that the scheme Π has *server storage* $s(N, \lambda)$ if on input a database DB with N keyword/identifier pairs and a key $K \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$, $\mathsf{EDBSetup}$ outputs EDB such that $|\mathsf{EDB}| = s(N, \lambda)$, where $|\mathsf{EDB}|$ is the bit-length of EDB.

Security. We recall the non-adaptive indistinguishability-based version of security from [13] which will be considered in the lower bound.

Definition 3. *Let* $\Pi = (\mathsf{KeyGen}, \mathsf{EDBSetup}, \mathsf{TokGen}, \mathsf{Search})$ *be an SSE scheme and let* $\mathcal{L}$ *be a leakage function and* $\mathcal{A}$ *be an adversary. For* $b \in \{0, 1\}$ *we define the game* $\text{IND-SSE}^b_{\Pi,\mathcal{L},\mathcal{A}}(\lambda)$ *as follows: The adversary chooses* $\mathsf{DB}_0, \mathsf{DB}_1, \mathbf{w}$*. The game runs* $K \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$*,* $\mathsf{EDB} \xleftarrow{\$} \mathsf{EDBSetup}(K, \mathsf{DB}_b)$ *and* $\boldsymbol{t}[i] \leftarrow \mathsf{TokGen}(K, \mathbf{w}[i])$ *for each* $i \in [|\mathbf{w}|]$*. It gives* $(\mathsf{EDB}, \boldsymbol{t})$ *to* $\mathcal{A}$*, which outputs a bit* $\hat{b}$*. Finally, if* $\mathcal{L}(\mathsf{DB}_0, \mathbf{w}) \neq \mathcal{L}(\mathsf{DB}_1, \mathbf{w})$*, the game outputs* $\perp$ *and otherwise it outputs* $\hat{b}$*.*

We define the $\mathcal{L}$-IND advantage *of* $\mathcal{A}$ *to be*

$$\mathbf{Adv}^{\text{ind-sse}}_{\Pi,\mathcal{L},\mathcal{A}}(\lambda) = |\Pr[\text{IND-SSE}^0_{\Pi,\mathcal{L},\mathcal{A}}(\lambda) = 1] - \Pr[\text{IND-SSE}^1_{\Pi,\mathcal{L},\mathcal{A}}(\lambda) = 1]| \,,$$

and we say that Π *is* $\mathcal{L}$*-IND-secure if* $\mathbf{Adv}^{\text{ind-sse}}_{\Pi,\mathcal{L},\mathcal{A}}(\lambda)$ *is negligible for every* $\mathcal{A}$*.*

Our construction will achieve the stronger (adaptive, simulation-based) definition from [13], which we recall here. (A non-adaptive version is such that in both games $\mathcal{A}$ must choose all of its queries beforehand.)

Definition 4. *Let* $\Pi = (\mathsf{KeyGen}, \mathsf{EDBSetup}, \mathsf{TokGen}, \mathsf{Search})$ *be an SSE scheme and let* $\mathcal{L}$ *be a leakage function. For algorithms* $\mathcal{A}$ *and* $\mathcal{S}$*, we define the two games* $\text{SIM-SSE}^0{}_{\Pi,\mathcal{L},\mathcal{A}}(\lambda)$ *and* $\text{SIM-SSE}^1{}_{\Pi,\mathcal{L},\mathcal{A},\mathcal{S}}(\lambda)$ *as follows:*

$\text{SIM-SSE}^0{}_{\Pi,\mathcal{L},\mathcal{A}}(\lambda)$*:* $\mathcal{A}(1^\lambda)$ *chooses* $\mathsf{DB}, \mathbf{w}$*. The game then runs* $(K, \mathsf{EDB}) \leftarrow \mathsf{EDBSetup}(\mathsf{DB})$ *and* $\boldsymbol{t}[i] \leftarrow \mathsf{TokGen}(K, \mathbf{w}[i])$ *for each* $i \in [|\mathbf{w}|]$*. It gives* $\mathsf{EDB}, \boldsymbol{t}$ *to* $\mathcal{A}$*, which eventually returns a bit that the game uses as its own output.*

$\text{SIM-SSE}^1{}_{\Pi,\mathcal{L},\mathcal{A},\mathcal{S}}(\lambda)$*:* $\mathcal{A}(1^\lambda)$ *chooses* $\mathsf{DB}, \mathbf{w}$*. The game then runs* $(\mathsf{EDB}, \boldsymbol{t}) \leftarrow \mathcal{S}(\mathcal{L}(\mathsf{DB}, \mathbf{w}))$ *and gives* $\mathsf{EDB}, \boldsymbol{t}$ *to* $\mathcal{A}$*, which eventually returns a bit that the game uses as its own output.*

We define the $\mathcal{L}$*-SIM-advantage of* $\mathcal{A}$ *and* $\mathcal{S}$ *to be*

$$\mathbf{Adv}^{\text{sim-sse}}_{\Pi,\mathcal{L},\mathcal{A},\mathcal{S}}(\lambda) = |\Pr[\text{SIM-SSE}^0{}_{\Pi,\mathcal{L},\mathcal{A}}(\lambda) = 1] - \Pr[\text{SIM-SSE}^1{}_{\Pi,\mathcal{L},\mathcal{A},\mathcal{S}}(\lambda) = 1]|,$$

and we say that Π is $\mathcal{L}$-SIM-secure *if for all adversaries $\mathcal{A}$ there exists an algorithm $\mathcal{S}$ such that $\mathbf{Adv}^{\text{sim-sse}}_{\Pi,\mathcal{L},\mathcal{A},\mathcal{S}}(\lambda)$ is negligible.*

Leakage functions. Below we will consider two leakage functions $\mathcal{L}_{\min}$ and $\mathcal{L}_{\max}$. The first is called the *size minimal leakage function,*[4] which is defined as follows: $\mathcal{L}_{\min}(\mathsf{DB}, \mathbf{w})$ outputs $N = \sum_{w \in \mathsf{W}} |\mathsf{DB}(w)|$ and the sets $(\mathsf{DB}(\mathbf{w}[1]), \ldots, \mathsf{DB}(\mathbf{w}[|\mathbf{w}|]))$. The second is called the *maximal leakage function* which outputs (N, n, m, M) as well as $(\mathsf{DB}(\mathbf{w}[1]), \ldots, \mathsf{DB}(\mathbf{w}[|\mathbf{w}|]))$, where N is defined as before, n is the number of unique identifiers in DB, $m = |\mathsf{W}|$ (the number of unique keywords), and $M = \max_w |\mathsf{DB}(w)|$. It is of course possible to consider "more" leakage, but this is more than any existing scheme leaks, meaning out lower bound will apply to all of them.

3 Read Efficiency and Locality Metrics for SSE Schemes

This section introduces the notions of locality and read efficiency of SSE schemes.

Read patterns. First, we observe that the searching procedure of any SSE scheme can be decomposed into a sequence of *contiguous* reads from the encrypted database. To formalize this point of view, fix an SSE scheme Π, an EDB output by $\mathsf{EDBSetup}$ and a token τ output by TokGen. Viewing EDB as a bitstring of length M, we may express the computation of $\mathsf{Search}(\tau, \mathsf{EDB})$ as follows: It starts by computing an interval $[a_1, b_1]$ that depends only on τ. It then computes another interval $[a_2, b_2]$ that depends only on τ and $\mathsf{EDB}[a_1, b_1]$, and continues computing intervals to read based on τ and all previously read intervals from EDB. We write $\mathsf{RdPat}(\tau, \mathsf{EDB})$ for these intervals. In the following, denote as $\mathsf{BinEnc}(\mathsf{DB}(w))$ the binary *representation* of $\mathsf{DB}(w)$, i.e., the concatenation of all identifiers represented as bit strings, and $\mathsf{BinEnc}(\mathsf{DB})$ to be the concatenation of all the $\mathsf{BinEnc}(\mathsf{DB}(w))$ for each w in the database. Under our assumption that all identifiers are in $\{0,1\}^\lambda$, we have $|\mathsf{BinEnc}(\mathsf{DB})| = \lambda |\mathsf{DB}(w)|$ and $\mathsf{BinEnc}(\mathsf{DB}) = \lambda N$.

Locality of an SSE scheme. We put forward the notion of locality of an SSE scheme, capturing the fact that every read pattern consists of at most a bounded number of intervals.

Definition 5 (Locality). *An SSE scheme Π is* r-local *(or has locality r) if for any λ, DB, and $w \in \mathsf{W}$, we have that $\mathsf{RdPat}(\tau, \mathsf{EDB})$ consists of at most r intervals with probability 1 when EDB, τ are computed as $K \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$, $\mathsf{EDB} \xleftarrow{\$} \mathsf{EDBSetup}(K, \mathsf{DB})$, $\tau \leftarrow \mathsf{TokGen}(K, w)$. If $r = 1$, we say Π has* perfect *locality.*

In particular, the value r can depend both on the security parameter λ *and* the index size $|\mathsf{DB}|$.

[4] It appears to be impossible to define a true "minimal" amount of leakage, as we could consider a leakage function that leaks only some upper bound on N.

Read efficiency. The notion of locality alone is not very meaningful. Of course, we can just make every scheme perfectly local by reading the whole EDB. This is why the notion of locality is directly tied to the notion of *read efficiency*, which measures the overall size of the portion read by a search operation.

Definition 6 (Read Efficiency). *An SSE scheme Π is* c-read efficient *(or has read efficiency c) if for any λ, DB and $w \in \mathsf{W}$, we have that $\mathsf{RdPat}(\tau, \mathsf{EDB})$ consists of intervals of total length at most $c \cdot |\mathsf{BinEnc}(\mathsf{DB}(w))|$ bits.*

We allow c to depend on the security parameter here.

Read disjointness. The above definition of read efficiency is very general. In particular, for sufficiently large c, it allows multiple queries to read *exactly* the same bits. Our lower bound below will apply to a more restricted class of r-local schemes which read sufficiently many new bits. We feel this class is natural, and moreover it contains all prior constructions.

Definition 7 (Overlapping reads). *An SSE scheme Π has* α-overlapping reads *if for all λ and all DB, the read pattern induced by the search of each keyword in DB has an overlap of at most α with the read patterns induced by the searches of all previous keywords (with probability 1 over the computation of $K \leftarrow \mathsf{KeyGen}(1^\lambda)$, $\mathsf{EDB} \leftarrow \mathsf{EDBSetup}(K, \mathsf{DB})$, and the computation of the tokens). When $\alpha = 0$ we say Π has* disjoint reads*.*

In general, the value α is independent of N, but may additionally depend on λ or possibly on the number of words $|\mathsf{W}|$ in order for example to take into account a common portion of EDB which can be read at every search operation. Typically, a scheme will read some metadata like hash table entries and then perform reads to retrieve the actual results. (Of course we make no assumption on what computation the scheme actually does, beyond being of the form above.)

4 Lower Bound

In this section we sketch our proof that a secure SSE scheme cannot simultaneously achieve $O(1)$ locality and $O(|\mathsf{BinEnc}(\mathsf{DB})|)$ server storage. Concretely, we are going to prove the following theorem, where $\mathcal{L}_{\max}$ was defined at the end of Section 2 and the locality metrics were defined in the previous section.

Theorem 8. *If Π is an $\mathcal{L}_{\max}$-IND-secure SSE scheme with locality r as well as α-overlapping reads, then Π has $\omega\left(\frac{|\mathsf{BinEnc}(\mathsf{DB})|}{r \cdot (\alpha+2)}\right)$ server storage.*

We note that we consider $\mathcal{L}_{\max}$ for the lower bound as this strengthens the result by considering schemes that are "very leaky." In the theorem statement, we assume that α does not depend on N, but may additionally depend on λ or possibly on the number of words $|\mathsf{W}|$.

The proof is rather long and will not fit in this version of the paper (a full proof was submitted and will appear in the full version). Instead, we provide a sketch of the proof approach and its implementation.

Proof approach. We will first sketch our lower bound with a few simplifications. First, we assume the SSE scheme is has perfect locality and read efficiency, meaning the server always performs exactly one contiguous read for exactly $|\mathsf{BinEnc}(\mathsf{DB}(w))| = \lambda \cdot |\mathsf{DB}(w)|$ bits from EDB when searching for a word w, the minimum required for the response. Second, we assume all reads are perfectly disjoint. Third, we consider the lower bound leakage against SSE schemes achieving security with leakage function $\mathcal{L}_{\min}$ instead of $\mathcal{L}_{\max}$ (thus making the result easier). It turns out that this case encompasses most of the technical difficulties for the general result, which we derive afterwards.

The principle behind the attack extends the idea sketched in Section 1. The adversary will choose two indexes $\mathsf{DB}_0, \mathsf{DB}_1$ of the same size in a careful way so that DB_1 has keywords that match a large number of documents while DB_0 does not. Then it will query for tokens for several keywords matching relatively small numbers of documents. Using the tokens, it will compute the read pattern of the server when searching for those keywords, and then look at the *unread* portions of EDB. Since we are assuming perfect locality, if DB_1 was encrypted, there must be large regions that go untouched by any query. On the other hand, we will show that this is sometimes not the case if DB_0 is encrypted, allowing the adversary to distinguish.

To describe the proof it will be useful to introduce a compact notation for the *shape* of a DB input.

Definition 9. *We write*

$$\mathsf{DB} \leftarrow (n_1 \times s_1;\ n_2 \times s_2;\ \dots\ ;\ n_t \times s_t)$$

when $\mathsf{DB} = (\mathrm{id}_i, \mathsf{W}_i)$ *has* shape $(n_1 \times s_1;\ n_2 \times s_2;\ \dots\ ;\ n_t \times s_t)$*, which means that it satisfies the following:*

- DB *has a keyword set* W *of size* $\sum_{j=1}^{t} n_j$ *comprised of* λ*-bit strings.*
- *For each* $j \in [t]$*, there are* n_j *keywords* $w \in \mathsf{W}$ *such that* $|\mathsf{DB}(w)| = s_j$*.*
- *For all* $w \neq w'$*, the sets* $\mathsf{DB}(w)$ *and* $\mathsf{DB}(w')$ *are disjoint.*

Our attack sketched in the introduction corresponded to picking indexes $\mathsf{DB}_0, \mathsf{DB}_1$ with shapes $\mathsf{DB}_0 \leftarrow (N \times 1)$ and $\mathsf{DB}_1 \leftarrow (1 \times 1\ ; 1 \times N-2)$, meaning that DB_0 consists of N "singletons" and DB_1 consists of two singletons and one large set of results. It is possible to formalize that attack and show that a secure perfectly local SSE scheme must produce an EDB that is at least twice as large as the bit representation of DB, but as we observe at the end of this section, any attack that uses indexes with such simple shapes will not be able to prove a better lower bound.

We now proceed to extend that attack. Let Π be perfectly local scheme with server storage $k\lambda N$ for some constant $k \geq 1$. Our attack against the security Π will select two random inputs $\mathsf{DB}_0, \mathsf{DB}_1$ with shapes

$$\mathsf{DB}_0 \leftarrow (n_1 \times \varepsilon_1 N;\ n_2 \times \varepsilon_2 N;\ \dots\ ;\ n_{k-1} \times \varepsilon_{k-1} N;\ \hat{n}_k \times \varepsilon_k N) \tag{1}$$

$$\mathsf{DB}_1 \leftarrow (n_1 \times \varepsilon_1 N;\ n_2 \times \varepsilon_2 N;\ \dots\ ;\ n_{k-1} \times \varepsilon_{k-1} N;\ n_k \times \varepsilon_k N;\ \hat{n}_{k+1} \times \varepsilon_{k+1} N) \tag{2}$$

where $n_1 > n_2 > \cdots > n_k > 1$ and $\varepsilon_1 < \varepsilon_2 < \ldots < \varepsilon_k < \varepsilon_{k+1} < 1$ are appropriately chosen constants. Intuitively, DB_0 consists only of many small result sets $\mathsf{DB}_0(w)$ while DB_1 consists of many small result sets and some relatively large sets of $\varepsilon_{k+1}N$ keywords.

The attack will query for tokens for all $\sum_{i=1}^{k} n_i$ keywords matching $\varepsilon_i N$ documents with $i \in [k]$. It then calculates the read patterns of the server. Since Π is perfectly local, the reads are for $\varepsilon_i \lambda N$ bit intervals respectively, and moreover they are all disjoint. Thus if DB_1 is encrypted, then the perfect locality of Π means there must exist an interval of $\varepsilon_{k+1}\lambda N$ bits in EDB that was not touched by the observed reads (actually, there will be at least $\hat{n}_{k+1}$ such intervals) – These are where the large result sets are stored (note that the adversary does not query for the keyword corresponding to any of large sets, but only notices the presence of a suspiciously large untouched interval). However, as we will show, if DB_0 was encrypted then the security of Π will mean there is a noticeable probability that there is no such interval remaining untouched. We stress that this will be due to the (forced) distribution of the reads, and not simply because there is no room, as the same number of bits is read when either DB_0 or DB_1 in encrypted.

Proving the latter claim on DB_0 is the main technical part of the proof. Intuitively, it holds because security forces the small intervals (say of size $\varepsilon_i \lambda N$) to be located in random-looking locations which do not leave large gaps (say of size $\varepsilon_{i+1}\lambda N$) between them too often, which implies that large intervals cannot fit between them while remaining disjoint, effectively "killing" that space for large intervals.

We will show that, for a specific choice of the constants, that for each $i = 1, \ldots, k$, the queried sets of size $\varepsilon_i \lambda N$ will each kill at least λN bits of the EDB from storing any larger intervals (in particular, intervals of size $\varepsilon_j \lambda N$ for $j > i$) with constant probability. Since we have k different read pattern sets each killing λN bits from storing anything larger with constant probability, we get that $k\lambda N$ bits are killed with constant probability. But this means the entire EDB has been killed with constant probability (here we use that k is a constant), and when that happens the adversary can conclude that DB_0 was encrypted – If DB_1 had been encrypted this would happen with probability 0.

We now discuss how to show that queries for a constant number of sets of size $\varepsilon_i N$ will kill λN bits (which is much larger than the actual number of bits read by the server during its perfectly-local searching). To prove this we will consider a sequence of adversaries $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k$ - the purpose of $\mathcal{A}_i$ is to show that the sets of size $\varepsilon_i N$ kill enough space with constant probability, assuming that the smaller sets each do so. The first adversary $\mathcal{A}_1$ is the simplest to describe, and resembles our original attack from the Introduction. Adversary $\mathcal{A}_1$ draws $\mathsf{DB}_0, \mathsf{DB}_1$ of size N with shapes

$$\mathsf{DB}_0 \leftarrow (\hat{n}_1 \times \varepsilon_1 N) \tag{3}$$

$$\mathsf{DB}_1 \leftarrow (n_1 \times \varepsilon_1 N;\ \hat{n}_2 \times \varepsilon_2 N), \tag{4}$$

where $n_1 < \hat{n}_1 = \varepsilon_1^{-1}$ and $\varepsilon_1 < \varepsilon_2$ are constants. It populates the two databases with a consistent set of keywords, meaning that the n_1 keywords matching $\varepsilon_1 N$ documents DB_1 are a random subset of the $\hat{n}_1$ such keywords in DB_0. Intuitively, DB_0 has a large number of keywords matching $\varepsilon_1 N$ documents each, and searching for each keyword induces a read by the server for a disjoint interval $\lambda\varepsilon_1 N$ bits. Thus searching for a random subset of $n_1 < \hat{n}_1$ of those keywords will reveal the location of a random subset of the disjoint intervals. In DB_1, however, there are only n_1 of these keywords, but we can show that security forces their distributions to be as they are in DB_0.

Specifically, we have $\mathcal{A}_1$ query for the n_1 keywords matching $\varepsilon_1 N$ documents in either database, and then it computes their read patterns. If DB_0 was encrypted, then the intervals read by the server are chosen randomly from amongst $\hat{n}_1$ intervals of that size. We show (unconditionally) with good probability there is a lot of space (about λN bits) in EDB where intervals of size $\varepsilon_2 N$ or larger cannot fit after the n_1 intervals have been read. This happens because, for randomly chosen intervals, the gap between them cannot be larger than $\varepsilon_2 N$ too often. Thus the larger intervals must go elsewhere in EDB. And since the scheme is secure, DB_1 must also exhibit this behavior (despite the read intervals not being chosen from a larger set of intervals). In fact, this shows that when any database contains n_1 keywords with εN results each, then the resulting reads for those keywords must be laid out in a way that eliminates a large amount of space for larger intervals even though the actual bits read for them is very small, namely $n_1\varepsilon_1\lambda N \ll \hat{n}_1\varepsilon_1\lambda N$.

We then iterate this approach; The next adversary $\mathcal{A}_2$ queries $\mathsf{DB}_0, \mathsf{DB}_1$ with shapes

$$\mathsf{DB}_0 \leftarrow (n_1 \times \varepsilon_1 N;\ \hat{n}_2 \times \varepsilon_2 N) \tag{5}$$

$$\mathsf{DB}_1 \leftarrow (n_1 \times \varepsilon_1 N;\ n_2 \times \varepsilon_2 N;\ \hat{n}_3 \times \varepsilon_3 N). \tag{6}$$

(So DB_0 now has the shape that $\mathcal{A}_1$ chose for DB_1.) The adversary $\mathcal{A}_2$ then queries for tokens for all n_1 keywords matching $\varepsilon_1 N$ documents, and then a random subset of the $\hat{n}_2$ keywords matching $\varepsilon_2 N$ documents in DB_1. (As before these databases are made with consistent keywords and identifiers.) We show that when DB_0 is encrypted, conditioned on the read intervals of size $\varepsilon_1\lambda N$ disallowing λN bits for larger intervals, a random subset of intervals of size $\varepsilon_2\lambda N$ will disallow about another λN bits for larger intervals. Security again forces this is to be true when DB_1 is encrypted despite not being forced statistically. The result is that with constant probability about $2\lambda N$ bits of EDB can no longer accommodate larger intervals.

By considering the sequence of $\mathcal{A}_1, \ldots, \mathcal{A}_k$ of adversaries and applying this reasoning k times, we have that the entire database has been disallowed by a relatively small number of small reads intervals with good probability, and then we can finish the proof as sketched above.

Extensions to more general locality. The argument above worked for *perfect* locality, meaning the server search algorithm for keyword w worked with a single, contiguous read from EDB for exactly $\lambda \cdot |\mathsf{DB}(w)|$ bits that is disjoint from the read for any other search. It is easy to extend the lower bound to when the server works with $r = O(1)$ contiguous reads that total exactly $\lambda \cdot |\mathsf{DB}(w)|$ bits and are disjoint from all other reads by observing that one of the r reads must have size at least $\lambda \cdot |\mathsf{DB}(w)|/r$ contiguous bits, and then adjusting the parameters of the above argument to ensure that intervals of that size can be be disallowed with good probability by the final adversary.

Other relaxations will be given. For instance, to adaptive the attack to work with leakage function $\mathcal{L}_{\max}$, we need to additionally arrange for the submitted databases to always have the same number of documents, keywords, maximum size result set $\mathsf{DB}(w)$. This only introduces minor technicalities.

Would a simpler attack work? It is fair to ask if the complexity of this attack is necessary, and specifically if an attack like $\mathcal{A}_1$, which only queries for keyword with $|\mathsf{DB}(w)|$ equal to two possible sizes (either $\varepsilon_1 N$ or $\varepsilon_2 N$) could give the lower bound and avoid the iterative argument.

While it is always possible in principle to simplify proofs, we *can* argue that no such simple adversary could prove a lower bound better than $M \geq 2\lambda N$ by observing read patterns alone. This is because an SSE scheme, knowing that it will only be queried for keywords with two different sizes $|\mathsf{DB}(w)|$, could have EDB reserve λN bits for the first size, and another λN bits for the second size. Then it could simply store sets $\mathsf{DB}(w)$ of the first size in random order first half of EDB with padding, then the sets $\mathsf{DB}(w)$ with the second size in the second half.

This reasoning generalizes to show that any attack proving $M \geq k\lambda N$ must query keywords with $|\mathsf{DB}(w)|$ having at least $k+1$ different sizes, as our attack does.

5 A Positive Result: SSE with Logarithmic Locality

In the previous section, we have seen that any scheme with *constant* locality produces encrypted index of size $\omega(N)$. To complement this result, we provide a new scheme with logarithmic locality, at the cost of an asymptotically larger encrypted index of size roughly $N \log N$. At the same time, our scheme is going to *only* leak the database size N, i.e., it is going to be $\mathcal{L}_{\min}$-secure. None of the previous SSE schemes achieved such locality level without additional leakage or a larger worst-case blow-up of the encrypted database.

Hash tables. The scheme below relies on *hash tables.* Concretely, a hash table *implementation* consists of a pair of algorithms $(\mathsf{HTCreate}, \mathsf{HTGet})$. The function $\mathsf{HTCreate}$ takes as input a list $L = \{(l_i, d_i)\}_{1 \leq i \leq k}$ of pairs (l_i, d_i) of strings, where $l_i \in \{0,1\}^\ell$ is the *label* and $d_i \in \{0,1\}^r$ is the *data*, and outputs the *hash table* HT. After running $\mathsf{HT} \leftarrow \mathsf{HTCreate}(L)$, we have that $\mathsf{HTGet}(\mathsf{HT}, l)$ returns d if and only if $(l, d) \in L$, and returns $\perp$ otherwise.

There exist hash-table implementations (for example, via variants of cuckoo hashing [21]) with the following properties: The overall size of HT is $O(k(r +$

$\ell) + \log^2 k)$, and the algorithm HTGet needs to read from the hash table HT a constant number (e.g. two) of blocks of ℓ contiguous bits, as well as one r-bit block, when searching for a label $l = l_i$. Moreover, HT does not depend on the ordering of the list L.

Description of the scheme. We now proceed to specify our new SSE scheme $\Pi = (\mathsf{KeyGen}, \mathsf{EDBSetup}, \mathsf{TokGen}, \mathsf{Search})$ with logarithmic locality. It relies on two keyed functions F and F', where $F : \mathcal{K} \times \{0,1\}^* \to \mathcal{K} \times \mathcal{K}'$ and $F' : \mathcal{K} \times \mathbb{N} \to \{0,1\}^\ell$ (both later to be assumed as pseudorandom). Moreover, it uses a symmetric encryption scheme $(\mathcal{E}, \mathcal{D})$ with key space $\mathcal{K}'$ and m-bit ciphertexts. In particular, we are going to use the latter scheme to encrypt document identifiers and we are going to assume that all identifiers are in the message space of the scheme $(\mathcal{E}, \mathcal{D})$ and their encryption results in ciphertext of *exactly* length s.

The four algorithms of Π now operate as follows:

Key Generation. Algorithm KeyGen simply generates a key $K \xleftarrow{\$} \mathcal{K}$ for F.

Setup. Assume that we are given DB with size $N = 2^t$ for some $t \geq 1$, and for every word $w \in \mathsf{W}$, we use the notation $\mathsf{DB}(w) = \{\mathrm{id}_1, \ldots, \mathrm{id}_{n_w}\}$ to denote its n_w associated identifiers. We also need to consider the binary expansion $n_w = \sum_{i=0}^{t-1} n_{w,i} \cdot 2^i$. (If N is not a power of two, we need to pad DB to satisfy this by adding some dummy keyword-identifier pairs.)
Algorithm $\mathsf{EDBSetup}$, on input DB and K, proceeds as follows: It initially sets up t empty lists $L_0, L_1, \ldots, L_{t-1}$. For every word $w \in \mathsf{W}$, it then computes two derived keys $F_K(w) = (K_{w,0}, K_{w,1})$ and sets $c = 0$. Subsequently, for all $i = 0, \ldots, t-1$, if $n_{w,i} = 1$, we define the ℓ-bit label $l = F'_{K_{w,0}}(i)$ and the $(2^i \cdot s)$-bit data

$$d = \mathcal{E}(K_{w,1}, \mathrm{id}_c) \,\|\, \ldots \,\|\, \mathcal{E}(K_{w,1}, \mathrm{id}_{c+2^i}) \ ,$$

increase c by 2^i, and add (l, d) to L_i. Once done with the iteration, for all $i = 0, \ldots, t-1$, we first add pairs (l, d) to L_i until it contains *exactly* 2^{t-i} elements, where l is a random label and d is a random $(2^i \cdot s)$-bit string, and then compute $\mathsf{HT}_i \leftarrow \mathsf{HTCreate}(L_i)$. The final output is

$$\mathsf{EDB} = \mathsf{HT}_0 \,\|\, \mathsf{HT}_1 \,\|\, \ldots \,\|\, \mathsf{HT}_{t-1} \ .$$

Token Generation. Algorithm TokGen, on inputs K and w, computes and outputs the two derived keys $(K_{w,0}, K_{w,1}) \leftarrow F_K(w)$.

Search. The search algorithm Search, on input $\mathsf{EDB} = \mathsf{HT}_0 \,\|\, \mathsf{HT}_1 \,\|\, \ldots \,\|\, \mathsf{HT}_{t-1}$ and (K_0, K_1), initially defines an empty response set $R = \emptyset$. Then, for all $i = 0, \ldots, t-1$, it computes $l \leftarrow F'_{K_0}(i)$ and $d \leftarrow \mathsf{HTGet}(\mathsf{HT}_i, l)$. If $d = C_1 \,\|\, \ldots \,\|\, C_{2^i} \neq \perp$, it adds $\mathcal{D}(K_1, C_1), \ldots, \mathcal{D}(K_1, C_{2^i})$ to the response set R. At the end, it outputs R.

Correctness, complexity and locality. Correctness of the SSE scheme Π holds with high probability assuming pseudorandomness of F and F' – we dispense with a formal analysis.

Assume now that we use the space- and lookup-efficient hash-table implementation mentioned above. Note first that every L_i is going to always contain 2^{t-i} elements consisting of a pair (l, d) where $|l| = \ell$ and $|d| = 2^i \cdot s$. Indeed, we cannot add more than 2^{t-i} pairs (before possibly filling up L_i) because each such pair is associated with 2^i keyword-identifier pairs, and overall there are $N = 2^t$ such pairs. For this reason, the size of HT_i is going to be $O(N(\ell + s) + \log(N)^2)$, and thus the overall size of EDB is

$$|\mathsf{EDB}| = O(N \log N \cdot (\ell + s) + \log(N)^3) \ .$$

As for locality, by the property of the hash tables, we are going to read $O(1)$ blocks of consecutive values for every $i = 0, \ldots, t-1$, thus obtaining locality $O(\log N)$. Also, read efficiency is constant.

Security. We turn to the security of the SSE scheme Π. We start with non-adaptive security, and below discuss the changes necessary in order to prove adaptive security in the random-oracle model. Here, we are going to prove that the scheme achieves the strong notion $\mathcal{L}_{\min}$-SIM-security. Recall that we say that $(\mathcal{E}, \mathcal{D})$ has *pseudorandom ciphertexts* if no polynomial-time attacker can decide whether a given oracle is behaving as $\mathcal{E}_K(\cdot)$ for random secret key K or whether it is returning a fresh random string upon each invocation, except with negligible advantage.

Theorem 10 (Non-adaptive Security of Π.). *The above SSE-scheme Π is $\mathcal{L}_{\min}$-SIM-secure against non-adaptive attacks if F and F' are pseudorandom functions and $(\mathcal{E}, \mathcal{D})$ has pseudorandom ciphertexts.*

Proof. Recall that in a non-adaptive attack, the attacker $\mathcal{A}$ first commits to keyword queries $\mathbf{w}$ and a database DB. We also recall that in the real experiment $\text{SIM-SSE}^0{}_{\Pi,\mathcal{L}_{\min},\mathcal{A}}(\lambda)$, KeyGen is run, resulting in a key K, and then $\mathsf{EDBSetup}$ is run, producing the encrypted database EDB. The attacker $\mathcal{A}$ is then given EDB, together with the search tokens $(K_{\mathbf{w}[i],0}, K_{\mathbf{w}[i],1}) = F_K(\mathbf{w}[i])$ for $i \in [|\mathbf{w}|]$. In contrast, in the ideal experiment $\text{SIM-SSE}^1{}_{\Pi,\mathcal{L}_{\min},\mathcal{A},\mathcal{S}}(\lambda)$, the simulator $\mathcal{S}$ initially only obtains $N = |\mathsf{DB}|$ and $\mathsf{DB}(\mathbf{w}[i])$ for $i \in [|\mathbf{w}|]$, and needs to output EDB' as well as search tokens $(K'_{\mathbf{w}[i],0}, K'_{\mathbf{w}[i],1})$ such that

$$\Pr[\text{SIM-SSE}^0{}_{\Pi,\mathcal{L}_{\min},\mathcal{A}}(\lambda) \Rightarrow 1] - \Pr[\text{SIM-SSE}^1{}_{\Pi,\mathcal{L}_{\min},\mathcal{A},\mathcal{S}}(\lambda) \Rightarrow 1] = \text{negl}(\lambda) \ .$$

Concretely, the simulator $\mathcal{S}$ operates as follows, assuming $N = 2^t$. First, it creates random and independent tokens $(K_{\mathbf{w}[i],0}, K_{\mathbf{w}[i],1})$ for $i \in [|\mathbf{w}|]$ and initializes empty sets $L_0, \ldots, L_{t-1}$. For every $i \in [|\mathbf{w}|]$, it then does the following, with $\mathsf{DB}(\mathbf{w}[i]) = \{\text{id}_1, \ldots, \text{id}_{n_w}\}$ and $n_{\mathbf{w}[i]} = \sum_{j=0}^{t-1} n_{\mathbf{w}[i],j} \cdot 2^j$. It sets $c = 0$, and for every $j = 0, \ldots t-1$, if $n_{\mathbf{w}[i],j} = 1$, it computes

$$d = \mathcal{E}(K_{\mathbf{w}[i],1}, \text{id}_c) \,\|\, \ldots \,\|\, \mathcal{E}(K_{\mathbf{w}[i],1}, \text{id}_{c+2^j}) \ ,$$

adds $(F_{K_{\mathbf{w}[i],0}}(j), d)$ to L_j, and increases c by 2^j. Once done with the iteration, for all $j = 0, \ldots, t-1$, the simulator adds pairs (l, d) to L_j until it contains *exactly* 2^{t-j} elements (where l is a random ℓ-bit label and d is a random $(2^j \cdot s)$-bit string) and computes $\mathsf{HT}_j \leftarrow \mathsf{HTCreate}(L_j)$. The final output is $\mathsf{EDB}' = \mathsf{HT}_0 \,\|\, \mathsf{HT}_1 \,\|\, \ldots \,\|\, \mathsf{HT}_{t-1}$, together with the tokens $(K_{\mathbf{w}[i],0}, K_{\mathbf{w}[i],1})$ for $i \in [|\mathbf{w}|]$.

The proof now proceeds via a hybrid argument. The first hybrid experiment H_0 behaves the real-world experiment, in particular returning the distribution $[\mathsf{EDB}, \{(K_{\mathbf{w}[i],0}, K_{\mathbf{w}[i],1})\}_{i \in [|\mathbf{w}|]}]$ to $\mathcal{A}$. In the second hybrid, the function F_K is replaced by a truly random function when running $\mathsf{EDBSetup}$ and when producing the tokens $(K_{\mathbf{w}[i],0}, K_{\mathbf{w}[i],1})$ given to $\mathcal{A}$, i.e., every search token is replaced with a truly-random key pair. It is easy to see that $\Pr[H_0 \Rightarrow 0] - \Pr[H_1 \Rightarrow 1] = \mathrm{negl}(\lambda)$ by the pseudorandomness of F.

For the next hybrid H_2, when running $\mathsf{EDBSetup}$, we are going to replace $F'_{K_{\mathbf{w}[i],0}}$ with an independent random function for every $i \in [|\mathbf{w}|]$. In particular, this means that every label l of a pair (l,d) added to L_j when processing the key-word w in $\mathsf{EDBSetup}$ is independent and uniform. Similarly to the above, $\Pr[H_1 \Rightarrow 0] - \Pr[H_2 \Rightarrow 1] = \mathrm{negl}(\lambda)$ by the pseudorandomness of F'.

Finally, in H_3, for all $i \in [|\mathbf{w}|]$, we replace every data-block d containing encryptions of identifiers in $\mathsf{DB}(\mathbf{w}[i])$ produced in $\mathsf{EDBSetup}$ with a randomly chosen string of the appropriate length. It is not hard to see that H_3 behaves exactly as $\text{SIM-SSE}^1{}_{\Pi,\mathcal{L}_{\min},\mathcal{A},\mathcal{S}}$, and moreover, $\Pr[H_2 \Rightarrow 0] - \Pr[H_3 \Rightarrow 1] = \mathrm{negl}(\lambda)$ by the pseudorandomness of $(\mathcal{E}, \mathcal{D})$. $\square$

Adaptive security. We additionally propose an efficient instantiation of the above scheme which is actively secure *in the random oracle model.* Note that, in this case, the security notion allows the simulator to *program* the random oracle.

Concretely, we instantiate $(\mathcal{E}, \mathcal{D})$ with the scheme encrypting M under secret key K as $\mathcal{E}_K(M) = R \,\|\, (H(K \,\|\, R) \oplus M)$, where R is a random λ-bit string, and H is a hash function with output length equal the message length, to be modeled as a random oracle in the proof. (As above, the total ciphertext length is denoted as s.) Moreover, we also instantiate F' using the same hash function H, letting $F'(K_{w,0}, i) = H(K_{w,0} \| \langle i \rangle)$, where $\langle i \rangle$ is a binary encoding of the integer $i \in \mathbb{N}$.

In the proof, the simulator $\mathcal{S}$ handles the random oracle queries, setting $H(x)$ to a random value whenever handling a query on input $x \in \{0,1\}^*$. Moreover, when the attacker chooses an index DB, $\mathcal{S}$ is given $N = 2^t$ and for all $i = 0, 1, \ldots, t-1$, adds 2^{t-i} pairs (l,d) to the set L_i, where l is a random ℓ-bit label and d is a random $(2^i \cdot s)$-bit string. It then generates EDB as the concatenation of the hash table created from $L_0, \ldots, L_{t-1}$, and hands EDB over to the attacker. (Still, $\mathcal{S}$ keeps $L_0, L_1, \ldots, L_{t-1}$ as its state.)

Later, upon each query w from the attacker, the simulator learns $\mathsf{DB}(w) = \{\mathrm{id}_1, \ldots, \mathrm{id}_{n_w}\}$, where $n_w = \sum_{i=0}^{t-1} n_{w,i} \cdot 2^i$. In this case, it generates random $K_{w,0}$ and $K_{w,1}$ as the corresponding token. Moreover, it sets $c = 0$, and for every $i = 0, 1, \ldots, t-1$, if $n_{w,i} = 1$, the simulator picks a random pair $(l,d) \in L_i$ where $d = R_1 \,\|\, C_1 \,\|\, \ldots \,\|\, R_{2^i} \,\|\, C_{2^i}$, removes (l,d) from L_i, and *programs* the random oracle so that

$$H(K_{w,0} \| \langle i \rangle) = l \;, \qquad H(K_{w,1} \| R_j) \oplus C_j = \mathrm{id}_{c+j} \text{ for all } j = 1, \ldots, 2^i \;,$$

and adds 2^i to c. If the programming cannot succeed (because the corresponding values are already set for H), the simulator aborts.

We omit a formal analysis that the above is a good simulation strategy, as it follows from standard techniques. Overall, we obtain the following theorem.

Theorem 11 (Adaptive Security of Π.). *The above hash-based instantiation of the SSE-scheme Π is $\mathcal{L}_{\min}$-SIM-secure in the random oracle model if F is a pseudorandom function.*

Acknowledgments Part of this work was done when the second author was with MIT CSAIL, partially supported by NSF Contract CCF-1018064. Moreover, this material is based on research sponsored by DARPA under agreement number FA8750-11-2-0225. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

References

1. M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, and H. Shi. Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions. In V. Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 205–222. Springer, Aug. 2005.
2. M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, and H. Shi. Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions. *Journal of Cryptology*, 21(3):350–391, July 2008.
3. R. D. Barve, E. A. M. Shriver, P. B. Gibbons, B. Hillyer, Y. Matias, and J. S. Vitter. Modeling and optimizing i/o throughput of multiple disks on a bus. In *SIGMETRICS*, pages 83–92, 1999.
4. M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In A. Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 535–552. Springer, Aug. 2007.
5. M. Bellare, M. Fischlin, A. O'Neill, and T. Ristenpart. Deterministic encryption: Definitional equivalences and constructions without random oracles. In D. Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 360–378. Springer, Aug. 2008.
6. A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill. Order-preserving symmetric encryption. In A. Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 224–241. Springer, Apr. 2009.
7. A. Boldyreva, N. Chenette, and A. O'Neill. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In *CRYPTO 2011*, LNCS, pages 578–595. Springer, Aug. 2011.
8. A. Boldyreva, S. Fehr, and A. O'Neill. On notions of security for deterministic encryption, and efficient constructions without random oracles. In D. Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 335–359. Springer, Aug. 2008.

9. D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In C. Cachin and J. Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 506–522. Springer, May 2004.
10. D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. Crypto 2013, Mar. 2013. `http://eprint.iacr.org/2013/169`.
11. Y.-C. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In J. Ioannidis, A. Keromytis, and M. Yung, editors, *ACNS 05*, volume 3531 of *LNCS*, pages 442–455. Springer, June 2005.
12. M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *ASIACRYPT 2010*, LNCS, pages 577–594. Springer, Dec. 2010.
13. R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In A. Juels, R. N. Wright, and S. Vimercati, editors, *ACM CCS 06*, pages 79–88. ACM Press, Oct. / Nov. 2006.
14. C. Gentry. Fully homomorphic encryption using ideal lattices. In M. Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.
15. E.-J. Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2003. `http://eprint.iacr.org/`.
16. O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM*, 43(3):431–473, 1996.
17. P. Golle, J. Staddon, and B. R. Waters. Secure conjunctive keyword search over encrypted data. In M. Jakobsson, M. Yung, and J. Zhou, editors, *ACNS 04*, volume 3089 of *LNCS*, pages 31–45. Springer, June 2004.
18. S. Kamara and C. Papamanthou. Parallel and dynamic searchable symmetric encryption. In *FC 2013*. Springer, 2013.
19. S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *ACM CCS 12*, pages 965–976. ACM Press, 2012.
20. K. Kurosawa and Y. Ohtaki. UC-secure searchable symmetric encryption. In *FC 2012*, LNCS, pages 285–298. Springer, 2012.
21. R. Pagh and F. F. Rodler. Cuckoo hashing. *J. Algorithms*, 51(2):122–144, 2004.
22. R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *SOSP*, pages 85–100, 2011.
23. C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *IEEE Computer*, 27(3):17–28, 1994.
24. D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy*, pages 44–55. IEEE Computer Society Press, May 2000.
25. P. van Liesdonk, S. Sedhi, J. Doumen, P. H. Hartel, and W. Jonker. Computationally efficient searchable symmetric encryption. In *Proc. Workshop on Secure Data Management (SDM)*, pages 87–100, 2010.
26. J. S. Vitter. Algorithms and data structures for external memory. *Foundations and Trends in Theoretical Computer Science*, 2(4):305–474, 2006.