Searchable Symmetric Encryption: Optimal Locality in Linear Space via Two-Dimensional Balanced Allocations

Gilad Asharov* Moni Naor† Gil Segev‡ Ido Shahaf‡

Abstract

Searchable symmetric encryption (SSE) enables a client to store a database on an untrusted server while supporting keyword search in a secure manner. Despite the rapidly increasing interest in SSE technology, experiments indicate that the performance of the known schemes scales badly to large databases. Somewhat surprisingly, this is not due to their usage of cryptographic tools, but rather due to their poor *locality* (where locality is defined as the number of *non-contiguous* memory locations the server accesses with each query). The only known schemes that do not suffer from poor locality suffer either from an impractical *space overhead* or from an impractical *read efficiency* (where read efficiency is defined as the ratio between the number of bits the server reads with each query and the actual size of the answer).

We construct the first SSE schemes that simultaneously enjoy optimal locality, optimal space overhead, and nearly-optimal read efficiency. Specifically, for a database of size N, under the modest assumption that no keyword appears in more than $N^{1-1/\log\log N}$ documents, we construct a scheme with read efficiency $\tilde{O}(\log\log N)$. This essentially matches the lower bound of Cash and Tessaro (EUROCRYPT '14) showing that any SSE scheme must be sub-optimal in either its locality, its space overhead, or its read efficiency. In addition, even without making any assumptions on the structure of the database, we construct a scheme with read efficiency $\tilde{O}(\log N)$.

Our schemes are obtained via a *two-dimensional* generalization of the classic balanced allocations ("balls and bins") problem that we put forward. We construct nearly-optimal two-dimensional balanced allocation schemes, and then combine their algorithmic structure with subtle cryptographic techniques.

*IBM Research. Email: gsasharo@us.ibm.com. This work was completed while the author was a post-doctoral researcher at the Hebrew University's School of Computer Science and Engineering.

†Incumbent of the Judith Kleeman Professorial Chair, Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, Israel. Email: moni.naor@weizmann.ac.il. Research supported in part by the Israel Science Foundation and by the Israeli Centers of Research Excellence (I-CORE) Program (Center No. 4/11).

‡School of Computer Science and Engineering, Hebrew University of Jerusalem, Jerusalem 91904, Israel. Email: {segev,ido_shahaf}@cs.huji.ac.il. Supported by the Israel Science Foundation (Grant No. 483/13) and by the Israeli Centers of Research Excellence (I-CORE) Program (Center No. 4/11).

Contents

1 Introduction

Outsourcing data storage to remote servers is an extremely useful technology that has been adopted both by large organizations and by individual users. It offers great benefits, but at the same time, raises various concerns when dealing with sensitive data. In particular, in order to preserve the confidentially of the data against an untrusted server, user-side symmetric encryption methods are typically employed prior to storing the data. As a result, operations as basic as keyword search become extremely expensive and sometimes even infeasible. This problem has motivated the cryptographic community to develop encryption methods that enable to search over symmetrically-encrypted data while not revealing sensitive information.

A searchable symmetric encryption (SSE) scheme [SWP00, CGK$^+$06] is a mechanism that allows a client to store data on an untrusted server and later perform keyword searches: Given a keyword w, the client should be able to retrieve all documents that contain w. First, the client encrypts its database and uploads it to the server. The client can then repeatedly query the server with various keywords. Informally, the security requirement asks that the server does not learn any information about keywords for which the client did not issue any queries.

A very productive line of research (see, for example, [SWP00, Goh03, CM05, CGK$^+$06, CK10, vLSD$^+$10, CGK$^+$11, KO12, KPR12, CJJ$^+$13, KO13, KP13, CJJ$^+$14, CT14, CGP$^+$15] and the references therein) has been devoted to the construction of searchable symmetric encryption schemes. However, implementations and experiments with real-world databases [CJJ$^+$13] indicate that the performance of the known schemes is quite disappointing and scales badly to large databases. Somewhat surprisingly, it turns out that the main bottleneck is in fact not the cryptographic processing of the data, but rather lower-level issues resulting from the inefficient memory layouts required by these schemes.

Specifically, the common drawback in the known schemes is poor *locality*: the server has to access a rather large number of *non-contiguous* memory locations with each query. The only known schemes that do not suffer from poor locality suffer either from an impractical *space overhead* (i.e., their encrypted databases are much larger than the original databases), or from an impractical *read efficiency* (i.e., they read much more data than needed for answering each query).[1]

This state of affairs naturally poses the challenge of constructing a searchable symmetric encryption scheme that simultaneously enjoys optimal space overhead, locality, and read efficiency. However, Cash and Tessaro [CT14] elegantly proved that obtaining such optimal efficiency guarantees is in fact impossible: any scheme must be sub-optimal in either its space overhead, its locality, or its read efficiency. Cash and Tessaro also presented a scheme that significantly improved the trade-off between the space overhead, locality, and read efficiency of the previously known schemes. However, although their scheme offers optimal read efficiency, it is far from optimal in both its space overhead and locality (see Table 1). This leads to the following fundamental problem, introducing a subtle combination of cryptographic and algorithmic challenges:

Can we construct a searchable symmetric encryption scheme that simultaneously enjoys nearly-optimal space overhead, locality, and read efficiency?

[1] In this work we rely on the notions of locality and read efficiency as formalized by Cash and Tessaro [CT14]. Specifically, the locality of a scheme is the maximal number of non-contiguous memory accesses that the server performs with each query, and the read efficiency of a scheme is the ratio between the number of bits the server reads with each query and the actual size of the answer. We refer the reader to Section 2.1 for the formal definitions.

1.1 Our Contributions

We construct searchable symmetric encryption schemes that simultaneously enjoy nearly-optimal space overhead, locality, and read efficiency. Our main results are obtained by basing searchable symmetric encryption on a *two-dimensional* generalization of the classic balanced allocations problem (also known as the "balls and bins" problem) that we put forward. We construct efficient two-dimensional balanced allocation schemes, and by combining them with subtle cryptographic techniques we obtain significant improvements over the previously known searchable symmetric encryption schemes.

Following the line of research on searchable symmetric encryption, we assume that a database is represented as a collection $\mathsf{DB} = \{\mathsf{DB}(w_1), \ldots, \mathsf{DB}(w_{n_\mathsf{W}})\}$, where $w_1, \ldots, w_{n_\mathsf{W}}$ are distinct keywords, and $\mathsf{DB}(w)$ is the list of all documents (or document identifiers) that contain the keyword w. We denote by $N = \sum_{i=1}^{n_\mathsf{W}} |\mathsf{DB}(w_i)|$ the size of the database, and for each keyword w we denote by $n_w = |\mathsf{DB}(w)|$ the number of documents containing w. We consider the standard unit-cost word-RAM model, and assume for simplicity that keywords and document identifiers are represented using a constant number of machine words (thus we measure space usage in machine words)[2].

We construct three searchable symmetric encryption schemes whose efficiency guarantees are summarized in Theorem 1.1 and in Table 1.[3] Whereas our first and second schemes are based on the above mentioned two-dimensional balanced allocations problem, our third scheme follows a different approach. For our third scheme we show that the recent scheme of Cash and Tessaro [CT14] can be modified to have optimal locality without hurting its space overhead or read efficiency.

Theorem 1.1. *Assuming the existence of one-way functions, there exist searchable symmetric encryption schemes offering the following guarantees for databases of size N:*

1. *Space $O(N)$, locality $O(1)$, and read efficiency $\tilde{O}(\log N)$ without any assumptions on the structure of the database.*
2. *Space $O(N)$, locality $O(1)$, and read efficiency $\tilde{O}(\log \log N)$ assuming that no keyword appears in more than $N^{1-\frac{1}{\log \log N}}$ documents.*
3. *Space $O(N \log N)$, locality $O(1)$, and read efficiency $O(1)$ without any assumptions on the structure of the database.*

In Table 1 we compare the efficiency guarantees of our schemes to those of the previously known schemes that have less than a polynomial space overhead[4]. Compared to those schemes, our first and second schemes show that optimal space and locality can be achieved together with a nearly-optimal read efficiency. Our third scheme shows that by slightly increasing the space overhead, it is possible to achieve optimal locality and read efficiency.

1.2 Related Work

Searchable symmetric encryption. The notion of searchable symmetric encryption was put forward by Song, Wagner and Perrig [SWP00] who suggested several practical constructions. Formal notions of security and constructions satisfying them were later provided by Curtmola, Garay, Kamara, and Ostrovsky [CGK$^+$06, CGK$^+$11]. Additional work in this line of research developed

[2]The unit cost word-RAM model is considered the standard model for analyzing the efficiency of data structures (see, for example, [DP08, Hag98, HMP01, Mil99, PP08] and the references therein).

[3]Throughout this paper, for any function $f(n)$ we let $\tilde{O}(f(n))$ denote $O(f(n)(\log f(n))^c)$ for some constant c.

[4]There are various schemes (e.g. [CGK$^+$06, CK10, vLSD$^+$10, KO12, KP13]) that have a *polynomial* space overhead, and offer constant locality and read efficiency. However, for concreteness we compare our schemes only to those with a potentially practical space overhead.

Scheme	Space	Locality	Read Efficiency
[CGK+06, KPR12, CJJ+13]	$O(N)$	$O(n_w)$	$O(1)$
[CT14]	$O(N \log N)$	$O(\log N)$	$O(1)$
This work I	$O(N)$	$O(1)$	$\tilde{O}(\log N)$
This work II	$O(N)$	$O(1)$	$\tilde{O}(\log \log N)$
This work III	$O(N \log N)$	$O(1)$	$O(1)$
Lower bound [CT14]	$\omega(N)$	$O(1)$	$O(1)$

Table 1: The efficiency guarantees of our schemes and of the previously known schemes that have less than a polynomial space overhead. Recall that we denote by N the size of the underlying database, and by n_w the number of documents that contain each keyword w (note that n_w may be as large as N). The efficiency of our second scheme is based on the modest assumption that no keyword appears in more than $N^{1-1/\log\log N}$ documents.

searchable symmetric encryption schemes with various efficiency properties [Goh03, CM05, CK10, vLSD+10, CT14], support for data updates [KPR12, KP13, CJJ+14], authenticity [KO13], and support for more advanced searches [CJJ+13].

The known constructions can be roughly divided into two, somewhat orthogonal, underlying approaches. The first approach (see, for example, [CGK+06, KPR12, CJJ+13] and the references therein) provides schemes with linear space and constant read efficiency, but with very poor locality. Given a database $\mathsf{DB} = \{\mathsf{DB}(w_1), \ldots, \mathsf{DB}(w_{n_\mathsf{W}})\}$ of size N, the idea underlying this approach is to allocate an array of size roughly N, and to uniformly map the N elements of the database into the N entries of the array (one element per entry). For efficiently recovering a list $\mathsf{DB}(w)$ given a keyword w (i.e., recovering the list of documents that contain w), each document identifier in the list $\mathsf{DB}(w)$ is stored in the array together with a pointer to the next document identifier in the list. Thus, since the elements are uniformly mapped into the array, recovering $\mathsf{DB}(w)$ requires the server to access essentially $n_w = |\mathsf{DB}(w)|$ random locations in the array.

The second approach (see, for example, [CGK+06, CK10, vLSD+10, CGK+11, KO13, KP13] and the references therein) provides schemes with optimal locality and read efficiency, but with a significant space overhead. Given a database $\mathsf{DB} = \{\mathsf{DB}(w_1), \ldots, \mathsf{DB}(w_{n_\mathsf{W}})\}$ of size N, the idea underlying this approach is to allocate a sufficiently large array, and to uniformly map each list $\mathsf{DB}(w)$ into a contiguous interval of length $n_w = |\mathsf{DB}(w)|$ in the array, without any overlaps between different lists. For efficiently recovering a list $\mathsf{DB}(w)$ given a keyword w, the server needs to access only a single random location, and then read n_w consecutive entries (thus resulting in optimal locality and read efficiency). However, since the locations of the lists in the array reveal information of the structure of the underlying database, some padding must be applied for hiding information on the lengths of the lists (e.g., padding each list according to the length of the longest list), thus resulting in a *polynomial* space overhead. Very recently, Cash and Tessaro showed that the space overhead can in fact be reduced from polynomial to logarithmic, at the cost of increasing the locality from constant to logarithmic (see Table 1).

As mentioned above, the work of Cash and Tessaro [CT14] explored the trade-off between locality, space overhead, and read efficiency. More specifically, instead of considering read efficiency directly, they considered the measure of *overlapping reads*, where a scheme has α-overlapping reads if reads for a new search overlaps in at most α bits with all previous reads. As noted by Cash and Tessaro, a (somewhat weak) read efficiency requirement is implicit in the condition on overlapping reads,

and they proved a super-linear space lower bound that ties together locality and overlapping reads (see [CT14] for further details).

Balanced allocations. Our two-dimensional balanced allocations problem is a novel generalization of the classical balanced allocations problem (the "balls and bins" problem), and may be of independent interest. The classical balanced allocations problem has seen a large number of generalizations over the years, and we refer the reader to [MU05, MRS00] for an overview of this problem and of some of its generalizations. A multi-dimensional generalization of this problem was first introduced by Broder and Mitzenmacher [BM05]. They considered that task of throwing balls into bins, where each ball is a random D-dimensional 0-1 vector of weight f (i.e., each vector has exactly f non-zero entries, and is chosen uniformly from all $\binom{D}{f}$ such vectors). Broder and Mitzenmacher were interested in the average load in each dimension for each bin. The multi-dimensional generalization that we consider, as we discuss in Section 1.3, seems completely different.

1.3 Overview of Our Approach

The problem of search on encrypted data offers an interesting challenge in the interface between data structure design and cryptography. We provide an overview of the main ideas underlying our schemes and in order to emphasize the news ones, we focus here on the first and second schemes, as these are based on a completely new approach for constructing searchable symmetric encryption schemes. Our third scheme follows a more standard approach (based on that of Cash and Tessaro [CT14]), and the reader is referred to Section 5 for the main ideas underlying that scheme.

As discussed in Section 1.2, the known schemes can be roughly divided into two, somewhat orthogonal, underlying approaches. The first approach provides schemes with linear space and constant read efficiency, but with very poor locality. The second approach provides schemes with optimal locality and read efficiency, but with a significant space overhead. Our approach can be viewed as a subtle mixture of these two approaches that enables us to enjoy their advantages while mitigating their disadvantages.

Specifically, given a database $\mathsf{DB} = \{\mathsf{DB}(w_1), \ldots, \mathsf{DB}(w_{n_\mathsf{W}})\}$, recall that the first approach completely ignores the structure of the lists, and places each document in a random location (leading to locality $n_w = |\mathsf{DB}(w)|$). The second approach preserves the structure of the lists and places the documents of each list in consecutive locations (leading to optimal locality and read efficiency, but with an impractical space overhead). Our approach preserves the structure of the lists, but does not place documents from the same list in consecutive locations. Instead, we make sure that documents from the same list are placed sufficiently near each other, yet in a random manner allowing documents from other lists to be placed between them. This flexibility enables us to enjoy constant locality in linear space, while only slightly affecting the read efficiency.

In what follows we begin by describing a *one-choice* scheme that follows this approach, and then describe a more efficient *two-choice* scheme. Throughout the exposition, we focus both on the algorithmic aspect of our schemes, introducing the *two-dimensional balanced allocations* problem, and on their cryptographic aspect.

Warm-up: A one-choice scheme. Given a database $\mathsf{DB} = \{\mathsf{DB}(w_1), \ldots, \mathsf{DB}(w_{n_\mathsf{W}})\}$ of size N, our one-choice scheme allocates an array of "bins", and constructs an encrypted database as follows: For each keyword w we compute a hash value $h(w)$, and place the ith document from the list $\mathsf{DB}(w)$ in the bin $h(w) + i - 1$. That is, the first document containing w is placed in the bin $h(w)$, the second document containing w is placed in the bin $h(w) + 1$, and so on.

Once this process is completed for all keywords, note that each bin may contain more than a single document. Specifically, each bin may contain several documents corresponding to different keywords. This can naturally be viewed as a two-dimensional generalization of the classic "balls and bins" problem, where one dimension consists of the keywords and the additional dimension consists of the structure of their lists (i.e., the lengths of their lists).

Equipped with this view of the process, we are interested in upper bounding its maximal load (that is, the maximal number of documents contained in each bin). However, this random process introduces new challenges when compared to its one-dimensional variant: The locations of the documents in the array are *not* independent. Specifically, given that a certain bin has many documents, it is rather likely that its adjacent bins also have many documents. Nevertheless, by carefully analyzing the dependencies that the lists introduce, we are able to bound its maximal load. Specifically, we show that for any database of size N, if we allocate an array of $N/\tilde{O}(\log N)$ bins each of size $\tilde{O}(\log N)$ (we make sure that the overall space is linear in N), then with all but a negligible probability there are no overflowing bins.

This enables us to transform this algorithmic template into a searchable encryption scheme as follows. First, we pad each bin to contain exactly $\tilde{O}(\log N)$ documents by adding "dummy" documents when needed, and we uniformly shuffle the documents in each bin. Then, the documents in each list $\mathsf{DB}(w)$ are each encrypted using an encryption key that is derived from the keyword w (say, by using a pseudorandom function that also generates the hash value $h(w)$). We use an encryption scheme that produces *pseudorandom ciphertexts*, and has an *elusive and verifiable range* (see Section 2.2).

In terms of functionality, by providing the server with the value $h(w)$ and with the decryption key corresponding to w, the server can recover the entire list $\mathsf{DB}(w)$ as follows. The server goes to the bin $h(w)$, and tries decrypting all of the ciphertexts that are stored there. Since we use an encryption scheme with an elusive and verifiable range, the server will be successful in decrypting exactly one of these ciphertexts, and this is the first document that contains w. The server then proceeds to the bin $h(w) + 1$ and so on until it reaches a bin in which no ciphertext decrypts.

In terms of efficiency, the locality is optimal: The server goes to the bin $h(w)$, and from that point it only uses *contiguous* memory access. In addition, the read efficiency is $\tilde{O}(\log N)$, since for retrieving each document the server reads a bin containing $\tilde{O}(\log N)$ encrypted documents. This yields a scheme with space $O(N)$, locality $O(1)$, and read efficiency $\tilde{O}(\log N)$.

Finally, the security of the scheme is based on the observation that the choices of bins for documents from different lists are statistically-close to being completely independent. Therefore, the access pattern that the server sees (i.e., the values $h(w)$ for different keywords) does not reveal any unnecessary information on the database.

An exponential improvement: A two-choice scheme. In the classic balanced allocations problem it is well known that the "two-choice paradigm" [ABK+99, MU05] exponentially improves the maximal load for various ranges of the parameters[5]. This motivates us to consider a two-choice generalization of the above-described one-choice scheme.

Given a database $\mathsf{DB} = \{\mathsf{DB}(w_1), \ldots, \mathsf{DB}(w_{n_\mathsf{W}})\}$ of size N, our two-choice scheme allocates an array of bins and constructs an encrypted database as follows. For each keyword w we compute two independent hash values, $h_1(w)$ and $h_2(w)$, and place all documents containing w in consecutive bins (as in our one-choice scheme) starting either from the bin $h_1(w)$ or from the bin $h_2(w)$ (but

[5]Roughly speaking, when throwing about N balls into N bins by placing each ball in the currently least loaded bin out of two possible choices, the maximal load is $O(\log \log N)$ with high probability (as opposed to a maximal load of $\Omega(\log N / \log \log N)$ in the one-choice process).

not from both). We choose between $h_1(w)$ and $h_2(w)$ as the starting point based on the maximal load of the bins in the relevant range (i.e., we choose the one with the smaller maximal load at the time of insertion)[6].

This two-choice variant of our two-dimensional balanced allocations problem turns to be significantly more challenging to analyze (even when compared to the two-choice *one-dimensional* problem). Nevertheless, we are able to show that under the modest assumption that no keyword appears in more than $N^{1-1/\log\log N}$ documents, we indeed obtain an exponential improvement: if we allocate an array of $N/\tilde{O}(\log\log N)$ bins each of size $\tilde{O}(\log\log N)$ (we again make sure that the overall space is linear in N), then with all but a negligible probability there are no overflowing bins. Moreover, we show that the assumption that no keyword appears in more than $N^{1-1/\log\log N}$ documents is in fact essential for obtaining such an exponential improvement.

Our analysis builds upon and generalizes the layered induction technique of Azar et al. [ABK+99]. Since our scheme places all elements of a list according to two random choices that are made for the *entire* list (and not for each element), this process introduces many dependencies between elements, as well as various dependencies between, say, the loads of any two consecutive bins. Such dependencies make the process significantly more challenging to analyze.

We once again to transform this algorithmic template into a searchable encryption scheme. First, we pad each bin to contain exactly $\tilde{O}(\log\log N)$ documents by adding "dummy" documents when needed, and we uniformly shuffle the documents in each bin. Then, the documents in each list $\mathsf{DB}(w)$ are each encrypted using an encryption key that is derived from the keyword w (by using a pseudorandom function as above). As in our one-choice scheme, we use an encryption scheme that produces *pseudorandom ciphertexts*, and has an *elusive and verifiable range*.

In terms of functionality, note that by providing the server with the values $h_1(w)$ and $h_2(w)$, as well as with the number n_w of documents that contain the keyword w, the server can simply send back to the client the content of the n_w consecutive bins starting from the location $h_1(w)$ and the content of the n_w consecutive bins starting from the location $h_2(w)$. The client can then decrypt the content of these bins and recover the list $\mathsf{DB}(w)$ using a decryption key that is derived from the keyword w. We note that we use an additional linear-space hash table for enabling the server to recover the value n_w on its own, and we emphasize that in this scheme we do not allow the server to decrypt the content of the bins on its own. This is due to the fact that whether the list $\mathsf{DB}(w)$ is stored starting from $h_1(w)$ or $h_2(w)$ may reveal unnecessary information on the structure of the database.

In terms of efficiency, as in our one-choice scheme we obtain optimal locality in linear space, but here we improve the read efficiency from $\tilde{O}(\log N)$ to $\tilde{O}(\log\log N)$: for retrieving each document the server reads the content of two bins, each of which contains $\tilde{O}(\log\log N)$ encrypted documents.

Finally, the security of the scheme is based on the observation that the two possible starting locations of documents from different lists in the database are statistically-close to being completely independent. Their actual starting locations are far from being independent, but this information is not revealed to the server since it cannot decrypt the content of the bins. Therefore, the access pattern that the server sees (i.e., the values $h_1(w)$ and $h_2(w)$ for various keywords w) does not reveal any unnecessary information on the database.

1.4 Paper Organization

The remainder of this paper is organized as follows. In Section 2 we introduce the formal definition of symmetric searchable encryption schemes, as well as present various tools and tail bounds that

[6] In fact, our actual allocation rule is a bit more subtle, and we refer the reader to Section 3.2 for the specific details.

are used in our constructions and proofs. Then, in Section 3 we put forward the two-dimensional balanced allocation problem, and then present and analyze a one-choice allocation scheme and a two-choice allocation scheme. In Section 4 we present a unified framework for constructing searchable symmetric encryption schemes from two-dimensional allocation scheme, obtaining our first and second schemes as specific instantiations. Finally, in Section 5 we present our third scheme, improving the scheme of Cash and Tessaro [CT14].

2 Preliminaries

In this section we present the notation and basic definitions that are used in this work. We denote by $\lambda \in \mathbb{N}$ the security parameter. For a distribution X we denote by $x \leftarrow X$ the process of sampling a value x from the distribution X. Similarly, for a set $\mathcal{X}$ we denote by $x \leftarrow \mathcal{X}$ the process of sampling a value x from the uniform distribution over $\mathcal{X}$. For an integer $n \in \mathbb{N}$ we denote by $[n]$ the set $\{1, \ldots, n\}$. A function $\mathsf{negl} : \mathbb{N} \to \mathbb{R}^+$ is $\mathsf{negligible}$ if for every constant $c > 0$ there exists an integer N_c such that $\mathsf{negl}(n) < n^{-c}$ for all $n > N_c$. All logarithms in this paper are to the base of 2.

2.1 Searchable Symmetric Encryption

Let λ denote the security parameter. We let $\mathsf{W} = \{w_1, \ldots, w_{n_\mathsf{W}}\}$ denote the set of keywords, where n_W is polynomial in λ, and we assume that each keyword w_i can be represented using a constant number of machine words, each of length $O(\lambda)$ bits, in the unit-cost RAM model. For each keyword w_i, we associate a list $\mathsf{DB}(w_i) = \{\mathsf{id}_1, \ldots, \mathsf{id}_{n_i}\}$ of documents (or document identifiers) in which the keyword w_i appears. We assume that each identifier is of length $O(\lambda)$ bits. We let $\mathsf{DB} = \{\mathsf{DB}(w_1), \ldots, \mathsf{DB}(w_{n_\mathsf{W}})\}$, and let $N = \sum_{i=1}^{n_\mathsf{W}} n_i$ to denote the total number of keyword/identifier pairs in DB. Finally, let n_id denote the number of unique identifiers.

Searchable symmetric encryption. There are various different syntaxes for SSE schemes in the literature, where the main differences are the complexity of the interaction between the server and the client with each query. While some consider an interaction between the two that may consist of few rounds (where the server learns no information), others consider a single round interaction where the server also decrypts the data and sends the result to the client (and thus the server learn the output of each query). Usually, apart from the index of keywords/identifiers, the client uploads a sequence of encrypted documents, and the index is used to help the client locate the documents corresponding to its keyword.

Some of our schemes consist of two rounds of interaction, whereas some consist of a single round (and can thus be easily adapted to two rounds). In a one-round scheme, the client sends a token, the server performs some computation during which it learns the set of identifiers. It then fetches the corresponding (encrypted) documents from the database and sends them back to the client, which decrypts them and learns the result. In a two-round scheme, the server does not decrypt the set of identifiers on its own, and it sends the client some message, which the client resolves and learns the set of identifiers and send them back to the server. As in a one-round scheme, given the identifiers the server fetches the encrypted documents from the database and sends them back to the client.

In fact, two-rounds schemes can be viewed as having one round of interaction, if the server stores a database of keywords/documents instead of a database of keywords/identifiers. We note that the lower bound of [CT14] holds in both setting, since their lower bound relates only to the *structure* of the database. We proceed to the formal definitions. Although the following definition seems to relate to one-round schemes, it actually captures both options. This is because the second round is just the transmission of the identifiers, and fetching them from storage.

2.1.1 Functionality

A searchable symmetric encryption (SSE) scheme Π consists of algorithms (KeyGen, $\mathsf{EDBSetup}$, TokGen, Search, $\mathsf{Resolve}$) such that:

- $K \leftarrow \mathsf{KeyGen}(1^\lambda)$. The key generation algorithm KeyGen takes as input the security parameter 1^λ and outputs a secret key K.
- $\mathsf{EDB} \leftarrow \mathsf{EDBSetup}(K, \mathsf{DB})$. The database-setup $\mathsf{EDBSetup}$ algorithm takes as input a key K and a database DB and outputs an encrypted database EDB.
- $(\tau, \rho) = \mathsf{TokGen}(K, w)$. The token generator algorithm is a deterministic algorithm that takes as input the private key K and the keyword w and outputs a token τ to be sent to the server, and some internal state ρ for the resolving algorithm.
- $M = \mathsf{Search}(\mathsf{EDB}, \tau)$. The searching algorithm Search takes as input the token τ and the encrypted database EDB, and outputs some list M of results.
- $S = \mathsf{Resolve}(\rho, M)$. The resolving algorithm receives the list M and the state, and outputs the set of decrypted results L.

An SSE scheme for databases of size $N = N(\lambda)$ is correct if for every PPT adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that the output of the experiment $\mathsf{correct}_{\mathcal{A},\mathsf{DB}}(1^\lambda)$ is 1 with probability $1 - \mathsf{negl}(\lambda)$ for all sufficiently large $\lambda \in \mathbb{N}$, where the experiment is defined as follows:

1. The adversary $\mathcal{A}$ on input 1^λ outputs a database DB of size $N = N(\lambda)$ together with some state information state.
2. A key $K \leftarrow \mathsf{KeyGen}(1^\lambda)$ is chosen and the database is encrypted by computing $\mathsf{EDB} \leftarrow \mathsf{EDBSetup}(K, \mathsf{DB})$.
3. The adversary $\mathcal{A}$ is invoked on input $(\mathsf{EDB}, \mathsf{state})$, and can repeatedly issue queries w_i, where each query is answered as follows:
   (a) $(\tau_i, \rho_i) \leftarrow \mathsf{TokGen}(K, w_i)$, $M_i \leftarrow \mathsf{Search}(\tau_i, \mathsf{EDB})$ and $S_i = \mathsf{Resolve}(\rho_i, M_i)$.
4. The output of the experiment is 1 if and only if for every query w_i it holds that $S_i = \mathsf{DB}(w_i)$.

We note that in case where the database is an index of keywords/documents, then the above syntax is a one-round protocol. In case where the database is an index of keyword/identifiers, then this syntax corresponds to a two-round protocol, where the algorithms (TokGen, Search, $\mathsf{Resolve}$) describes the first round, and in which in the second round has the following fixed structure: The client sends S, gets back the corresponding documents and then decrypts them. The single-round syntax for keyword/identifiers index is a sub-case of the above, where there is no $\mathsf{Resolve}$ algorithm, no secret state ρ, and the algorithm Search simply outputs S (and this is known to the server). Recall that the server then fetches the documents from storage according to the identifiers, and the client decrypts the actual document.

2.1.2 Locality and Read Efficiency

Our notions of locality and read efficiency follow those introduced by Cash and Tessaro [CT14].

Locality. The search procedure of any SSE scheme can be decomposed into a sequence of contiguous reads from the encrypted database EDB. Specifically, assume that the Search algorithm does not get the encrypted database as input, but rather only gets oracle access to it. Each query to this oracle consists of some interval $[a_i, b_i]$, and the oracle replies with the words that are stored on those intervals of EDB. The Search algorithm is invoked on some token τ and queries its oracle on some

interval $[a_1, b_1]$. It continues to compute the next intervals to read based on τ and all previously read intervals from EDB. We denote these intervals by $\mathsf{ReadPat}(\mathsf{EDB}, \tau)$.

Definition 2.1 (Locality)**.** *An SSE scheme* Π *is* d-local *(or has locality* d*) if for every* λ*,* DB *and* $w \in \mathsf{W}$*,* $K \leftarrow \mathsf{KeyGen}(1^\lambda)$*,* $\mathsf{EDB} \leftarrow \mathsf{EDBSetup}(K, \mathsf{DB})$ *and* $\tau \leftarrow \mathsf{TokGen}(K, w)$ *we have that* $\mathsf{ReadPat}(\mathsf{EDB}, \tau)$ *consists of at most* d *intervals with probability* 1*.*

Read efficiency. The notion of locality alone is lacking, since the Search algorithm can read the whole database with a single interval. The notion of read efficiency measures the overall size of portion read by a search operation. For a given DB and w, we let $||\mathsf{DB}(w)||$ denote the number of words in the encoding of $\mathsf{DB}(w)$.

Definition 2.2 (Read efficiency)**.** *An SSE scheme* Π *is* r-read efficient *(or has read efficiency* r*) if for any* λ*,* DB*, and* $w \in \mathsf{W}$*, we have that* $\mathsf{ReadPat}(\tau, \mathsf{EDB})$ *consists of intervals of total length at most* $\mathsf{r} \cdot ||\mathsf{DB}(w)||$ *words.*

2.1.3 Simulation-Based Security

The standard security definition for SSE schemes follows the ideal/real simulation paradigm. We consider both static and adaptive security, where the difference is whether the adversary chooses its queries statically (i.e., before seeing any token), or in an adaptive manner (i.e., the next query may be a function of the previous tokens). In both cases, some information is leaked to the server, which is formalized by letting the simulator receive the evaluation of some "leakage function" on the database itself and the real tokens. We start with the static case.

The real execution. The real execution is parameterized by the scheme Π, the adversary $\mathcal{A}$, and the security parameter λ. In the real execution the adversary is invoked on 1^λ, and outputs a database DB and a list of queries $\mathbf{w} = \{w_i\}_i$. Then, the experiment invokes the key-generation algorithm and the database setup algorithms, $K \leftarrow \mathsf{KeyGen}(1^\lambda)$ and $\mathsf{EDB} \leftarrow \mathsf{EDBSetup}(K, \mathsf{DB})$. Then, for each query $\mathbf{w} = \{w_i\}_i$ that the adversary has outputted, the token generator algorithm is run to obtain $\tau_i = \mathsf{TokGen}(w_i)$. The adversary is given the encrypted database EDB and the resulting tokens $\boldsymbol{\tau} = \{\tau_i\}_{w_i \in \mathbf{w}}$, and outputs a bit b.

The ideal execution. The ideal execution is parameterized by the scheme Π, a leakage function $\mathcal{L}$, the adversary $\mathcal{A}$, a simulator $\mathcal{S}$ and the security parameter λ. In this execution, the adversary $\mathcal{A}$ is invoked on 1^λ, and outputs $(\mathsf{DB}, \mathbf{w})$ similarly to the real execution. However, this time the simulator $\mathcal{S}$ is given the evaluation of the leakage function on $(\mathsf{DB}, \mathbf{w})$ and should output $\mathsf{EDB}, \boldsymbol{\tau}$ (i.e., $(\mathsf{EDB}, \boldsymbol{\tau}) \leftarrow \mathcal{S}(\mathcal{L}(\mathsf{DB}, \mathbf{w}))$). The execution follows by giving $(\mathsf{EDB}, \boldsymbol{\tau})$ to the adversary $\mathcal{A}$, which outputs a bit b.

Let $\text{SSE-}\textsc{Real}_{\Pi,\mathcal{A}}(\lambda)$ denote the output of the real execution, and let $\text{SSE-}\textsc{Ideal}_{\Pi,\mathcal{L},\mathcal{A},\mathcal{S}}(\lambda)$ denote the output of the ideal execution, with the adversary $\mathcal{A}$, simulator $\mathcal{S}$ and leakage function $\mathcal{L}$. We now ready to define security of SSE:

Definition 2.3 (static $\mathcal{L}$-secure SSE)**.** *Let* $\Pi = (\mathsf{KeyGen}, \mathsf{EDBSetup}, \mathsf{TokGen}, \mathsf{Search})$ *be an SSE scheme and let* $\mathcal{L}$ *be a leakage function. We say that the scheme* Π *is* static $\mathcal{L}$-secure searchable encryption *if for every* PPT *adversary* $\mathcal{A}$*, there exists a* PPT *simulator* $\mathcal{S}$ *and a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$|\Pr[\text{SSE-}\textsc{Real}_{\Pi,\mathcal{A}}(\lambda) = 1] - \Pr[\text{SSE-}\textsc{Ideal}_{\Pi,\mathcal{L},\mathcal{A},\mathcal{S}}(\lambda) = 1]| < \mathsf{negl}(\lambda)$$

Adaptive setting. In the adaptive setting, the adversary is not restricted to specifying all of its queries $\mathbf{w}$ in advance, but can instead choose its queries during the execution in an adaptive manner, depending on the encrypted database EDB and on the tokens that it sees. Let $\text{SSE-REAL}^{\mathsf{adapt}}_{\Pi,\mathcal{A}}(\lambda)$ denote the output of the real execution in this adaptive setting. In the ideal execution, the simulator $\mathcal{S}$ is now an interactive Turing machine, which interacts with the experiment by responding to queries. First, the simulator $\mathcal{S}$ is initially invoked on $\mathcal{L}(\mathsf{DB})$ and outputs EDB. Then, for every query w_i that $\mathcal{A}$ may output, the function $\mathcal{L}$ is invoked on DB and all previously queries $\{w_j\}_{j<i}$ and the new query w_i, outputs some new leakage $\ell(w_i)$ which is given to the simulator $\mathcal{S}$. The latter outputs some t_i, which is given back to $\mathcal{A}$, who may then issue a new query. At the end of the execution, $\mathcal{A}$ outputs a bit b. Let $\text{SSE-IDEAL}^{\mathsf{adapt}}_{\Pi,\mathcal{L},\mathcal{A},\mathcal{S}}(\lambda)$ be the output of the ideal execution. The adaptive security of SSE is defined as follows:

Definition 2.4 (adaptive $\mathcal{L}$-secure SSE)**.** *Let* $\Pi = (\mathsf{KeyGen}, \mathsf{EDBSetup}, \mathsf{TokGen}, \mathsf{Search})$ *be an SSE scheme and let* $\mathcal{L}$ *be a leakage function. We say that the scheme* Π *is* $\mathsf{adaptive}\ \mathcal{L}\text{-}\mathsf{secure}\ \mathsf{searchable}\ \mathsf{encryption}$ *if for every* PPT *adversary* $\mathcal{A}$*, there exists a* PPT *simulator* $\mathcal{S}$ *and a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\left|\Pr\left[\text{SSE-REAL}^{\mathsf{adapt}}_{\Pi,\mathcal{A}}(\lambda)=1\right]-\Pr\left[\text{SSE-IDEAL}^{\mathsf{adapt}}_{\Pi,\mathcal{L},\mathcal{A},\mathcal{S}}(\lambda)=1\right]\right|<\mathsf{negl}(\lambda)$$

The leakage function. Following the standard notions of security for SSE we consider the leakage function $\mathcal{L}_{\min}$ for one-round protocols and the leakage function $\mathcal{L}_{\mathsf{sizes}}$ for two-round protocols, where

$$\begin{aligned}\mathcal{L}_{\min}(\mathsf{DB},\mathbf{w}) &= \left(N,\{\mathsf{DB}(w)\}_{w\in\mathbf{w}}\right),\\ \mathcal{L}_{\mathsf{sizes}}(\mathsf{DB},\mathbf{w}) &= \left(N,\{|\mathsf{DB}(w)|\}_{w\in\mathbf{w}}\right),\end{aligned}$$

and $N=\sum_{w\in W}|\mathsf{DB}(w)|$ is the size of the database. That is, both functions return the size of the database, and the difference between them is that the function $\mathcal{L}_{\min}$ returns the actual documents that contain each keyword $w\in\mathbf{w}$ that the adversary has queried, whereas the function $\mathcal{L}_{\mathsf{sizes}}$ returns only the number of such documents.

The leakage functions in the adaptive setting are defined analogously. That is, for a database DB, a set of "previous" queries $\{w_j\}_{j<i}$, and a new query w_i, we define

$$\begin{aligned}\mathcal{L}^{\mathsf{adap}}_{\min}(\mathsf{DB},\{w_j\}_{j<i},w_i) &= \begin{cases} N & \text{if } (\{w_j\}_{j<i},w_i)=(\perp,\perp)\\ \mathsf{DB}(w_i) & \text{otherwise}\end{cases}\\ \mathcal{L}^{\mathsf{adap}}_{\mathsf{size}}(\mathsf{DB},\{w_j\}_{j<i},w_i) &= \begin{cases} N & \text{if } (\{w_j\}_{j<i},w_i)=(\perp,\perp)\\ |\mathsf{DB}(w_i)| & \text{otherwise}\end{cases}.\end{aligned}$$

2.2 Additional Tools

Pseudorandom encryption schemes with elusive and verifiable range. In our scheme, we will encrypt the identifiers of list with a different key, and we need to ensure that a decryption of some ciphertext is valid only when decrypting with the "correct" key. In addition, the client pads its lists using some random elements, and we want to ensure that by doing so it does not accidentally introduce valid ciphertexts for some keys, and that ciphertexts seems independent of their keys. We therefore use an encryption scheme with the following properties.

Definition 2.5. *Let* $(\mathsf{Enc},\mathsf{Dec})$ *be a private-key encryption scheme and denote the range of a key* $K\leftarrow\{0,1\}^\lambda$ *in the scheme by* $\mathsf{Range}(K)\stackrel{\text{def}}{=}\{\mathsf{Enc}_K(x)\}_{x\in\{0,1\}^n}$*. Then,*

1. *We say that* $(\mathsf{Enc}, \mathsf{Dec})$ *has an* elusive range *if for every* PPT *machine* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\Pr_{K \leftarrow \{0,1\}^\lambda}\left[\mathcal{A}(1^\lambda) \in \mathsf{Range}(K)\right] < \mathsf{negl}(\lambda)$$

2. *We say that* $(\mathsf{Enc}, \mathsf{Dec})$ *has an* efficiently verifiable range *if there exists a* PPT *machine* M *such that* $M(1^\lambda, K, c) = 1$ *if and only if* $c \in \mathsf{Range}(K)$.
By convention, for every $c \notin \mathsf{Range}(K)$*, we have that* $\mathsf{Dec}_K(c) = \perp$.
3. *We say that* $(\mathsf{Enc}, \mathsf{Dec})$ *has* pseudorandom ciphertexts *if for every* PPT *adversary* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that:*

$$\left|\Pr\left[\mathcal{A}^{\mathsf{Enc}_K(\cdot)}(1^\lambda) = 1\right] - \Pr\left[\mathcal{A}^{\mathcal{R}(\cdot)}(1^\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda)$$

where $\mathcal{R}$ *is a probabilistic oracle that given any input outputs a freshly-sampled uniform value of the appropriate length (i.e., as the output length of* $\mathsf{Enc}_K(\cdot)$*), the probability on the left is taken over the choice* $K \leftarrow \{0,1\}^\lambda$ *and the internal randomness of the algorithm* Enc*, and the probability on the right is taken over the randomness of the oracle* $\mathcal{R}$.

We omit the Gen algorithm since the scheme is a symmetric-key one and thus we can assume w.l.o.g. that the key is a uniform string (i.e., the randomness for Gen). We note that an encryption scheme that satisfy all these properties can be constructed from any pseudorandom function (and therefore from one-way function) $\mathsf{PRF}_K : \{0,1\}^m \to \{0,1\}^{m+\lambda}$ for $K \in \{0,1\}^\lambda$ as $\mathsf{Enc}_K(x) = \langle r, \mathsf{PRF}_K(r) \oplus x0^\lambda \rangle$, where $x \in \{0,1\}^m$, $r \leftarrow \{0,1\}^\lambda$, and $x0^m$ denotes the concatenation of x and 0^λ. This scheme satisfy the first two notions [LP09], and it is easy to see that it has also pseudorandom ciphertexts.

Static hash tables. In our schemes we rely on static hash tables (also known as static dictionaries). These are data structures that given a set S can support lookup operations in constant time in the standard unit-cost word-RAM model. Specifically, a static hash table consists of a pair of algorithms denoted $(\mathsf{HTSetup}, \mathsf{HTLookup})$. The algorithm $\mathsf{HTSetup}$ gets as input a set $S = \{(\ell_i, d_i)\}_{i=1}^k$ of pairs (ℓ_i, d_i) of strings, where $\ell_i \in \{0,1\}^s$ is the label and $d_i \in \{0,1\}^r$ is the data. The output of this algorithm is a hash table $\mathsf{HT}(S)$. The lookup algorithm $\mathsf{HTLookup}$ on input $(\mathsf{HT}(S), \ell)$ returns d if $(\ell, d) \in S$, and $\perp$ otherwise.

There exist many constructions of static hash tables that use linear space (i.e., $O(k(r+s))$ bits) and answer lookup queries by reading a constant number of contiguous s-bit blocks and r-bit blocks (see, for example, [PR04, ANS10], and the many references therein).

2.3 Some Standard Tail Bounds

Our proofs in this paper rely on the following standard tail-bound inequalities for sequences of random variables (see, for example, [MU05, DP09]).

Lemma 2.6 (Multiplicative Chernoff bound)**.** *Suppose* $X_1, \ldots, X_n$ *are independent random variables taking values in* $\{0,1\}$*. Let* X *denote their sum and let* $\mu = E[X]$ *denote the sum's expected value. Then for any* $\delta > 0$*,*

$$\Pr\left[X > (1+\delta)\mu\right] \leq \left(\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right)^\mu.$$

Lemma 2.7. *Suppose $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_n$ are random variables such that each Y_i takes values in $\{0,1\}$, $Y_i = Y_i(X_1, \ldots, X_i)$ and $\Pr[Y_i = 1 | X_1, \ldots, X_{i-1}] \leq p$ for some $p \in [0,1]$. Let $Z_1, \ldots, Z_n$ be independent Bernoulli trials with success probability p. Then for any non-negative coefficients $c_1, \ldots, c_n$ and a bound k,*

$$\Pr\left[\sum_{i=1}^{n} c_i Y_i > k\right] \leq \Pr\left[\sum_{i=1}^{n} c_i Z_i > k\right].$$

Lemma 2.7 can be proved using a standard coupling argument (i.e., defining a joint probability distribution where always $Y_i \leq Z_i$).

Claim 2.8 (Bernstein bound)**.** *Let $W_1, \ldots, W_n$ be independent zero-mean random variables. Suppose that $|W_j| \leq M$ for all j. Then, for any $t > 0$,*

$$\Pr\left[\sum_{i=1}^{n} W_i > t\right] \leq \exp\left(-\frac{\frac{1}{2}t^2}{\sum_j E\left[W_j^2\right] + \frac{1}{3}Mt}\right).$$

Corollary 2.9. *Given Lemma 2.7's conditions, denote $N = \sum_{i=1}^{n} c_i$ and $M = \max c_i$. Then,*

$$\Pr\left[\sum_{i=1}^{n} c_i Y_i > 2Np\right] \leq \exp\left(-\frac{3}{8}\frac{N}{M}p\right).$$

Proof. Let $Z_1, \ldots, Z_n$ be the i.i.d. variables mentioned in the statement of Lemma 2.7. Set $W_i = c_i(Z_i - p)$. Then $E[W_i] = 0$, $|W_i| \leq c_i \max(1-p, p) \leq c_i$ and $E[W_i^2] = p(1-p)c_i^2$. Applying Lemma 2.7 and the Bernstein bound, we get

$$\begin{aligned}
\Pr\left[\sum_{i=1}^{n} c_i Y_i > 2Np\right] &\leq \Pr\left[\sum_{i=1}^{n} c_i Z_i > 2Np\right] \\
&= \Pr\left[\sum_{i=1}^{n} W_i > Np\right] \\
&\leq \exp\left(-\frac{\frac{1}{2}N^2p^2}{p(1-p)\sum_{i=1}^{n} c_i^2 + \frac{1}{3}MNp}\right) \\
&\leq \exp\left(-\frac{\frac{1}{2}N^2p^2}{pMN + \frac{1}{3}MNp}\right) = \exp\left(-\frac{3}{8}\frac{N}{M}p\right),
\end{aligned}$$

where we used the fact that $\sum_{i=1}^{n} c_i^2 \leq \sum_{i=1}^{n} c_i M = NM$. ■

3 Two-Dimensional Balanced Allocations

In this section we analyze the maximal load of two schemes for the two-dimensional balanced allocation problem, as discussed in Section 1.3. Our first scheme is a one-choice process that is described and analyzed in Section 3.1, and our second scheme is a two-choice process that is described and analyzed in Section 3.2.

Recall that the input to the two-dimensional balanced allocation problem is a collection of lists, $L_1, \ldots, L_k$, where the ith list L_i is of length n_i. The goal in this problem is to allocate an array of bins and to place each element from these lists in a bin (where a bin may contain several elements), such that given an index i we can efficiently recover the entire list L_i. Motivated by our goal of searchable symmetric encryption, we would like to design allocation scheme with good space overhead, locality, and read efficiency (while ignoring any security properties for now).

3.1 A One-Choice Allocation Scheme

Our one-choice scheme is described as Algorithm 3.1. As discussed in Section 1.3, for each list L_i we choose a uniform bin, and then place the elements of L_i in consecutive bins starting from that bin (one element in each bin).

ALGORITHM 3.1 (Algorithm OneChoiceAllocation$(m, (n_1, \ldots, n_k))$).

- **Input:** Number of bins m, and a vector of integers $(n_1, \ldots, n_k)$ representing the length of the lists $L_1, \ldots, L_k$.
- **The algorithm:**
  1. Initialize m empty bins $B_0, \ldots, B_{m-1}$.
  2. For each list L_i where $i = 1, \ldots, k$:
     (a) Sample $\alpha \leftarrow \{0, 1, \ldots, m-1\}$.
     (b) For $j = 0, \ldots, n_i - 1$:
        i. Place the jth element of the list L_i in bin $B_{\alpha+j \bmod m}$.

We show that for an appropriate choice of parameters, the maximal load is very close to its expectation. This is similar to the one-dimensional problem that considers balls instead of lists, and the locations of all balls are independent. In our case, the locations of the elements from the same list are clearly related, and the loads of any two consecutive bins are strongly correlated. This requires a more subtle analysis, and we prove the following theorem:

Claim 3.2. *Fix any m, k, and $n_1, \ldots, n_k$. Let $n = \sum_{i=1}^{k} n_i$ and assume that $m \leq n$. Then, with probability at least $1 - m \cdot 2^{-n/m}$, at the end of Algorithm 3.1 the maximal load of any bin is at most $3n/m$.*

Proof. For simplicity, we first assume that there is no list with more than m elements. We later remove this assumption.

For $0 \leq j \leq m-1$, let X_j be a random variable denotes the load of bin B_j, and for every $i = 1, \ldots, k$, let $Y_j[i]$ be an indicator that gets 1 if and only if some element of the ith list fells into bin B_j. Note that $X_j = \sum_{i=0}^{k-1} Y_j[i]$. Moreover, for a fixed $j \in \{0, \ldots, m-1\}$, $i \in \{1, \ldots, k\}$, we have:

$$E(Y_j[i]) = \frac{n_i}{m} \ .$$

This holds since there is no list with size greater than m, and therefore there is an element of some list L in bin B_j if and only if, the chosen placement for the head of the list is one of the values $\{j, [(j-1) \bmod m], \ldots, [(j - n_i + 1) \bmod m]\}$.

This implies that:

$$E[X_j] = E\left[\sum_{i=0}^{k-1} Y_j[i]\right] = \sum_{i=0}^{k-1} E\left[Y_j[i]\right] = \frac{\sum_{i=0}^{k-1} n_i}{m} = \frac{n}{m} \ .$$

Although the random variables $X_0, \ldots, X_{m-1}$ are dependent (i.e., the loads of the bins), and even for every list the random variables $Y_0[i], \ldots, Y_{m-1}[i]$ are dependent (i.e., the bins where the elements of some single list are placed), for every fixed $j \in \{0, \ldots, m-1\}$ the random variables $Y_j[1], \ldots, Y_j[k]$ are *independent* (i.e., the choices of elements inside a given bin) . Also, they all taking values in $\{0, 1\}$. Therefore, we can apply Chernoff's bound (Lemma 2.6), and get:

$$\Pr\left[X_j \geq (1+\delta) E\left[X_j\right]\right] \leq \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{E[X_j]}$$

For $\delta \geq e-1$, we have that:

$$\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}} = \frac{1}{1+\delta} \cdot \left(\frac{e}{1+\delta}\right)^{\delta} \leq \frac{1}{1+\delta} \leq \frac{1}{2} \ ,$$

and thus

$$\Pr\left[X_i \geq 3 \cdot \frac{n}{m}\right] \leq 2^{-n/m} \ .$$

Using union-bound, we conclude that the probability that one of the bins has load greater than $3 \cdot n/m$ is at most $m \cdot 2^{-n/m}$.

For case where there exist lists with size greater than m, observe that for each such a list i, each bin will have at least $\lfloor n_i/m \rfloor$ element of that list, and exactly $[n_i \bmod m]$ bins will have an additional one element from that list. For each such a list and for each bin B_i, instead of considering the single random variable $Y_j[i]$, we define $\lfloor n_i/m \rfloor + 1$ random variables $Y_j^{(0)}[i], \ldots, Y_j^{(\lfloor n_i/m \rfloor)}[i]$, where $Y_j^{(\gamma)}[i]$ indicates whether at least γ elements of the ith list fell into bin B_j. The first $\lfloor n_i/m \rfloor$ variables are constant, and always get 1. The random variable $Y_j^{(\lfloor n_i/m \rfloor)}[i]$ is 1 with probability $[n_i \bmod m]/m$. The expected sum of these random variables is n_i/m, and therefore $E[X_j] = n/m$, as in the simplified case. Again, these variables are independent and get values in $\{0,1\}$, so we can apply the Chernoff bound and get exactly the same result. ■

When constructing our symmetric searchable encryption schemes, we are interested in having a negligible failure probability, and we derive the following corollary from Claim 3.2:

Corollary 3.3. *Fix any k, and $n_1, \ldots, n_k$. Let $n = \sum_{i=1}^{k} n_i$ and $m = \frac{n}{\log n \cdot \log\log n}$. Then, with probability at least $1 - n^{-\omega(1)}$, at the end of Algorithm 3.1 the maximal load of any bin is at most $3 \log n \cdot \log\log n$.*

3.2 A Two-Choice Allocation Scheme

Our next step is generalizing our one-choice allocation scheme into a two-choice one. Before describing our scheme, we first discuss several possible generalizations that seem very reasonable and natural, but turn out somewhat insufficient and do not provide the desired properties.

Consider an algorithm that for every list L_i chooses two possible bins, B_{α_1} and B_{α_2}, and places the elements of the list in a consecutive manner starting from the least loaded bin $B_{\alpha_{i^*}}$ among these two choices. However, this approach seems insufficient, since the fact that $B_{\alpha_{i^*}}$ is the least loaded bin among the two possible bins B_{α_1} and B_{α_2} does not imply that $B_{\alpha_{i^*}+j}$ is the least loaded bin among B_{α_1+j} and B_{α_2+j} for any $j > 1$. That is, the decision we make in order to place the head of the list does not necessarily apply for the placement of the following element of the list.

Another natural approach is the following. Similarly to the previous proposal, for each list we choose two possible locations, B_{α_1} and B_{α_2}, and place its first element in the least loaded bin among these two possibilities. However, unlike the previous proposal, we make a similar decision for each element of the list. Specifically, the jth element of the list is placed in the least loaded bin among B_{α_1+j-1} and B_{α_2+j-1}. Unfortunately, this algorithm seems somewhat difficult to analyze as it introduces many (redundant) dependencies between elements, and there are too many decisions that are made in the algorithm. A more elegant approach would be to have a single comparison for the placement of the entire list.

We now turn to describe our two-choice approach. Assume without loss of generality that the lengths of all lists are powers of two (otherwise we can pad each with dummy elements and thus increase the database size at most by a factor of 2), and that the number of bins is also a power of two. In our algorithm, we process the lists according to their length. We first sort the lists in a

descending order, and then, for a given list of size n_i, we view each block of n_i consecutive bins as a single possible "super bin". As a result, the possible locations for the head of the list is not m (the overall number of bins), but rather only m/n_i, the number of "super bins". Then, we choose two possible super bins, and place the entire list, in the least loaded super bin among these two possible choices. The actual placement is done element by element, in the respective bins that constitute this super bin (i.e., one element per "standard" bin).

This approach has the following advantages: First, the entire list is placed according to a *single* comparison, which significantly simplifies the analysis. Second, all the elements of each list are placed *at the exact same level*. That is, our algorithm has the following invariant: Whenever we place a list, all the bins that constitute a possible super bin have the exact same number of elements. This is mainly due to the fact that we place the lists in descending order, and in each iteration the super bins are split into smaller super bins that each matches the size of the current list. A formal description is presented as Algorithm 3.4.

ALGORITHM 3.4 (Algorithm $\mathsf{TwoChoiceAllocation}(m, (n_1, \ldots, n_k))$).

- **Input:** Number of bins m and a vector of integers $(n_1, \ldots, n_k)$ representing the length of the lists $L_1, \ldots, L_k$. We let $n = \sum_{i=1}^{k} n_i$, and assume for concreteness that m and the n_i's are powers of two, and that $m \geq n_1 \geq n_1 \geq \cdots \geq n_k$.
- **The algorithm:**
  1. Initialize m empty bins $B_0, \ldots, B_{m-1}$.
  2. For each list L_i where $i = 1, \ldots, k$:
     (a) Choose $\alpha_1, \alpha_2 \leftarrow \{0, \ldots, \frac{m}{n_i} - 1\}$ independently and uniformly at random. Consider the two super bins $\widetilde{B}_{\alpha_1} = (B_{n_i \cdot \alpha_1 + j})_{j=0}^{n_j - 1}$ and $\widetilde{B}_{\alpha_2} = (B_{n_i \cdot \alpha_2 + j})_{j=0}^{n_j - 1}$.
     (b) Let $\widetilde{B}_{\alpha}$ be the least loaded[7] among $\widetilde{B}_{\alpha_1}$ and $\widetilde{B}_{\alpha_2}$. We place the list L_i in the super bin $\widetilde{B}_{\alpha}$. That is, for every $j = 0, \ldots, n_i - 1$:
        i. Place the jth element of the list L_i in the bin $B_{n_i \cdot \alpha + j}$.

3.2.1 Bounding the Maximal Load

We prove the following theorem:

Theorem 3.5. *Assume that each n_i is a power of two and that $n_1 \geq \cdots \geq n_k$. Let $S \geq n_1$ be some bound on the sizes, and let m be the number of bins. Then*

1. *With probability $1 - n^{-\Omega(\log n)}$, there are at most $S \log^2 n$ elements at level greater than $\frac{4n}{m} + \log\log\frac{n}{S} + 2$.*
2. *Let $\epsilon > 0$, $S = n^{1-\epsilon}$, and assume that $m \geq \frac{n}{\log n}$. Then for any non-decreasing function $f(n)$ satisfying $f(n) = \Omega(\log\log n)$, $f(n) = O(\sqrt{\log n})$ and $f(2n) = O(f(n))$, the maximal load is at most $\frac{4n}{m} + O(\log \epsilon^{-1} \cdot f(n))$ with probability $1 - O(\log \epsilon^{-1}) \cdot n^{-\Omega(\epsilon \cdot f(n^{\epsilon}))}$.*

For our searchable symmetric encryption schemes, we will rely on the following corollary (say, with $A = 1$) that follows from Theorem 3.5:

Corollary 3.6. *For any constant $A \geq 1$, let $\epsilon = (\log\log n)^{-A}$ (and so the size of the maximal list is bounded by $n^{1-1/(\log\log n)^A}$) and $m = \frac{n}{(\log\log n)^A \cdot (\log\log\log n)^2}$. Then with overwhelming probability the maximal load is $O((\log\log n)^A \cdot (\log\log\log n)^2)$.*

[7] The load of a super bin is defined as the sum of loads of the "standard" bins that constitutes that super bin. According to the invariant of this algorithm, the load of these standard bins is equal, and therefore the sum is simply the number of standard bins that constitute the super bin times the load of each one of them.

Proof overview. The proof is rather subtle, and is inspired by the "layered induction" technique of the one-dimensional problem [ABK+99] (see also [MRS00, MU05]). The following overview is rather simplified, and as a result some of its notation is not identical to that of the formal proof below.

In order to bound the maximal load, we need to approximately bound the number of bins with j elements for every value of j. The proof defines a sequence of values $\beta_1, \beta_2, \ldots$ and shows that the number of bins with load at least j is bounded by β_j with high probability. The proof follows by induction, where in each step, we show that if the number of bins with load at least j is at most β_j, then with high probability, the number of bins with load at least $j+1$ is at most β_{j+1}. The values $\beta_1, \beta_2, \ldots$ are defined according to a recursive relation. Intuitively, the sequence $\beta_1, \beta_2, \ldots$ is monotonic decreasing, and we look for the first j for which $\beta_j < 1$ (i.e., there are no bins with load at least j). As we will see, this description is a bit simplified and we cannot actually use the recursive relation all the way to $\beta_j < 1$, since the induction does not hold unconditionally. Thus, we will have to stop at a larger β_j and conclude the proof using some additional arguments. We will elaborate on this point later on.

The recursive relation. According to our algorithm, when we place the ith list, all of its elements share the exact same level. Therefore, a list will have height (or, level) at least $j+1$ only if the two choices for its super bins are of level at least j. Since there are at most β_j bins with load at least j (or, β_j/n_i super bins of size n_i) and m bins (or, m/n_i super bins of size n_i), the probability that the two possible choices have level at least j is $(\beta_j/m)^2$. Therefore, the expected number of list elements of height at least $j+1$ is at most $n \cdot (\beta_j/m)^2$, and so the number of list elements of height at least $j+1$ is no more than twice this amount with high probability (using some tail bound). This implies that there are no more than $2n \cdot (\beta_j/m)^2$ bins with height at least $j+1$ and this leads to the following recursive relation:

$$\beta_{j+1} \stackrel{\text{def}}{=} 2n \cdot \left(\frac{\beta_j}{m}\right)^2 .$$

The induction. We confirm the recursive relation by induction. We prove that with high probability, the number of bins with list elements of height at least $j+1$ is bounded by β_{j+1}, conditioned that the number of bins with load at least j is bounded by β_j. This step is somewhat subtle in the original proof, primarily because one must handle the conditioning appropriately, and the formal argument requires some care. In our case, the argument is even more subtle. In the original proof, this step is shown using Chernoff's bound, however, we cannot use this bound here due to the fact that the random variables do not take values in $\{0,1\}$, and depend on the lengths of the lists. We use Bernstein's bound instead, and this step already introduces our limitation on the length of the longest list.

Concluding the first part of Theorem 3.5. The proof then proceeds as follows, where for this high-level overview, we assume for simplicity that $m = n$. We may set $\beta_4 = n/4$, since there cannot be more than $n/4$ bins with at least 4 elements. As in the original proof, the induction step does not hold unconditionally, and therefore we cannot use the recursion all the way to the first index k for which $\beta_k < 1$. Instead, we have to stop earlier, when $\beta_j = S \log^2 n$ where S is some upper bound on the length of the longest list. It turns out that the original proof is a special case of ours, where this layered induction holds as long as $\beta_j \geq \log^2 n$ (i.e., $S = 1$). The first index j^* for which $\beta_{j^*} = S \log^2 n$ is when $j^* = O(\log\log(n/S))$. This means that there are no more than $S \log^2 n$ list elements at level greater than $O(\log\log(n/S))$, proving the first part of Theorem 3.5.

The second part of Theorem 3.5. At this stage, we cannot use the tail bound any more and we need a different (more coarse) argument. Clearly, if there is a bin with load at least $j^* + a$, then there are at least a lists with height at least $j^* + 1$. Assuming that there are no more than $\beta_{j^*} = S \log^2 n$ bins with load at least j^*, using a simple union bound argument, we can bound the probability that there are at least a lists with height at least $j^* + 1$ by roughly $\left(S^2/n\right)^a$. This is negligible if $a = O(\log \log n)$ and $S = n^\delta$ for some constant $\delta < 1/2$. Combining this with the above, we conclude that the maximal load of the process is $O(\log \log n)$ assuming that there are no lists with more than n^δ elements.

A scaling argument. The above shows that the algorithm places lists of length up to n^δ for some constant $\delta < 1/2$ (say, for concreteness, $\delta = 1/3$). We claim that the algorithm actually succeeds in placing much longer lists, and we show this using a more subtle argument.

Our first observation is that the algorithm can be "scaled". Namely, consider the following two executions of the algorithm. In the first execution, we have list of lengths $(n_1, \ldots, n_k)$ and number of bins m where all are multiples of some s. In the second execution, instead of running the algorithm on these inputs, we divide the elements to blocks of size s, run the algorithm on the input $(n_1/s, \ldots, n_k/s)$ and m/s, and then extend each bin to s bins (by placing the blocks in the respective bins). This, in fact, is what implicitly happens in the first execution, and therefore these two executions are equivalent.

As a matter of fact, using the behavior of the algorithm on the latter case we can conclude its properties on the former. Specifically, let $n' = n/s$ (i.e., n' is the sum of lengths in the scaled case). The failure probability of this execution is about $(S^2/n')^a$ and it can handle lists of length at most $n'^{1/3}$, which implies that the former case in fact fails with probability about $(S^2/(n/s))^a$ and handles lists of lengths at most $(n/s)^{1/3}$. By taking $s = n^{1/3}$, we obtain that the algorithm can handle lists of lengths between $s = n^{1-(2/3)^1}$ and $n^{1-(2/3)^2}$ with all but negligible probability. We can repeat this argument carefully, and show that the algorithm can handle lists of length between $n^{1-(2/3)^\alpha}$ and $n^{1-(2/3)^{\alpha+1}}$ with all but negligible probability, for any constant $\alpha > 0$.

Our second observation is that the algorithm can in fact handle all of these possible inputs *simultaneously*. This is because the algorithm behaves similarly to an algorithm that divides the lists into sets of inputs of lengths between $n^{1-(2/3)^\alpha}$ and $n^{1-(2/3)^{\alpha+1}}$, invokes the algorithm on each set of inputs independently (i.e., on a different set of bins), and then combines the bins of all these executions. By carefully defining the number of sets, we conclude that the maximal load of the latter algorithm is at most $\tilde{O}(\log \log n)$, implying that our algorithm behaves similarly to that as well.

Proof of Theorem 3.5. We will do our calculations with respect to some bound $N \geq n$. We point out an important loop invariant: Before the loop i, for each $0 \leq \beta \leq \frac{m}{n_i} - 1$, the current load of all the bins in the super bin $\widetilde{B}_\beta = (B_{n_i \cdot \beta + j})_{j=0}^{n_j - 1}$ is the same. So all the elements of the list L_i are placed at the same height. We refer to that common height as the height of the list L_i, and denote it by $h(i)$.

We make further notations: Let $\ell = \lceil \frac{4n}{m} \rceil$. X_i is where we placed the head of the list L_i. We refer to the stage right after placing the list L_i as "time i". $\nu_j(i)$ is the number of bins with load $\geq \ell + j$ at time i. $\mu_j(i)$ is the number of list *elements* with height $\geq \ell + j$ at time i (mathematically, $\sum_{s \leq i \wedge h(s) \geq \ell + j} n_s$). We also define $\nu_j(0) = \mu_j(0) = 0$. Trivially we have $\nu_j(i) \leq \mu_j(i)$. We define $\beta_0 = \frac{m^2}{4n}$, $\beta_{j+1} = 2n\frac{\beta_j^2}{m^2}$ and $\mathcal{E}_j = \{\nu_j(k) \leq \beta_j\}$. We have $\Pr[\mathcal{E}_0] = 1$ since there cannot be more than $\frac{m}{\ell}$ bins with height $\geq \ell$.

Fix a level $\ell + j$. For $i \in [k]$ define the random variable Y_i as follows,

$$Y_i = \begin{cases} 1 & h(i) \geq \ell + j + 1 \wedge \nu_j(i-1) \leq \beta_j \\ 0 & \text{else} \end{cases}$$

Conditioned on $\mathcal{E}_j$ we have $\mu_{j+1}(k) = \sum_{i=1}^{k} n_i Y_i$. Letting $p_j = \frac{\beta_j^2}{m^2}$, we have $Y_i = Y_i(X_1, \ldots, X_i)$ and $\Pr[Y_i = 1 | X_1, \ldots, X_{i-1}] \leq p_j$, so by applying Corollary 2.9 we get

$$\Pr\left[\sum_{i=1}^{k} n_i Y_i > \beta_{j+1}\right] = \Pr\left[\sum_{i=1}^{k} n_i Y_i > 2np_j\right] \leq \exp\left(-\frac{3}{8}\frac{n}{S}p_j\right) \leq N^{-c \cdot \log N}$$

where the last step holds as long as $\frac{2n}{S} p_j \geq \log^2 N$, therefore

$$\begin{aligned} \Pr[\neg\mathcal{E}_{j+1}] &\leq \Pr[\neg\mathcal{E}_{j+1} \wedge \mathcal{E}_j] + \Pr[\neg\mathcal{E}_j] \\ &\leq \Pr[\mu_{j+1}(k) > \beta_{j+1} \wedge \mathcal{E}_i] + \Pr[\neg\mathcal{E}_j] \\ &\leq \Pr\left[\sum_{i=1}^{k} n_i Y_i > \beta_{j+1}\right] + \Pr[\neg\mathcal{E}_j] \\ &\leq N^{-c \log N} + \Pr[\neg\mathcal{E}_j] \end{aligned}$$

Let j^* be the minimal j such that $p_j < \frac{S}{2n} \log^2 N$. We shall estimate j^*. By induction one can see that

$$\beta_j \leq \frac{m^2}{n \cdot 2^{2^j}}, \qquad p_j = \frac{\beta_j^2}{m^2} \leq \left(\frac{m}{n \cdot 2^{2^j}}\right)^2.$$

We have

$$\left(\frac{m}{n \cdot 2^{2^{j^*-1}}}\right)^2 \geq p_{j^*-1} \geq \frac{S}{2n} \log^2 N,$$

and by applying log we get

$$2^{j^*} \leq \log \frac{n}{S} - 2 \log \frac{n}{m} - 2 \log\log N + 1 \leq \log \frac{n}{S}$$

so $j^* \leq \log\log \frac{n}{S}$. Now we apply the bound one more time with $p'_{j^*} = \frac{S}{2n} \log^2 N > p_{j^*}$. We set $\beta'_{j^*+1} = 2np'_{j^*} = S \log^2 N$, $\mathcal{F} = \{\mu_{j^*+1}(k) \leq \beta'_{j^*+1}\}$ and $\mathcal{E}'_{j^*+1} = \{\nu_{j^*+1}(k) \leq \beta'_{j^*+1}\}$. As before we get $\Pr[\neg\mathcal{F}] \leq N^{-c \log N} + \Pr(\neg\mathcal{E}_{j^*})$, so we conclude by induction that $\Pr(\neg\mathcal{E}'_{j^*+1}) \leq \Pr[\neg\mathcal{F}] \leq (j^* + 1) \cdot N^{-c \log N} = N^{-\Omega(\log N)}$, so

$$\Pr[\mathcal{F}] = \Pr\left[\mu_{j^*+1}(k) \leq S \log^2 N\right] = 1 - N^{-\Omega(\log N)}$$

as claimed.

Now we assume $m \geq \frac{N}{\log^2 N}$ and $S = N^\delta$ for some $0 \leq \delta < \frac{1}{2}$. For the level $\ell + j^* + 1$ we simply define

$$Y'_i = \begin{cases} 1 & h(i) \geq \ell + j^* + 2 \\ 0 & \text{else} \end{cases}$$

then we have

$$\Pr[Y'_i = 1 \wedge \mathcal{E}'_{j^*+1}] \leq \frac{\beta'^2_{j^*+1}}{m^2} = \frac{S^2}{m^2} \log^4 N \leq \frac{\log^8 N}{N^{2-2\delta}}.$$

If $\nu_{j^*+a+1}(k) \geq 1$ then there at least a lists with height $\geq \ell + j^* + 2$, therefore by a union bound we get

$$\begin{aligned}\Pr[\nu_{j^*+a+1}(k) \geq 1 \wedge \mathcal{E}'_{j^*+1}] &\leq \Pr\left[\sum_{i=1}^{k} Y'_i \geq a \wedge \mathcal{E}'_{j^*+1}\right] \\ &\leq \binom{k}{a}\left(\frac{\log^8 N}{N^{2-2\delta}}\right)^a \\ &\leq N^a\left(\frac{\log^8 N}{N^{2-2\delta}}\right)^a = \left(\frac{\log^8 N}{N^{1-2\delta}}\right)^a\end{aligned}$$

so by taking $a = f(N)$ we get

$$\begin{aligned}\Pr[\nu_{j^*+a+1}(k) \geq 1] &\leq \Pr[\nu_{j^*+a+1}(k) \geq 1 \wedge \mathcal{E}'_{j^*+1}] + \Pr[\neg\mathcal{E}'_{j^*+1}] \\ &\leq \left(\frac{\log^8 N}{N^{1-2\delta}}\right)^{f(N)} + N^{-\Omega(\log N)} = N^{-\Omega(f(N))}\end{aligned}$$

for the level $\ell + j^* + a + 1 = \frac{4n}{m} + O(f(N))$. Note that the constants here depend on δ, but we will actually consider a specific case, namely $\delta = \frac{1}{3}$. We obtain the claim for any $\epsilon > 0$ by the following scaling argument. First we point out some observations:

- Our algorithm is scalable, i.e., if every list is larger than some power of two s, and we define a new input by dividing the length of every list, $n'_i = \frac{n_i}{s}$, and also the number of bins is $m' = \frac{m}{s}$, then the algorithm works on the original input as if it worked on the new input and scaled the result by duplicating each bin s times.
- Suppose $S = N^{\frac{1}{3}}$, $m \geq \frac{N}{\log^2 N}$. Then we saw that the maximal load exceeds $\frac{4n}{m} + O(f(N))$ with probability at most $N^{-\Omega(f(N))}$. Moreover, the same argument shows that if we already placed some other larger lists, we will increase the current maximal load by at least $\frac{4n}{m} + O(f(N))$ with probability at most $N^{-\Omega(f(N))}$.

Now we assume $S = n^{1-\epsilon}$ and $m \geq \frac{n}{\log n}$. We may assume that $\epsilon \geq \frac{1}{\sqrt{\log n}}$ since the claim is vacuously true for smaller ϵ. We set $\gamma = \frac{2}{3}$ and let $z = \lceil \log_\gamma \epsilon \rceil$. Then $\gamma^z \leq \epsilon < \gamma^{z-1}$. We divide the lists into z sets by $V_j = \{i \mid n^{1-\gamma^j} \leq n_i \leq n^{1-\gamma^{j+1}}\}$ for $j = 0, \ldots, z-1$. We scale the lists in V_j by $s = n^{1-\gamma^j}$, $n'_i = \frac{n_i}{s}$. Also we scale then number of bins $m' = \frac{m}{s}$. Now the input size is $n' = \sum_{i \in V_j} n'_i$. Let $N' = \frac{n}{s} \geq n'$. We have $N' = n^{\gamma^j} \geq n^\epsilon$, so

$$m' = \frac{N'}{\log n} = \frac{N'}{\log^2 N'} \cdot \frac{\log^2 N'}{\log n} \geq \frac{N'}{\log^2 N'} \cdot \epsilon^2 \log n \geq \frac{N'}{\log^2 N'}$$

Each list is of length at least $\frac{n^{1-\gamma^{j+1}}}{n^{1-\gamma^j}} = n^{\gamma^j(1-\gamma)} = (N')^{\frac{1}{3}}$, So by the scaling observation, the lists in V_j increase the maximal load by at least

$$\frac{4n'}{m'} + O(f(N')) \leq \frac{4\sum_{i \in V_j} n_i}{m} + O(f(n))$$

with probability at most $(N')^{-\Omega(f(N'))} \leq n^{-\Omega(\epsilon \cdot f(n^\epsilon))}$. So by a union bound, we get that the maximal load is at most $\frac{4n}{m} + O(\log \epsilon^{-1} \cdot f(n))$ with probability $1 - O(\log \epsilon^{-1}) \cdot n^{-\Omega(\epsilon \cdot f(n^\epsilon))}$. ∎

3.2.2 The Tightness of Our Analysis

The above analysis assumes a bound on the length of the longest list. Specifically, we assumed an upper bound $n^{1-\epsilon(n)}$ on the length of the longest list, where $\epsilon = (\log\log n)^{-A}$, and proved an upper bound $g(n)$ on the maximal load of Algorithm 3.4 with an overwhelming probability, where $g(n) = \tilde{O}((\log\log n)^A)$. The following claim shows that this trade-off is in fact tight, namely that $g(n) = \omega((\log\log n)^A)$:

Claim 3.7. *For any constants $A \geq 0$ and $c \geq 0$, and for any sufficiently large enough n, there exists an input for Algorithm 3.4 such that (1) the number of bins is $m = n$, (2) the length of the longest list is $n^{1-\epsilon}$ where $\epsilon = (\log\log n)^{-A}$, and (3) the maximal load is at least $c \cdot (\log\log n)^A$ with a noticeable probability.*

Proof. Consider the following input to Algorithm 3.4. The number of bins is $m = n$, and the set of lists consist of $L \leq n^\epsilon$ lists of the length $n' \stackrel{\text{def}}{=} n^{1-\epsilon}$, and an arbitrary number of lists (and structure) for the remaining $n - L \cdot n^{1-\epsilon}$ elements (but still with no list longer than n').

We compute the probability that all two choices of all the L long lists share the exact same super bin (therefore, the load of each of the bins that constitute that super bin is at least L). Since we have independence between different lists, this probability is

$$\left(\frac{n'}{m}\right)^{2 \cdot L} = \left(\frac{n^{1-\epsilon}}{n}\right)^{2 \cdot L} = n^{-2L\epsilon} \ .$$

Taking $\epsilon = (\log\log n)^{-A}$ and $L = c \cdot (\log\log n)^A$, the above occurs with a noticeable probability and the maximal load is at least $c \cdot (\log\log n)^A$. Note that n should be large enough such that $L \leq n^\epsilon$. ■

We note that the above argument can be generalized for a similar d-choice process for $d > 2$, and for a number of bins which is as large as $n \log n$. In addition, in Section 4.4 we generalize the above to a more general family of allocation algorithms.

4 SSE Schemes from Two-Dimensional Balanced Allocations

In this section we present a general framework for basing searchable symmetric encryption schemes on two-dimensional balanced allocations algorithms. First, in Section 4.1 we formally define the notion of an allocation algorithm, and derive concrete instantiations based on our two-dimensional allocation schemes from Section 3. Then, in Section 4.2 we show how to transform any allocation algorithm into a searchable symmetric encryption scheme using cryptographic techniques.

4.1 Allocation Algorithms

An allocation algorithm receives as input a database $\mathsf{DB} = \{\mathsf{DB}(w_1), \ldots, \mathsf{DB}(w_{n_\mathsf{W}})\}$ and places its elements in an array. For each list $\mathsf{DB}(w_i)$, we distinguish between its *possible* locations in the array, and its *actual* locations in the array. We restrict our attention to allocation algorithms in which the *possible* locations of each list $\mathsf{DB}(w_i)$ depend only on its length $n_i = |\mathsf{DB}(w_i)|$ and on the size $N = \sum_{i=1}^{n_\mathsf{W}} |\mathsf{DB}(w_i)|$ of the database. In particular, the possible locations of different lists are independent (in contrast, the actual locations of different lists are naturally allowed to be dependent). We model this property by considering allocation algorithms that follow a two-step structure. First, they separately generate the possible locations for each list using a procedure denoted $\mathsf{RangesGen}$. Then, given the entire collection of possible locations of all list, they determine the actual locations of the lists using a procedure denote $\mathsf{Allocation}$. The structure of the algorithm is described as Structure 4.1.

STRUCTURE 4.1 (The structure of the allocation algorithm Allocation-Alg)**.**

- **Input:** k integer values $(n_1, \ldots, n_k)$ representing the lengths of the lists. Let $N = \sum_{i=1}^{k} n_i$.
- **The algorithm:**
  1. For every $1 \leq i \leq k$ let $R_i \leftarrow \mathsf{RangesGen}(N, n_i)$.
     (The procedure $\mathsf{RangesGen}(N, n_i)$ *outputs the possible ranges* $R_i = \{[a_1, b_1], \ldots, [a_d, b_d]\}$ *for the* i*th list.)*
  2. Let $\mathsf{map} \leftarrow \mathsf{Allocation}\left((n_1, R_1), \ldots, (n_k, R_k)\right)$.
     (The array map *holds the actual locations of the lists. Each entry in this array contains either a pair* (i, j) *representing that this entry is the actual location of the* j*th element from the* i*th list, or* NULL *representing an empty entry.)*
  3. Output map.

We sometimes find it convenient to denote by $\mathsf{RangesGen}(N, n_i; r)$ an invocation of the procedure RangesGen on input (N, n_i) using specific randomness r.

Efficiency measures. We measure the efficiency of an allocation algorithm of the above structure with respect to its space overhead, its locality, and its read efficiency.

Definition 4.2. *We say that* Allocation-Alg $= (\mathsf{RangesGen}, \mathsf{Allocation})$ *is an* $(\mathsf{s}, \mathsf{d}, \mathsf{r})$*-*allocation algorithm*, for some functions* $\mathsf{s}(\cdot)$*,* $\mathsf{d}(\cdot)$ *and* $\mathsf{r}(\cdot)$*, if the following properties hold:*

- **Correctness:** *There exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every input* $(n_1, \ldots, n_k)$

$$\Pr\left[\left(\mathsf{Allocation\text{-}Alg}\left(n_1, \ldots, n_k\right) = \perp\right) \ \vee \ \mathsf{invalid\text{-}allocation}\right] \leq \mathsf{negl}(N) \ ,$$

  where $N = \sum_{i=1}^{k} n_i$*, and* invalid-allocation *denotes the event in which* $\mathsf{Allocation\text{-}Alg}(n_1, \ldots, n_k)$ *outputs an allocation such that there exists two different actual placements for some element, or that there exists some element with no actual placement. The probability is taken over the internal coin tosses of the algorithm* Allocation-Alg*.*
- **Space:** *For every input* $(n_1, \ldots, n_k)$*, the array produced by* Allocation-Alg *is of size at most* $\mathsf{s}(N)$*, where* $N = \sum_{i=1}^{k} n_i$*.*
- **Locality:** *For every input* (N, n_i)*, the algorithm* RangesGen *outputs at most* $\mathsf{d}(N)$ *ranges.*
- **Read efficiency:** *For every input* (N, n_i) *for the algorithm* RangesGen *it holds that:*

$$\frac{\sum_{j=1}^{\mathsf{d}} |b_j - a_j|}{n_i} \leq \mathsf{r}(N) \ ,$$

  where $\{[a_1, b_1], \ldots, [a_{\mathsf{d}}, b_{\mathsf{d}}]\} \leftarrow \mathsf{RangesGen}(N, n_i)$*.*

Concrete instantiations. Our two-dimensional balanced allocation schemes described in Section 3 have the above-discussed two-step structure (more accurately, they can be easily re-written as algorithms that follow this structure). From Corollaries 3.3 and 3.6 we obtain that our one-choice scheme OneChoiceAllocation (Algorithm 3.1) is an $(O(N), O(1), \tilde{O}(\log N))$-allocation algorithm, and that our two-choice scheme TwoChoiceAllocation (Algorithm 3.4) is an $(O(N), O(1), \tilde{O}(\log \log N))$-allocation algorithm for databases in which no list has more than $N^{1-1/\log \log N}$ elements.

4.2 From Allocation Algorithms to SSE Schemes

We show a generic transformation from any allocation algorithm $\mathsf{Allocation\text{-}Alg} = (\mathsf{RangesGen}, \mathsf{Allocation})$ to a searchable symmetric encryption scheme. In our SSE scheme, the client will run the $\mathsf{RangesGen}$ and the $\mathsf{Allocation}$ procedures, and then encrypt each of the document identifiers from list $\mathsf{DB}(w)$ with a key that is derived from the keyword w using a pseudorandom function. In addition, any unused entry in the array will be filled with a uniform random string of the appropriate length. Then, when issuing a query corresponding to a keyword w, the client will ask the server to retrieve the encrypted content of all possible locations of the list $\mathsf{DB}(w)$. Since these locations are chosen independently at random, this does not reveal any information on the list except for its length. The client then decrypts the contents using the respective key for the list. The formal construction is described in Construction 4.3.

CONSTRUCTION 4.3.

Parameters: A keyword set $\mathsf{W} = \{w_1, \ldots, w_{n_\mathsf{W}}\}$, and a database $\mathsf{DB} = \{\mathsf{DB}(w_1), \ldots, \mathsf{DB}(w_{n_\mathsf{W}})\}$. We denote by $N = \sum_{i=1}^{n_\mathsf{W}} |\mathsf{DB}(w_i)|$ the size of the database, and for each keyword w_i we denote by $n_i = |\mathsf{DB}(w)|$ the number of documents containing w_i.

Key generator. The algorithm $\mathsf{KeyGen}(1^\lambda)$ samples a PRF key K.

Setup. The algorithm $\mathsf{EDBSetup}(\mathsf{DB}, K)$ proceeds as follows:

1. Initialize an empty set S.
2. For every keyword $w_i \in \mathsf{W}$:
   (a) Let $\mathsf{DB}(w_i) = \left\{\mathsf{id}_1^{(i)}, \ldots, \mathsf{id}_{n_i}^{(i)}\right\}$.
   (b) Compute $(\ell_i, k_i, r_i, K_i) = \mathsf{PRF}_K(w_i)$.
   (c) Compute $\widehat{n_i} = n_i \oplus k_i$.
   (d) Add the pair $(\ell_i, \widehat{n_i})$ to the set S.
   (e) Compute $R_i = \mathsf{RangesGen}(N, n_i; r_i)$.
3. Pad S until it contains exactly N pairs with random elements. Uniformly shuffle S and compute $\mathsf{HT} \leftarrow \mathsf{HTSetup}(S)$.
4. Compute $\mathsf{map} \leftarrow \mathsf{Allocation}(\{(n_i, R_i)\}_{i=1}^{n_\mathsf{W}})$. If $\mathsf{map} = \perp$ then abort and output $\perp$. Otherwise, define the data block Data of size $\mathsf{s}(N)$ as follows, where for every $1 \le t \le \mathsf{s}(N)$:

$$\mathsf{Data}[t] = \begin{cases} \mathsf{Enc}_{K_i}(\mathsf{id}_j^{(i)}) & \text{if } \mathsf{map}[t] = (i,j) \\ U_\ell & \text{otherwise} \end{cases}, \tag{1}$$

   where U_ℓ denote a uniformly and independently sampled ℓ-bit string for each entry.
5. Output: $\mathsf{EDB} = (\mathsf{Data}, \mathsf{HT})$.

Token generator. The algorithm $\mathsf{TokGen}(K, w_i)$ computes the derived keys $(\ell_i, k_i, r_i, K_i) = \mathsf{PRF}_K(w_i)$. It outputs $\tau_i = (\ell_i, k_i, r_i)$ as the public token, and $\rho_i = K_i$ as the secret state for the algorithm $\mathsf{Resolve}$.

Search. The algorithm $\mathsf{Search}(\tau_i, \mathsf{EDB})$, with $\tau_i = (\ell_i, k_i, r_i)$ and $\mathsf{EDB} = (\mathsf{Data}, \mathsf{HT})$ proceeds as follows:

1. Obtain $\widehat{n_i} \leftarrow \mathsf{HTGet}(\mathsf{HT}, \ell_i)$ and compute $n_i = \widehat{n_i} \oplus k_i$.
2. Run $\mathsf{RangesGen}(N, n_i; r_i)$. Let $P = \{[a_1, b_1], \ldots, [a_\mathsf{d}, b_\mathsf{d}]\}$ be the resulting ranges.
3. Output $\widetilde{D} = \bigcup_{[a_j, b_j] \in P} \mathsf{Data}[a_j, \ldots, b_j]$.

Resolve. The algorithm $\mathsf{Resolve}(\rho_i, \widetilde{D})$ with $\rho_i = K_i$, decrypts each element in $\widetilde{D}$ using the key K_i, and returns the set for which the decryption was successful.

Theorem 4.4. *Assume that* PRF *is a pseudorandom function, and that* $(\mathsf{Enc}, \mathsf{Dec})$ *has pseudorandom ciphertexts, as well as elusive and efficiently verifiable range. Moreover, let* $\mathsf{Allocation\text{-}Alg}$ *be an* $(\mathsf{s}, \mathsf{d}, \mathsf{r})$*-allocation algorithm. Then, Construction 4.3 is an adaptive* $\mathcal{L}^{\mathsf{adap}}_{\mathsf{size}}$*-secure searchable symmetric encryption scheme, with space* $O(\mathsf{s}(N))$*, locality* $O(\mathsf{d}(N))$*, and read efficiency* $O(\mathsf{r}(N))$*.*

Proof. The correctness of the scheme follows from the elusive and verifiable range properties of the underlying encryption scheme, and from the correctness of the procedure $\mathsf{Allocation\text{-}Alg}$. In particular, from the correctness of the latter procedure and the pseudorandomness of PRF (i.e., the r_i's are computationally indistinguishable from uniform values), we have all placeholders for all elements of all lists in the array map. Then, we use this array to create the array Data. From the verifiable range property, encryptions under different keys do not "collide". In addition, since the encryption scheme has elusive range, the random elements that the client introduces (see Eq. (1)) do not collide with valid encryptions of any of the actual elements. Furthermore, with all but a negligible probability, encryptions under one key are not valid for other keys, and the probability that the same label ℓ_i or the same key K_i appear more than once is negligible due to the pseudorandomness of PRF.

Regarding the efficiency, the size of the encrypted database is $|\mathsf{Data}| + |\mathsf{HT}|$, where $|\mathsf{Data}|$ is of size $\mathsf{s}(N)$ and HT contains exactly N elements of at most $\log N = O(\log \lambda)$ bits each (and thus can be stored using $O(N)$ machine words in the unit-cost RAM model). Therefore, the space usage is $O(\mathsf{s}(N) + N) = O(\mathsf{s}(N))$ (note that $\mathsf{s}(N) \geq N$ always hold). Locality and read efficiency are straightforward.

We now prove the adaptive security of the scheme with respect to the leakage function $\mathcal{L}^{\mathsf{adap}}_{\mathsf{size}}$. Recall that in both the real and the ideal executions, the adversary $\mathcal{A}$ first outputs DB. In the real execution, it then receives the encrypted database $\mathsf{EDB} \leftarrow \mathsf{EDBSetup}(K, \mathsf{DB})$. In the ideal execution, the (interactive) simulator $\mathcal{S}$ receives $\mathcal{L}^{\mathsf{adap}}_{\mathsf{size}}(\mathsf{DB}) = N$ and has to generate EDB. Then, $\mathcal{A}$ adaptively issue queries $\mathbf{w}$. For each query $w_i \in \mathbf{w}$, in the real execution it receives the token $\tau_i \leftarrow \mathsf{TokGen}(K, w_i)$, and in the ideal execution the simulator $\mathcal{S}$ receives $\mathcal{L}^{\mathsf{adap}}_{\mathsf{size}}(\mathsf{DB}, \{w_j\}_{j<i}, w_i) = |\mathsf{DB}(w_i)|$ and has to generate τ_i. The adversary $\mathcal{A}$ should not be able to distinguish between the real and ideal executions. The simulator $\mathcal{S}$ works as follows.

- **Input:** *Initially,* $\mathcal{L}^{\mathsf{adap}}_{\mathsf{size}}(\mathsf{DB}) = N$*.*
- **The simulator:**
  - **Initialization phase.**
    1. $\mathcal{S}$ *creates data block* Data *containing* $\mathsf{s}(N)$ *elements, each chosen uniformly and independently at random.*
    2. *It initializes a set* S *with* N *random elements* $(\ell_{i'}, \widehat{n_{i'}})$*.*
    3. *It uniformly shuffles* S *and computes* $\mathsf{HT} \leftarrow \mathsf{HTSetup}(S)$*.*
    4. $\mathcal{S}$ *then outputs:* $\mathsf{EDB} = (\mathsf{Data}, \mathsf{HT})$*.*
  - **Query.** *With each query that the experiment performs, the simulator receives the leakage* $|\mathsf{DB}(w_i)|$*.*
    1. $\mathcal{S}$ *chooses a pair* $(\ell_{i'}, \widehat{n_{i'}})$ *from* S *and removes the pair.*
    2. *It computes* $k_i = |\mathsf{DB}(w_i)| \oplus \widehat{n_{i'}}$*.*
    3. *It uniformly samples* r_i*.*
    4. $\mathcal{S}$ *then outputs:* $\tau_i = (\ell_{i'}, k_i, r_i)$*.*

We now claim that no adversary can distinguish whether $(\mathsf{EDB}, \boldsymbol{\tau} = \{\tau_i\}_{w_i \in \mathbf{w}})$ was generated in the real execution or by the simulator. We show this by the following hybrid experiments:

- $\mathbf{Hyb_0}$. This is the real execution, where the adversary $\mathcal{A}$ receives EDB as output of $\mathsf{EDBSetup}$, and $\tau_i = (\ell_i, k_i, r_i)$ with $(\ell_i, k_i, r_i, K_i) = \mathsf{PRF}_K(w_i)$ for every query w_i.
- $\mathbf{Hyb_1}$. In this experiment, we run the real world execution, where instead of using PRF_K (where $K \leftarrow \mathsf{KeyGen}(1^\lambda)$) in Step 2b of Construction 4.3, we use a truly random function. Note that as a result, all the keys (ℓ_i, k_i, r_i, K_i) are uniform (for every $w_i \in \mathsf{W}$). In addition, note that the key K as outputted by KeyGen is redundant.
- $\mathbf{Hyb_2}$. In this experiment, we change the value $\widehat{n}_i$ (in Step 2c of Construction 4.3) to be a uniform value of the appropriate length. Then, when generating a token we compute $k_i = \widehat{n}_i \oplus n_i$ instead of using the key k_i generated by the truly random function. Note that this makes the key k_i generated by the truly random function redundant.
- $\mathbf{Hyb_3}$. Here, for every $w_i \in \mathsf{W}$, we replace the encryption of each element in $\mathsf{DB}(w_i)$ (i.e., Eq. (1) of Construction 4.3) with an independent truly random value of the appropriate length. As a result, all elements of the data block Data are uniform and independent.
- $\mathbf{Hyb_4}$. The last hybrid is the ideal execution. Here we run the simulator $\mathcal{S}$ defined above instead of the algorithms $\mathsf{EDBSetup}$ and TokGen.

Hyb_0 and Hyb_1 are computationally indistinguishable from the security assumption of the pseudorandom function PRF. The experiments Hyb_1 and Hyb_2 are identically distributed since for every $w_i \in \mathsf{W}$, $\widehat{n}_i$ is uniformly distributed in both experiments, and for every query $w_i \in \mathbf{w}$, k_i is determined by $k_i = \widehat{n}_i \oplus n_i$. Hyb_2 and Hyb_3 are computationally indistinguishable based on the pseudorandom ciphertexts property of the encryption scheme Enc. Finally, Hyb_3 and Hyb_4 are statistically close, where the only difference is the possibility of failure of the algorithm $\mathsf{Allocation}$, which is negligible in N. This concludes the proof. ■

Conclusion. By combining Theorem 4.4 with the allocation algorithms $\mathsf{OneChoiceAllocation}$ (Algorithm 3.1) and $\mathsf{TwoChoiceAllocation}$ (Algorithm 3.4) we obtain the following corollary:

Corollary 4.5. *Assuming the existence of one-way functions, there exist searchable symmetric encryption schemes with the following properties:*

1. *Space $O(N)$, locality $O(1)$, and read efficiency $\tilde{O}(\log N)$ without any assumptions on the structure of the database.*
2. *Space $O(N)$, locality $O(1)$, and read efficiency $\tilde{O}(\log \log N)$ assuming that no keyword appears in more than $N^{1-\frac{1}{\log \log N}}$ documents.*

4.3 Extensions

We discuss the following extensions of our generic transformation from allocation algorithms to searchable symmetric encryption schemes (Construction 4.3).

Reducing one round of interaction. When considering the SSE scheme that is based on our one-choice allocation algorithm $\mathsf{OneChoiceAllocation}$, we can in fact reduce one round of interaction (as in [CT14]). Specifically, we revisit Construction 4.3 and provide some modifications. We then claim that under some additional assumptions on the underlying allocation algorithm (which are satisfied by our one-choice algorithm), the resulting construction is still a secure SSE scheme.

The modifications are as follows. Assume that while the client encrypts the lists, in addition it shuffles the elements in each bin. Then, we just combine the $\mathsf{Resolve}$ algorithm into the algorithm Search. That is, we also give the server the secret state ρ_i, so that it can perform the $\mathsf{Resolve}$

algorithm by itself and will not have to send the additional message to the client. Given the previous algorithms ($\mathsf{KeyGen}, \mathsf{EDBSetup}, \mathsf{TokGen}, \mathsf{Search}, \mathsf{Resolve}$), our new scheme defines $\mathsf{EDBSetup}$ algorithm is modified as above (i.e., shuffles each bin), and for $(\tau_i, \rho_i) = \mathsf{TokGen}(K, w_i)$, we define the new search algorithm as $\mathsf{Search}'(\tau_i, \mathsf{EDB}) = \mathsf{Resolve}(\rho_i, \mathsf{Search}(\tau_i, \mathsf{EDB}))$. The scheme omits the $\mathsf{Resolve}$ algorithm, since the server already obtains the list of identifiers (or, documents).

We note that this is secure since the allocation algorithm satisfies the following (strong) property: not only that the *possible* location of each list is independent of the possible locations of other lists, but rather, the *actual* location of each list is independent of the structure of the other lists, conditioned that the allocation was successful (no bin was overflowing, which occurs with all but negligible probability). However, the security obtained in this manner is only static in the standard model (or adaptive in the random oracle model) – see Section 5 where we deal with a similar issue and provide a more elaborate discussion.

On the actual leakage of the scheme. The security obtained in Theorem 4.4 is for the modest leakage function $\mathcal{L}^{\mathsf{adap}}_{\mathsf{sizes}}$. However, this holds as long as the client stores a database of keyword-document pairs, and the decryption is done by the client and is not revealed to the server. In case that the database consists of keyword-identifier pairs, then recall that in the second round the client sends the identifiers and receives back the encrypted documents. As a result, although the first round of interaction can be simulated using the $\mathcal{L}^{\mathsf{adap}}_{\mathsf{sizes}}$-leakage function, this leakage is not enough for simulating the second round, and the actual leakage is $\mathcal{L}^{\mathsf{adap}}_{\min}$.

4.4 A Limitation of Allocation Algorithms

We prove that any "very efficient" allocation algorithm that follows Structure 4.1, must assume a certain upper bound on the length of the longest list (this bound essentially matches the bound that is assumed by our two-choice algorithm). This is a generalization of Section 3.2.2, where here we consider some more general algorithms, but still follow essentially the same approach.

We note that since we consider a specific and rather restricted structure of allocation algorithms, we are able to prove a tighter bound than the one presented by Cash and Tessaro [CT14]. In fact, our construction in Section 5, which is based on a different technique (and not on allocation algorithms), circumvents this lower bound. This shows some dichotomy with respect to allocation algorithms: While these algorithms behave nicely while restricting the length of the maximal list (with respect to all three measures of efficiency), their efficiency becomes somewhat unsatisfying without this restriction. Adding the same restriction on the length of the maximal list to any of the other SSE schemes that have appeared in the literature (as well as to our construction in Section 5), does not seem to improve these constructions by much and they all behave asymptotically the same[8].

Our argument here is based on that from Section 3.2.2, where we generalize it in the following two directions. First, the $\mathsf{RangesGen}$ algorithm may choose an arbitrary number of ranges using some arbitrary distributions, and it is not restricted to choosing only two ranges of the appropriate length uniformly at random. For instance, the $\mathsf{RangesGen}$ may output d dependent ranges for the same list. Second, in our two-choice allocation scheme we analyzed some specific $\mathsf{Allocation}$ algorithm. Here, we show that the limitation holds for any implementation of the $\mathsf{Allocation}$ algorithm.

Proposition 4.6. *Let* $\mathsf{Allocation\text{-}Alg} = (\mathsf{RangesGen}, \mathsf{Allocation})$ *be an* $(\mathsf{s}, \mathsf{d}, \mathsf{r})$*-allocation algorithm with* $\mathsf{s}(N) = O(N \log N)$ *and* $\mathsf{d}(N)^2\mathsf{r}(N) \leq \log N/(c \cdot \log\log N)$ *for some constant* $c \geq 1$*, and let*

[8]For example, assuming that there are no lists of length greater than $N^{1-1/\log\log N}$, the space usage of our construction in Section 5 is reduced from $O(N \log N)$ to $O(N \log N(1 - 1/\log\log N))$, which is still $O(N \log N)$, while its locality and read efficiency remain exactly the same.

$\epsilon(N) = c^{-1} \cdot \mathsf{d}(N)^{-2}\mathsf{r}(N)^{-1}$. *Then, for every* N*, there exists an input* $(n_1, \ldots, n_k)$ *with* $N = \sum_{i=1}^{k} n_i$ *and* $\max n_i = N^{1-\epsilon}$*, such that* Allocation-Alg $(n_1, \ldots, n_k)$ *fails with a noticeable probability.*

Proof. We modify RangesGen such that it returns exactly d ranges, each of size exactly $\mathsf{r} \cdot n_i$. Surely this only increases the probability that we will be able to accommodate the lists. Similarly to the proof of Claim 3.7, we take $L \leq N^{\epsilon}$ lists of size of $n' = N^{1-\epsilon}$ for some ϵ, and an arbitrary number of lists (and structure) for the remaining $N - L \cdot N^{1-\epsilon}$ elements (but still with no list longer than n'). From now on we will only refer to the first L lists.

For $1 \leq \ell \leq L$, $1 \leq j \leq \mathsf{d}$ and $1 \leq i \leq \mathsf{s}$, we define a random variable $X^i_{\ell,j} \in \{0,1\}$, which indicates whether the jth range of the ℓth list contains the ith memory cell or not. We also define $W_\ell = (X^i_{\ell,j})_{j,i}$ of size $\mathsf{d} \times \mathsf{s}$, which gathers the variables that are related to the ℓth list. We point out that the variables $W_1, \ldots, W_L$ are independent and identically distributed (IID).

We now compute the probability that all ranges that are chosen to the L lists share the same ranges, or at least, share some few memory cells. As we will see, this will imply that there are a lot of intersections between the ranges and therefore for appropriate choice of parameters, we cannot accommodate the lists.

We denote the event $\mathcal{E}^i_j = \{X^i_{\ell,j} = 1 : \forall 1 \leq \ell \leq L\}$, i.e., the jth range of all lists contains the ith memory cell. For all ℓ we have $\sum_{i=1}^{\mathsf{s}} X^i_{\ell,1} = \mathsf{r} \cdot n'$, so $\sum_{i=1}^{\mathsf{s}} \Pr[X^i_{\ell,1} = 1] = \sum_{i=1}^{\mathsf{s}} E[X^i_{\ell,1}] = \mathsf{r} \cdot n'$, therefore there exists i_1 with $\Pr[X^{i_1}_{\ell,1} = 1] \geq \frac{\mathsf{r} \cdot n'}{\mathsf{s}}$. This i_1 is the same for all the lists because $W_1, \ldots, W_L$ are identically distributed. Thus, by independency (of choices between different lists) we get

$$\Pr[\mathcal{E}^{i_1}_1] = \prod_{\ell=1}^{L} \Pr[X^{i_1}_{\ell,1} = 1] \geq \left(\frac{\mathsf{r} \cdot n'}{\mathsf{s}}\right)^L = \left(\frac{N}{\mathsf{s}} \cdot \frac{\mathsf{r}}{N^{\epsilon}}\right)^L .$$

Let $1 < j \leq \mathsf{d}$ and suppose that we already defined $i_1, \ldots, i_{j-1}$. Conditioned on $\mathcal{E}^{i_1}_1, \ldots, \mathcal{E}^{i_{j-1}}_{j-1}$, the variables $W_1, \ldots, W_L$ are still IID, so applying the same argument, but with respect the conditional distribution, yields that there exists i_j such that

$$\Pr[\mathcal{E}^{i_j}_j \mid \mathcal{E}^{i_1}_1, \ldots, \mathcal{E}^{i_{j-1}}_{j-1}] \geq \left(\frac{N}{\mathsf{s}} \cdot \frac{\mathsf{r}}{N^{\epsilon}}\right)^L .$$

Setting $\mathcal{E} = \bigcap_{j=1}^{\mathsf{d}} \mathcal{E}^{i_j}_j$ we get

$$\Pr[\mathcal{E}] = \prod_{j=1}^{\mathsf{d}} \Pr[\mathcal{E}^{i_j}_j \mid \mathcal{E}^{i_1}_1, \ldots, \mathcal{E}^{i_{j-1}}_{j-1}] \geq \left(\frac{N}{\mathsf{s}} \cdot \frac{\mathsf{r}}{N^{\epsilon}}\right)^{\mathsf{d} \cdot L} .$$

Now if $\mathcal{E}$ occurs then we have strictly less than $2\mathsf{d} \cdot \mathsf{r} \cdot n'$ available memory cells to accommodate the lists. This is due to the fact that we can only use cells in the ranges $[i_j - \mathsf{r} \cdot n', i_j + \mathsf{r} \cdot n' - 1]$ for $1 \leq j \leq \mathsf{d}$ (this is the point in the proof where we use the consecutiveness of the ranges). On the other hand, the lists require $L \cdot n'$ memory cells, so if we take $L = 2\mathsf{d} \cdot \mathsf{r}$ we get that this event means failure for the Allocation algorithm. We conclude

$$\Pr[\text{Allocation-Alg}(n_1, \ldots, n_k) \text{ fails}] \geq \Pr[\mathcal{E}] \geq \left(\frac{N}{\mathsf{s}} \cdot \frac{\mathsf{r}}{N^{\epsilon}}\right)^{2\mathsf{d}^2\mathsf{r}} = 2^{-2\mathsf{d}^2\mathsf{r}(\epsilon \log N + \log \frac{\mathsf{s}}{N} - \log \mathsf{r})} \tag{2}$$

so if $\mathsf{d}^2\mathsf{r}(\epsilon \log N + \log \frac{\mathsf{s}}{N} - \log \mathsf{r}) = O(\log N)$ then this algorithm fails with a noticeable probability. Note that we need to have $2\mathsf{d} \cdot \mathsf{r} \leq N^{\epsilon}$, since otherwise by taking $L = 2\mathsf{d} \cdot \mathsf{r}$ lists we will exceed N elements.

Finally, we show that under our assumptions on $\mathsf{s}, \mathsf{d}, \mathsf{r}, \epsilon$, the probability of failure of Eq. (2) is noticeable and that $2\mathsf{d} \cdot \mathsf{r} \leq N^{\epsilon}$. We start with the failure probability. We have $\epsilon = c \cdot \mathsf{d}^{-2}\mathsf{r}^{-1}$, $\mathsf{d}^2\mathsf{r} \leq \log N/(c \cdot \log\log N)$ and $\frac{\mathsf{s}}{N} = O(\log N)$. So we get that

$$\mathsf{d}^2\mathsf{r} \cdot \left(\epsilon \log N + \log \frac{\mathsf{s}}{N} - \log \mathsf{r}\right) \leq c^{-1} \cdot \log N + \mathsf{d}^2\mathsf{r} \cdot \log \frac{\mathsf{s}}{N} = O(\log N) \ .$$

It remains to show that $2\mathsf{d} \cdot \mathsf{r} \leq N^{\epsilon}$ for a large enough N. This is equivalent to showing that $\epsilon^{-1}(1 + \log \mathsf{d} + \log \mathsf{r}) \leq \log N$. Indeed, we have that

$$\begin{aligned}
\epsilon^{-1} \cdot (1 + \log \mathsf{d} + \log \mathsf{r}) &= \mathsf{d}^2\mathsf{r} \cdot (1 + \log \mathsf{d} + \log \mathsf{r}) \\
&\leq c \cdot \mathsf{d}^2\mathsf{r} \cdot (1 + \log(\mathsf{d}^2\mathsf{r})) \\
&\leq \frac{\log N}{\log\log N} \cdot \left(1 + \log\left(\frac{\log N}{\log\log N}\right)\right) \\
&= \log N - \left(1 - \frac{1}{\log\log\log N}\right) \cdot \frac{\log N \log\log\log N}{\log\log N} \\
&\leq \log N,
\end{aligned}$$

where the last inequality holds for $N \geq 16$. ∎

Remark. Suppose that every range returned by $\mathsf{RangesGen}(N, n_i)$ has size at most $f(N) \cdot n_i$ (clearly we can always take $f(N) \leq \mathsf{r}(N)$). It can be easily seen from the proof, that we can strengthen Proposition 4.6 by replacing each appearance of $\mathsf{r}(N)$ by $f(N)$, that is, we require $\mathsf{d}^2(N)f(N) \leq \log N/(c \cdot \log\log N)$ and set $\epsilon(N) = c^{-1} \cdot \mathsf{d}(N)^{-2}f(N)^{-1}$.

5 Improving the Cash-Tessaro Scheme

In this section we improve the scheme proposed by Cash and Tessaro [CT14]. Their scheme offers space $O(N \log N)$, locality $O(\log N)$, and read efficiency $O(1)$. Our improvement reduces the locality from $O(\log N)$ to $O(1)$ while not hurting the space overhead or read efficiency. We begin by first describing the Cash-Tessaro scheme, and then explain our modification.

The Cash-Tessaro scheme. Let $\mathsf{DB} = \{\mathsf{DB}(w_1), \ldots, \mathsf{DB}(w_{n_\mathsf{W}})\}$ be a database of size $N = \sum_{i=1}^{n_\mathsf{W}} |\mathsf{DB}(w_i)|$, where each keyword w_i appears in $n_i = |\mathsf{DB}(w_i)|$ documents. The scheme consists of $\log N$ hash tables, each of size $O(N)$, where the kth table will store $N/2^k$ blocks of length 2^k.

For every keyword w_i, let $\left(s^{(i)}_{\log N}, \ldots, s^{(i)}_0\right)$ be the binary representation of n_i. Then, each list $\mathsf{DB}(w_i)$ is broken into at most $\log N$ consecutive blocks, where the kth block consists of 2^k document identifiers for every k for which $s^{(i)}_k = 1$. For every k we then insert all blocks of length 2^k into the kth hash table. Note that there can be at most $N/2^k$ such blocks, and each hash table is padded with "dummy" blocks to contain exactly $N/2^k$ blocks. The blocks corresponding to each list $\mathsf{DB}(w_i)$ are then encrypted using a key that is derived from w_i.

Reconstruction of a list $\mathsf{DB}(w_i)$ can be done easily using the key that is derived from w_i by querying each of the $\log N$ hash tables. Therefore, the scheme uses space $O(N \log N)$, locality $O(\log N)$ (since one has to query all $\log N$ hash tables), and read efficiency $O(1)$.

Our scheme. As in the Cash-Tessaro scheme, our scheme consists of $\log N$ hash tables of size N each, where the kth tables stores 2^k blocks of length $N/2^k$. The main idea underlying our scheme is to store each list $\mathsf{DB}(w_i)$ in a single table, instead of splitting it across the $\log N$ tables. Specifically, every list $\mathsf{DB}(w_i)$ of length n_i is stored in the p_ith table, where $2^{p_i-1} < |\mathsf{DB}(w_i)| \leq 2^{p_i}$. We pad the list with dummy elements until its length is exactly 2^{p_i}, and this increases the overall space by a factor of 2, but the entire list is now stored in one continuous block. In addition, we include a separate hash table that contains an encryption of p_i for every keyword w_i. We therefore obtain a scheme with space $O(N \log N)$, locality $O(1)$, and read efficiency $O(1)$. The scheme is presented as Construction 5.1.

CONSTRUCTION 5.1.

Parameters. Let t denote the value for which $2^{t-1} < |\mathsf{DB}| = N \leq 2^t$. For every word $w_i \in \mathsf{W}$, let $\mathsf{DB}(w_i) = \{\mathsf{id}_1, \ldots, \mathsf{id}_{n_i}\}$, and let p_i denote the value for which $2^{p_i-1} < n_i \leq 2^{p_i}$.

Key generator. Algorithm $\mathsf{KeyGen}(1^\lambda)$ samples a key K for the PRF.

Setup. Algorithm $\mathsf{EDBSetup}(\mathsf{DB}, K)$ proceeds as follows:

1. Initialize $t+1$ empty sets $T_0, \ldots, T_t$, where T_j will store at most 2^{t-j} pairs (ℓ, d), where $|d| = 2^j$. Initialize a set S, which will be converted to a lookup table.
2. For every keyword $w_i \in \mathsf{W}$

   (Let $\mathsf{DB}(w_i) = \{\mathsf{id}_1, \ldots, \mathsf{id}_{n_i}\}$, and recall that $2^{p_i-1} < n_i \leq 2^{p_i}$. If necessary, pad $\mathsf{DB}(w_i)$ with dummy identifiers in order to contain exactly 2^{p_i} elements.)
   (a) Compute $\mathsf{PRF}_K(w_i) = ((\ell_i, K_i), (\widehat{\ell_i}, \widehat{K_i}))$.
   (b) Create the blocks:
   $$d_i = (\mathsf{Enc}_{K_i}(\mathsf{id}_1), \ldots, \mathsf{Enc}_{K_i}(\mathsf{id}_{2^{p_i}})) \quad \text{and} \quad \widehat{n_i} = \mathsf{Enc}_{\widehat{K_i}}(n_i) \ .$$
   (c) Insert the pair (ℓ_i, d_i) to the set T_{p_i} and $(\widehat{\ell_i}, \widehat{n_i})$ to the set S.
3. Pad the list S to contain exactly N elements by adding random elements, and pad each set T_j to contain exactly 2^{t-j} pairs by adding random elements.
4. For each set $S, T_0, \ldots, T_t$, uniformly shuffle the set, generate the respective hash table by invoking the $\mathsf{HTSetup}$ algorithm, and obtain the hash tables $\mathsf{HT}(S), (\mathsf{HT}(L_0), \ldots, \mathsf{HT}(L_t))$.
5. Output $\mathsf{EDB} = (\mathsf{HT}(S), (\mathsf{HT}(L_0), \ldots, \mathsf{HT}(L_t)))$.

Token generator. Algorithm $\mathsf{TokGen}(K, w_i)$ outputs the four derived keys $\tau_i = ((\ell_i, K_i), (\widehat{\ell_i}, \widehat{K_i})) = \mathsf{PRF}_K(w_i)$.

Search. Algorithm $\mathsf{Search}(\tau_i, \mathsf{EDB})$, where $\tau_i = ((\ell_i, K_i), (\widehat{\ell_i}, \widehat{K_i}))$ and $\mathsf{EDB} = (\mathsf{HT}(S), \mathsf{HT}(L_0), \ldots, \mathsf{HT}(L_t))$ works as follows:

1. Invoke $\mathsf{HTLookup}$ on the hash table $\mathsf{HT}(S)$ and label $\widehat{\ell_i}$, receive back $\widehat{n_i} = \mathsf{Enc}_{\widehat{K_i}}(n_i)$. Decrypt n_i using the key $\widehat{K_i}$, and compute p_i.
2. Invoke $\mathsf{HTLookup}$ on the hash table $\mathsf{HT}(T_{p_i})$ and the label ℓ_i. Obtain the block $d_i = (\mathsf{Enc}_{K_i}(\mathsf{id}_1), \ldots, \mathsf{Enc}_{K_i}(\mathsf{id}_{2^{p_i}}))$. Decrypt the first n_i elements of this block using the key K_i and return them.

Theorem 5.2. *Assume that* PRF *is a pseudorandom function, and that* Enc *has pseudorandom ciphertexts. Then, Construction 5.1 is a static* $\mathcal{L}_{\min}$*-secure searchable symmetric encryption scheme for databases of size* N *with space* $O(N \log N)$*, locality* $O(1)$*, and read efficiency* $O(1)$*.*

Proof. It is easy to see that the scheme is correct except for the event where the same label appears more than once, which occurs in some negligible probability. For proving the security of the scheme, recall that in the real execution, the adversary $\mathcal{A}$ outputs $\mathsf{DB}, \mathbf{w}$, and is given the encrypted database

EDB and the tokens $\tau_i = \mathsf{TokGen}(K, w_i)$ for every $w_i \in \mathbf{w}$. In the ideal execution, the simulator $\mathcal{S}$ is given only $\mathcal{L}_{\min}(\mathsf{EDB}, \mathbf{w})$ and should output both EDB and $\boldsymbol{\tau} = \{\tau_i\}_{w_i \in \mathbf{w}}$ such that the adversary cannot distinguish between these two executions. Observe that the algorithms $\mathsf{KeyGen}, \mathsf{EDBSetup}$ and TokGen are not invoked in the ideal execution (by the experiment itself. They may be invoked by the simulator).

The simulator $\mathcal{S}$ works as follows:

- **Input:** $\mathcal{L}(\mathsf{DB}, \mathbf{w}) = (N, \{\mathsf{DB}(w_i)\}_{w_i \in \mathbf{w}})$.
- **The simulator:**
  1. The simulator $\mathcal{S}$ chooses (uniformly) random keys $\tau_i = ((\ell_i, K_i), (\widehat{\ell_i}, \widehat{K_i}))$ for every $w_i \in \mathbf{w}$.
  2. $\mathcal{S}$ initializes the sets $L_0, \ldots, L_t$ and S to be empty sets. For every $w_i \in \mathbf{w}$ and given $\mathsf{DB}(w_i)$, it computes n_i and p_i. If necessary, it pads $\mathsf{DB}(w_i)$ to size 2^{p_i} with some random additional id's.
  3. For every $w_i \in \mathbf{w}$, $\mathcal{S}$ computes:

$$d_i = (\mathsf{Enc}_{K_i}(\mathsf{id}_1), \ldots, \mathsf{Enc}_{K_i}(\mathsf{id}_{2^{p_i}})) \quad \text{and} \quad \widehat{n_i} = \mathsf{Enc}_{\widehat{K_i}}(n_i) \ .$$

     It adds the pair $(\widehat{\ell_i}, \widehat{n_i})$ to the set S and (ℓ_i, d_i) to the set L_{p_i}.
  4. After adding all the keywords in $\mathbf{w}$, $\mathcal{S}$ adds for each set L_j random elements until it contains exactly 2^{t-j} elements, and it adds to S random elements until it contains exactly N elements.
  5. $\mathcal{S}$ uniformly shuffles each one of the sets $S, T_0, \ldots, T_t$. It then generates hash table from each set using the algorithm $\mathsf{HTSetup}$, and defines $\mathsf{EDB} = (\mathsf{HT}(S), (\mathsf{HT}(T_0), \ldots, \mathsf{HT}(T_t)))$.
  6. $\mathcal{S}$ outputs EDB and $\boldsymbol{\tau} = \{\tau_i\}_{w_i \in \mathbf{w}}$.

We now claim that the adversary cannot distinguishes between the pair $(\mathsf{EDB}, \boldsymbol{\tau})$ that it receives in the real execution and in the ideal execution. Consider the following hybrid experiments:

- $\mathbf{Hyb_0}$. This is the real execution, where the adversary $\mathcal{A}$ receives EDB and $\tau_i = \mathsf{PRF}_K(w_i)$ for every $w_i \in \mathbf{w}$.
- $\mathbf{Hyb_1}$. This experiment is obtained from Hyb_0 by replacing the pseudorandom function $\mathsf{PRF}_K(\cdot)$ with a truly random function. Observe that, as a result, the key K that is produced by KeyGen is redundant and the elements $\tau_j = ((\ell_j, K_j), (\widehat{\ell_j}, \widehat{K_j}))$ are distributed uniformly.
- $\mathbf{Hyb_2}$. This experiment is obtained from Hyb_1 as follows: For every $w_j \in W \setminus \mathbf{w}$ we replace the values d_j and $\widehat{n_j}$ (in Step 2b of $\mathsf{EDBSetup}$) with independent and uniformly-distributed values of the appropriate length.
- $\mathbf{Hyb_3}$. This is the ideal execution, where we run the simulator $\mathcal{S}$ defined above.

We observe that Hyb_0 and Hyb_1 are computationally indistinguishable based on the security of the pseudorandom function PRF, and that Hyb_1 and Hyb_2 are computationally indistinguishable based on the pseudorandom ciphertexts property of the encryption scheme Enc. Finally, the experiments Hyb_2 and Hyb_3 are identical by the definition of the simulator $\mathcal{S}$. ■

Adaptive security in the random-oracle model. The scheme can be proven with adaptive security in the programmable random oracle model, using similar techniques as in [CT14]. Towards this goal, all we need is to instantiate the encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ with an encryption that allows "explaining" a random ciphertext as the encryption of any particular message (as in [CT14]). Concretely, we can use $\mathsf{Enc}_K(x; r) = \big(r, H(K||r) \oplus x0^\lambda\big)$, where H is the random oracle. Then, for

any derived key k, random (r, c), and a message m, we can program the random oracle such that $\mathsf{Enc}_k(m) = (r, c)$, by fixing $H(k||r) = c \oplus m0^\lambda$.

For the proof of security, the simulator in the first step creates random sets $L_0, \ldots, L_t$ and S, apply the $\mathsf{HTSetup}$ on each one of them and outputs the resulting tables as EDB. Later, upon a query w_i with a list $\mathsf{DB}(w_i) = \{\mathsf{id}_1, \ldots, \mathsf{id}_{n_i}\}$, it computes p_i, chooses some random pair $(\ell_i, (d_i^{(1)}, \ldots, d_i^{(2^{p_i})}))$ from the list L_{p_i} (and removes this pair from L_{p_i}), and chooses some random pair $(\widehat{\ell_i}, \widehat{n_i})$ from the set S (and removes it from S). Then, it chooses random keys $K_i, \widehat{K_i}$, and programs the random oracle such that the following hold:

$$\mathsf{Enc}_{K_i}(\mathsf{id}_j) = d_i^{(j)} \;\; \forall\, 1 \le j \le n_i, \qquad \text{and} \qquad \mathsf{Enc}_{\widehat{K_i}}(n_i) = \widehat{n_i}\ .$$

If the programming cannot succeed (because the corresponding values are already set for H), the simulator aborts. Otherwise, it outputs $\tau_i = ((\ell_i, K_i), (\widehat{\ell_i}, \widehat{K_i}))$. We omit a formal analysis of the above, as it follows standard techniques. We conclude the following theorem:

Theorem 5.3. *Construction 5.1 is adaptive $\mathcal{L}^{\mathsf{adap}}_{\min}$-secure searchable encryption scheme for database of size N, with space $O(N \log N)$, locality $O(1)$ and read efficiency $O(1)$ in the random-oracle model.*

Adaptive security in the plain model. We note that the usage of random oracle can be avoided using somewhat similar ideas to our solution in Section 4.2. That is, we modify the scheme such that the token that the client sends does not give the server the ability to decrypt the elements of the resulting list, and these decryptions are performed by the client (i.e., the client does not send the key K_i). The encryption of n_i is performed using a one-time pad, similarly to Construction 4.3. However, in case where the client stores database of keywords/identifiers (and not keywords/documents), this results in one additional round of interaction, where in the second round the client sends the decrypted list as a list of documents' identifiers the server has to fetch from the database.

References

[ABK+99] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal. Balanced allocations. *SIAM Journal on Computing*, 29(1):180–200, 1999.

[ANS10] Y. Arbitman, M. Naor, and G. Segev. Backyard cuckoo hashing: Constant worst-case operations with a succinct representation. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science*, pages 787–796, 2010.

[BM05] A. Z. Broder and M. Mitzenmacher. Multidimensional balanced allocations. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 195–196, 2005.

[CGK+06] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, pages 79–88, 2006.

[CGK+11] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. *Journal of Computer Security*, 19(5):895–934, 2011.

[CGP+15] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart. Leakage-abuse attacks against searchable encryption. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security*, pages 668–679, 2015.

[CJJ+13] D. Cash, S. Jarecki, C. S. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology - CRYPTO '13*, pages 353–373, 2013.

[CJJ+14] D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *Proceedings of the 21st Annual Network and Distributed System Security Symposium*, 2014.

[CK10] M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *Advances in Cryptology - ASIACRYPT '10*, pages 577–594, 2010.

[CM05] Y.-C. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *Proceedings of the 3rd International Conference on Applied Cryptography and Network Security*, pages 442–455, 2005.

[CT14] D. Cash and S. Tessaro. The locality of searchable symmetric encryption. In *Advances in Cryptology - EUROCRYPT '14*, pages 351–368, 2014.

[DP08] M. Dietzfelbinger and R. Pagh. Succinct data structures for retrieval and approximate membership. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming*, pages 385–396, 2008.

[DP09] D. P. Dubhashi and A. Panconesi. Concentration of Measure for the Analysis of Randomized Algorithms. Cambridge University Press, 2009.

[Goh03] E. Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2003.

[Hag98] T. Hagerup. Sorting and searching on the word RAM. In *Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science*, pages 366–398, 1998.

[HMP01] T. Hagerup, P. B. Miltersen, and R. Pagh. Deterministic dictionaries. *Journal of Algorithms*, 41(1):69–85, 2001.

[KO12] K. Kurosawa and Y. Ohtaki. UC-secure searchable symmetric encryption. In *Proceedings of the 16th International Conference on Financial Cryptography and Data Security*, pages 285–298, 2012.

[KO13] K. Kurosawa and Y. Ohtaki. How to update documents verifiably in searchable symmetric encryption. In *Proceedings of the 12th International Conference on Cryptology and Network Security*, pages 309–328, 2013.

[KP13] S. Kamara and C. Papamanthou. Parallel and dynamic searchable symmetric encryption. In *Proceedings of the 16th International Conference on Financial Cryptography and Data Security*, pages 258–274, 2013.

[KPR12] S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *Proceedings of the 19th ACM Conference on Computer and Communications Security*, pages 965–976, 2012.

[LP09] Y. Lindell and B. Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009.

[Mil99] P. B. Miltersen. Cell probe complexity - a survey. In *Proceedings of the 19th Conference on the Foundations of Software Technology and Theoretical Computer Science, Advances in Data Structures Workshop*, 1999.

[MRS00] M. Mitzenmacher, A. W. Richa, and R. Sitaraman. The power of two random choices: A survey of techniques and results. In *Handbook of Randomized Computing*, pages 255–312, 2000.

[MU05] M. Mitzenmacher and E. Upfal. Probability and computing – randomized algorithms and probabilistic analysis. Cambridge University Press, 2005.

[PP08] A. Pagh and R. Pagh. Uniform hashing in constant time and optimal space. *SIAM Journal on Computing*, 38(1):85–96, 2008.

[PR04] R. Pagh and F. F. Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122–144, 2004.

[SWP00] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *Proceedings of the 21st Annual IEEE Symposium on Security and Privacy*, pages 44–55, 2000.

[vLSD+10] P. van Liesdonk, S. Sedghi, J. Doumen, P. H. Hartel, and W. Jonker. Computationally efficient searchable symmetric encryption. In *Proceedings of 7th VLDB Workshop on Secure Data Management*, pages 87–100, 2010.