Cryptogr. Commun. (2018) 10:139–158
DOI 10.1007/s12095-017-0232-y

Searchable symmetric encryption over multiple servers

Geong Sen Poh[1] · **Moesfa Soeheila Mohamad**[1] ·
Ji-Jian Chin[2]

Received: 30 November 2016 / Accepted: 26 May 2017 / Published online: 7 June 2017

Abstract Searchable Symmetric Encryption (SSE) allows a user to store encrypted documents on server(s) and later efficiently searches these documents in a private manner. So far most existing works have focused on a single storage server. Therefore in this paper we consider the natural extension of SSE to *multiple servers*. We believe it is of practical interest, given that a user may choose to distribute documents to various cloud storage that are now readily available. The main benefit compared to a single server scheme is that a server can be set to hold only subset of encrypted documents/blocks. A server learns only content of documents/blocks that it stores in the event of successful leakage attack or ciphertext cryptanalysis, provided servers do not collude. We define formally an extension of single server SSE to multiserver and instantiate provably secure schemes that provide the above feature. Our main scheme hides total number of documents and document size even after retrieval, achieving less leakages compared to prior work, while maintaining sublinear search time for each server. We further study leakages under the new setting of non-colluding and colluding servers.

Keywords Searchable encryption · Cloud privacy · Secure storage

Mathematics Subject Classification (2010) 94A60

1 Introduction

Outsourcing documents to third-party storage providers allows the providers to learn the content of the documents. In order to protect data privacy, a user may encrypt the documents

This article is part of the Topical Collection on Recent Trends in Cryptography.

✉ Geong Sen Poh
gspoh@mimos.my

[1] MIMOS Berhad, Technology Park Malaysia, 57000 Kuala Lumpur, Malaysia

[2] Faculty of Engineering, Multimedia University, 63100 Cyberjaya, Selangor, Malaysia

before submission, but straightforward encryption eliminates the capability of searching them. Searchable symmetric encryption (SSE) was proposed as a practical mechanism to address this issue. An SSE scheme encrypts documents in such a way that the output contains encrypted data and their associated metadata. Through querying the metadata the scheme returns references to the matching encrypted data, which can then be retrieved and decrypted locally. Existing (dynamic) SSE schemes mainly concentrate on settings for single-user single server [4, 6, 7, 13, 15, 19, 20, 22, 25, 28, 30, 32], multi-user single server [5, 10, 11, 18, 24], and two servers [3, 16, 29] where one of the servers serves as a query proxy. As far as we know, to date there is no proposal involving multiple ($\geq$ 2) providers where every provider stores only a portion of the documents and the associated metadata. Such a construction is of interest since many storage providers are readily available, i.e. Dropbox, Google Drive, Amazon S3 and Microsoft OneDrive. Given these choices, a user can choose to distribute searchable encrypted documents/blocks to many of them instead of one, so that no single provider will have the full set of encrypted documents. Each provider only holds a subset of blocks of an encrypted document. This cannot be achieved in a single provider setup. It also allows new constructions with potentially better hiding properties. For ease of description, we refer to storage providers as servers in the subsequent sections.

Our contributions We extend the notion of SSE to SSE over multiple servers, and instantiate multiserver schemes with first a generic construction, then subsequently another construction using similar structures of recent schemes [5, 7]. We also describe how existing schemes can be extended to multiserver scheme straightforwardly under the generic construction, using [5]'s scheme as an example, demonstrating the flexibility of our definition. In fact, this also shows the scalability of most of the existing schemes. Our second construction divides documents into blocks, with the blocks SSE-encrypted and randomly assigned to different servers in such a way that, with high probability, no single server alone holds all encrypted blocks of a document. Hence, only partial information of a document stored in a server is revealed if the server successfully cryptanalysed the encrypted blocks, provided servers do not collude. It provides better leakage profile than existing schemes, in that document sizes and total number of documents are hidden even after retrieval, while maintaining sublinear search time for each server. Instead of creating an index that links keywords directly to document identifiers as in most existing schemes, we create index tables that link keywords to block-server identifier pairs where a block referenced by an identifier in server i may be stored in a different server j. Because of the decoupling between the block identifiers and the exact locations of stored blocks, a server cannot categorise with certainty a group of encrypted identifiers that map to a set of actual blocks during retrieval. In fact, most schemes directly reveal the matching identifier-document pairs during retrieval, which allow for straightforward categorisation of tokens that match the queried documents. We prove security in the setting of colluding and non-colluding servers under the real-ideal simulation paradigm proposed in [10] and its generalisation in [4, 7].

2 Related works

Song et al. [30] introduced SSE schemes. Their schemes encrypt words, in which during retrieval the words are first unmasked based on a query stream before decryption. Search time is linear since all words must be scanned. Indexing techniques were also described without concrete constructions. Goh [13] then proposed secure index suitable for SSE, but query privacy is not necessarily required. Using the simulation model, Chang and Mitzenmacher [6] proposed a linear SSE scheme secure for index and queries. Curtmola et al. [10] however showed that Chang and Mitzenmacher's definition is non-adaptive and can be satisfied with an insecure SSE. An improved definition encompassing non-adaptive and adaptive keyword attack was thus proposed, together with a scheme achieving sublinear search time using inverted index. Many subsequent proposals adopted Curtmola et al.'s definition and index structure. These include Chase and Kamara [7]'s schemes that support SSE over arbitrary data structures, and dynamic and parallel schemes proposed in [19] and [20]. We also adopt their security definitions and use similar data structures.

Following these schemes, which focus on single-keyword search, Cash et al. [4] introduced sublinear SSE schemes that support multi-keyword search for large datasets. This was extended to multi-user in [18] and parallelizable dynamic schemes in [5]. Naveed et al. [28] then proposed a blind storage scheme based on hash tables with linear probing that can be used for SSE. It does not leak the size of a document until it is queried. In their scheme, the server does not perform index operations. This is done by the user instead so that readily available commodity storage server can be used without any modification. Stefanov et al. [32] proposed a dynamic scheme with less leakage by combining the practicality of SSE with the hiding properties of ORAM.

More recently, two-server schemes termed as distributed SSE were proposed [3, 16, 29]. A proxy server is introduced in addition to the storage server. The idea is for the proxy to assist in queries so that the schemes do not leak search patterns, with the assumption of non-colluding servers. In contrast, our constructions use two or more servers with a somewhat different focus. Our aim is to distribute documents/blocks to many servers, in order to prevent any one server from possessing a complete set of documents or blocks of a document. Due to such distributions our schemes minimise leakages on search patterns though not fully-hiding as with the two-server schemes. Furthermore we consider colluding servers. Another related scheme is the scheme proposed in [24], in which distributed indexes are constructed so that they can be easily distributed to many devices under a cloud storage system for efficient search and retrieval.

Moataz and Shikfa [25] proposed a boolean scheme with probabilistic tokens to address the issue of leakage due to deterministic tokens of existing schemes. However the scheme search time is linear. Verifiable SSE schemes that protect against unauthorised modification were also proposed [8, 22, 23, 26]. Islam et al. [17] provided empirical analysis on access pattern leakages and proposed possible mitigation techniques. A related notion is Private Information Retrieval (PIR) [9]. It proposes private retrieval over servers with each server having an exact copy of the database. PIR schemes are mostly related to public databases instead of encrypted databases. We also examine multiple servers but focus on practicality where each server stores only partial information of the encrypted database. A simple, static construct in this setting was proposed in [27]. However it does not provide general framework that caters for existing single server schemes, and it leaks the number of documents matching a keyword after a query.

Fully hidden schemes can be achieved using primitives such as fully homomorphic encryption [12] and ORAM [14, 31], but both have yet to achieve the efficiency of SSE [32]. There are also public-key based schemes, which were introduced by Boneh et al. [1]. For a comprehensive survey, we refer interested readers to [2].

3 Definitions

We adopt the notation and security model from [4, 7, 10]. Our proposals involve a user and a set of distinct servers $\mathbf{S} = \{S_1, \ldots, S_s\}$, $s \in \mathbb{N}$. Let $\lambda \in \mathbb{N}$ be the security parameter and $||$ as concatenation. The user owns a set of documents, $\mathbf{D} = \{D_1, D_2, \ldots, D_n\}$, and each document D_f is either a complete document or a document divided into h equal-length blocks, $D_f = (d_{f,1}||1, d_{f,2}||2, \ldots, d_{f,h}||h)$ for $1 \leq f \leq n$, $\max(h) = l$ and $d_{f,i} \in \{0,1\}^{\phi(\lambda)}$ where ϕ is a polynomial function in λ. Also, we use $id_x \in \{0,1\}^{\lambda}$ to denote an identifier that uniquely identifies x. $\mathbf{D}$ is stored in a database with index $\mathtt{DB}(id_{D_f}, W_f)_{f=1}^{n}$, where $W_f = \{w_{f,1}, w_{f,2}, \ldots, w_{f,t_f}\}$ denotes the set of keywords matching document D_f and $w_{f,j} \in \{0,1\}^*$. The list of documents matching a keyword w is denoted as $\mathtt{DB}(w)$ while the total number of identifiers matching all keywords is $N = \Sigma_{w \in \mathbf{W}} |\mathtt{DB}(w)|$, where $\mathbf{W} = \bigcup_{f=1}^{n} W_f$. We further denote $\{1, 2, \ldots, n\}$ as $[n]$, $x \leftarrow A$ to mean x is an output of an algorithm A and $x \overset{R}{\leftarrow} X$ to mean random selection of a value x from a set X. We use $\mathtt{negl}(x)$ to denote a negligible function. We assume a semi-honest adversary $\mathcal{A}$. For colluding servers, in the worst case we assume $\mathcal{A}$ takes full control of all servers, where servers communicate among themselves to learn information from the scheme.

$\mathcal{E}$ and F. A randomised symmetric encryption scheme $\mathcal{E} = (\mathtt{Gen}, \mathtt{Enc}, \mathtt{Dec})$ consists of three PPT algorithms. $\mathtt{Gen}$ takes λ and outputs a secret key K; $\mathtt{Enc}$ takes K and a message $d \in \{0,1\}^*$ and outputs a ciphertext c; For all K from $\mathtt{Gen}$ and $d \in \{0,1\}^*$ we have $\mathtt{Dec}(K, \mathtt{Enc}(K, d)) = d$ with probability 1. We say $\mathcal{E}$ is IND-CPA if for all PPT adversary $\mathcal{A}$,

$$\mathbf{Adv}_{\mathcal{E},\mathcal{A}}^{ind-cpa}(\lambda) = |\Pr[\mathcal{A}^{K \leftarrow \mathtt{Gen}(1^{\lambda}), c \leftarrow \mathtt{Enc}(K,d)} = 1] - \Pr[\mathcal{A}^{c \overset{R}{\leftarrow} \{0,1\}^*} = 1]|$$

is negligible. A function $F : \{0,1\}^{\lambda} \times \{0,1\}^* \rightarrow \{0,1\}^{\lambda}$ from $\mathcal{F}$ the set of all functions $\{0,1\}^* \rightarrow \{0,1\}^{\lambda}$ is pseudo-random if for all PPT adversary $\mathcal{A}$,

$$\mathbf{Adv}_{F,\mathcal{A}}^{prf}(\lambda) = |\Pr[\mathcal{A}^{F(K,.), K \overset{R}{\leftarrow} \{0,1\}^{\lambda}} = 1] - \Pr[\mathcal{A}^{g(.), g \overset{R}{\leftarrow} \mathcal{F}} = 1]|$$

is negligible. Detailed treatments of these primitives can be found in [21]. Index tables (or dictionaries) in our discussion denote data structure of the form $I[key] = value$. Given a *key*, the *value* matching the *key* is returned.

Definition 1 We define a single-server SSE scheme $\mathtt{SSE}$ as consisting of the following algorithm and protocols:

– $(K, \gamma, \mathbf{c}) \leftarrow \mathtt{Setup}(\mathtt{DB}, aux)$. It outputs a secret key $K \overset{R}{\leftarrow} \{0,1\}^{\lambda}$, takes $\mathtt{DB}$ and aux, and outputs an encrypted index γ using K, where aux can be an empty set, a document set $\mathbf{D}$ and/or a lists of auxiliary information. If aux includes $\mathbf{D}$ then $\mathtt{Setup}$ encrypts documents in $\mathbf{D}$ using $\mathcal{E}$ and also outputs a set of ciphertexts $\mathbf{c}$. Else $\mathbf{c}$ is an empty set.
– $\mathbf{J} \leftarrow \mathtt{Search}(K, w, \gamma)$. A protocol executed between a user and a server. At the user, the protocol takes as inputs K, a keyword $w \in \{0,1\}^*$ and returns a search token. At the server, the protocol takes as inputs the search token and γ, and returns a set of encrypted identifiers, $\mathbf{J}$.
– $\varsigma \leftarrow \mathtt{Update}(K, \gamma, \mathtt{op}, \mathtt{in})$. A protocol (for dynamic scheme) executed between a user and a server. At the user, the protocol takes as inputs K, an operation $\mathtt{op}$ (e.g. add or delete a document), an input $\mathtt{in}$ and outputs results to the server (e.g. encrypted index entry for a newly added document and its ciphertext). At the server, the protocol takes as inputs the results from the user and γ, and outputs a result ς, which contains an updated index γ' and optionally other information such as a revocation list.

The scheme is said to be correct if `Search` returns the correct encrypted identifiers that match the documents containing the query keyword w. Under the real-ideal paradigm and a leakage profile $\mathcal{L}_{\mathtt{SSE}} = (\mathcal{L}_{\mathtt{SSE}}^{setup}, \mathcal{L}_{\mathtt{SSE}}^{query})$ (and $\mathcal{L}_{\mathtt{SSE}}^{update}$ for dynamic scheme), a `SSE` scheme is $\mathcal{L}_{\mathtt{SSE}}$-secure against adaptive chosen keyword attacks if for all PPT adversary $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that $\mathbf{Adv}_{\mathtt{SSE},\mathcal{A},\mathcal{S}}(\lambda) = |\Pr[\mathbf{Real}_{\mathtt{SSE},\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{Ideal}_{\mathtt{SSE},\mathcal{A},\mathcal{S}}(\lambda) = 1| \leq \mathtt{negl}(\lambda)$ where the game is as follows:

$\mathbf{Real}_{\mathtt{SSE},\mathcal{A}}(\lambda)$. $\mathcal{A}$ gives the challenger `DB` and *aux* (e.g. **D**). The challenger executes `Setup`, where the resulting γ (and **c**) are given to $\mathcal{A}$. $\mathcal{A}$ then makes a polynomial number of adaptive queries to the `Search` protocol, and for dynamic scheme, the `Update` protocol. For each query the challenger returns the query (and update) result to $\mathcal{A}$. Finally $\mathcal{A}$ returns a bit b as the output of the experiment.

$\mathbf{Ideal}_{\mathtt{SSE},\mathcal{A},\mathcal{S}}(\lambda)$. $\mathcal{A}$ outputs `DB` and *aux* (e.g. **D**). The simulator $\mathcal{S}$ simulates γ (and **c**) based on the leakage information from $\mathcal{L}_{\mathtt{SSE}}^{setup}$ on index, and gives γ (and **c**) to $\mathcal{A}$, who then makes a polynomial number of adaptive queries. The simulator $\mathcal{S}$ returns the query result for every query (and update result) based on $\mathcal{L}_{\mathtt{SSE}}^{query}$ (and $\mathcal{L}_{\mathtt{SSE}}^{update}$) to $\mathcal{A}$. Finally $\mathcal{A}$ returns a bit b as the output of the experiment.

We note that security against non-adaptive chosen keyword attacks can be defined in a similar way, except that $\mathcal{A}$ must prepare all queries beforehand. In summary, the security intuition of a SSE scheme is such that nothing is leaked except for the outputs and the patterns of a sequence of queries [7, 10]. For the outputs, $\mathcal{L}_{\mathtt{SSE}}^{setup}$ measures leakage of information stored on the server, which includes the list of encrypted items $\mathbf{c} = (\mathcal{E}(\delta_1), \ldots, \mathcal{E}(\delta_m))$, list of identifiers of the items $(id_{\delta_1}, \ldots, id_{\delta_m})$, items' sizes $(|\delta_1|, \ldots, |\delta_m|)$, $m \in \mathbb{N}$, and the index γ. As for leakage on search, after q queries, $\mathcal{L}_{\mathtt{SSE}}^{query}$ returns the search results $(\mathtt{DB}(w_1), \ldots, \mathtt{DB}(w_q))$, the search pattern $\mathrm{SP}(w_q)$ measuring whether a query is repeated, and the intersection pattern $\mathrm{IP}(w_q)$ measuring whether a same item is accessed. $\mathcal{L}_{\mathtt{SSE}}^{update}$ measures leakage during updates where it gives $\mathcal{E}(\delta_u)$, the item size $|\delta_u|$, and an updated index table γ', where δ_u is the item added/deleted. Here δ_i may represent a document D_f or a block of a document $d_{i,f}$.

4 `ms-SSE`

We now extend `SSE` to one that works over multiple servers, `ms-SSE`. A straightforward approach is to deploy an existing `SSE` scheme directly to cater for s servers by first grouping the set of documents into s (disjoint) subsets. Each of the encrypted subsets and metadata are sent to the respective servers. In other words, we independently run s rounds of a `SSE` scheme to outsource the subsets of documents to their selected servers. To search, we create a query and broadcast it to all the servers to retrieve the matching documents. The benefit is that a non-colluding server does not have information on other encrypted documents not stored in it and obviously does not know the total number of documents. In case of statistical analysis or ciphertext cryptanalysis, the server only learns the content of the documents it holds. However this information is revealed if the servers collude. Regardless of whether the servers collude or not, the subset of the encrypted documents is in the possession of the server. Also, for each document, the document size is known.

Alternatively, we can do this in a way that a server does not hold a complete document even in the encrypted form, so that only partial information of a document is revealed, potentially through statistical analysis on the leakages or successful cryptanalysis on the encrypted documents, provided servers do not collude. This approach can be utilised to also hide the document sizes, total number of documents and probabilistically hide pairs of matching tokens and documents. The idea is to work on blocks of a document instead, and randomly distributing the blocks to different servers.

We shall construct schemes based on these two approaches. The scheme using the straightforward approach demonstrates the scalability of existing single server schemes, whereas the scheme using the second approach offers smaller leakage.

Definition 2 A multi-server SSE scheme *($\mathtt{ms\text{-}SSE}$)* consists of:

– $(\mathbf{K}, \mathbf{I}, \mathbf{c}) \leftarrow \mathtt{mSetup}(\mathtt{DB}, \mathbf{D}, \mathbf{S})$: A probabilistic algorithm that takes $\mathtt{DB}$, a set of documents $\mathbf{D}$ and a list of servers $\mathbf{S}$, to output a list of secret keys $\mathbf{K}$, where each key is randomly chosen from $\{0, 1\}^{\lambda}$, a set of encrypted index tables $\mathbf{I}$ and a sequence of encrypted data $\mathbf{c}$.
– $\mathbf{J}^{w} := \mathtt{mSearch}(\mathbf{K}, w, \mathbf{I}, \mathbf{S})$: A protocol executed between a user and a set of servers. At the user, where on input $\mathbf{K}$, a query word w and $\mathbf{S}$, the protocol outputs a set of tokens (or keys). At the servers, using these tokens (or keys) and $\mathbf{I}$ as inputs, the protocol outputs a set of identifiers $\mathbf{J}^{w}$.
– $\varsigma := \mathtt{mUpdate}(\mathbf{K}, \mathtt{op}, \mathtt{in}, \mathbf{I}, \mathbf{S})$: A protocol (for dynamic scheme) executed between a user and a set of servers. At the user, where on input $\mathbf{K}$, a tuple *($\mathtt{op}, \mathtt{in}$)*, and $\mathbf{S}$, the protocol outputs results to the servers (e.g. encrypted index entry for a newly added document and its ciphertext), where $\mathtt{op} \in \{\mathtt{Doc+}, \mathtt{Doc-}\}$. At the servers, given the inputs from the user and $\mathbf{I}$, the protocol outputs ς. For $\mathtt{Doc+}$, $\varsigma = (\mathbf{I}', \mathbf{c}_u)$ and optionally other information, where $\mathbf{I}'$ is a set of updated indexes and $\mathbf{c}_u$ denotes the set of ciphertexts of the added documents. For $\mathtt{Doc-}$, $\varsigma = \mathbf{I}'$, and also possibly other information.

We say that $\mathtt{ms\text{-}SSE}$ is *correct* if for all λ, for all $\mathtt{DB}$, for all $\mathbf{D}$, for all $\mathbf{S}$, for all $(\mathbf{K}, \mathbf{I}, \mathbf{c}) \leftarrow \mathtt{mSetup}(\mathtt{DB}, \mathbf{D}, \mathbf{S})$, for all $\varsigma \leftarrow \mathtt{mUpdate}(\mathbf{K}, \mathtt{op}, \mathtt{in}, \mathbf{I}, \mathbf{S})$, $\mathtt{mSearch}(\mathbf{K}, w, \mathbf{I}, \mathbf{S})$ returns the set of identifiers $\mathbf{J}^{w}$ such that the set of decrypted documents have their identifiers in $\mathtt{DB}(w)$.

Note that unlike the case of $\mathtt{SSE}$, a list of servers $\mathbf{S}$ is required as input to $\mathtt{mSetup}$ in addition to $\mathtt{DB}$. We also set $\mathbf{D}$ in place of *aux*, where $\mathtt{mSetup}$ processes $\mathbf{D}$ into structures that allow for distributions to many servers. $\mathtt{mUpdate}$ may further include adding a new server $\mathtt{Svr+}$ and removing an existing server $\mathtt{Svr-}$, which require new processes and cannot rely on existing update protocol of $\mathtt{SSE}$. We shall briefly discuss how $\mathtt{Svr+}$ and $\mathtt{Svr-}$ can be performed in the later section.

In the $\mathtt{ms\text{-}SSE}$, the leakage function is defined per server and written as $\mathcal{L}_{\mathtt{ms\text{-}SSE}} = \left(\mathcal{L}^{setup}_{\mathtt{ms\text{-}SSE},S_j}, \mathcal{L}^{query}_{\mathtt{ms\text{-}SSE},S_j}, \mathcal{L}^{update}_{\mathtt{ms\text{-}SSE},S_j}\right)_{S_j \in \mathbf{S}}$. Accordingly, the adversary's power is differentiated by whether the adversary controls one server or controls multiple servers. We call the adversary who controls exactly one server as non-colluding server, and an adversary who controls more than one server as colluding server. This is reflected in the $\mathcal{L}$-security $\mathbf{Real}_{\mathtt{ms\text{-}SSE}}$ and $\mathbf{Ideal}_{\mathtt{ms\text{-}SSE}}$ in which the adversary receives query responses for only the servers under its control. Hence the security model for $\mathtt{ms\text{-}SSE}$ as follows.

Definition 3 For $\mathtt{ms\text{-}SSE}$ parameterised by security parameter λ and leakage functions $\mathcal{L}_{\mathtt{ms\text{-}SSE}} = \left(\mathcal{L}^{setup}_{\mathtt{ms\text{-}SSE},S_j}, \mathcal{L}^{query}_{\mathtt{ms\text{-}SSE},S_j}\right)_{S_j \in \mathbf{S}}$ (and $\mathcal{L}^{update}_{\mathtt{ms\text{-}SSE},S_j}$ for dynamic scheme), and $\mathcal{S}$ a simulator, an adversary $\mathcal{A}$ who controls a set of servers $\mathbf{S}' \subseteq \mathbf{S}$ play the following game with the challenger. The challenger randomly choose to play the $\mathbf{Real}_{\texttt{ms-SSE}}$ or $\mathbf{Ideal}_{\texttt{ms-SSE}}$ game, and sends the list of servers $\mathbf{S}$ to $\mathcal{A}$.

$\mathbf{Real}_{\texttt{ms-SSE},\mathcal{A}}(\lambda)$. In receiving $\mathbf{S}$ from the challenger, $\mathcal{A}$ generates a set of documents $\mathbf{D}$ and a database index $\texttt{DB}$. $\mathcal{A}$ gives the challenger $(\texttt{DB}, \mathbf{D}, \mathbf{S}')$. The challenger executes $\texttt{mSetup}$ resulting in $(\mathbf{K}, \mathbf{I}, \mathbf{c})$ where $\mathbf{K}$ is a set of secret keys, $\mathbf{I} = \{\gamma_{S_j}\}$, $\mathbf{c} = \{\mathbf{c}_{S_j}\}$ for all $S_j \in \mathbf{S}$. The challenger returns to $\mathcal{A}$ $(\mathbf{I}', \mathbf{c}')$ where $\mathbf{I}' = \{\gamma_{S_j}\}$ and $\mathbf{c}' = \{\mathbf{c}_{S_j}\}$ for all $S_j \in \mathbf{S}'$. Then $\mathcal{A}$ makes a polynomial number of adaptive queries. For search queries of keyword w, the output of $\texttt{mSearch}(\mathbf{K}, w, \mathbf{I}, \mathbf{S})$ for every $S_j \in \mathbf{S}'$, $\mathbf{J}^w_{S_j}$, is returned to $\mathcal{A}$. For dynamic $\texttt{ms-SSE}$, the update queries for document addition or removal is replied with output of $\texttt{mUpdate}(\mathbf{K}, \texttt{op}, \texttt{in}, \mathbf{I}, \mathbf{S})$ for every $S_j \in \mathbf{S}'$,the updated $(\gamma_{S_j}, \mathbf{c}_{S_j})$. Finally $\mathcal{A}$ returns a bit b as the output of the experiment.

$\mathbf{Ideal}_{\texttt{ms-SSE},\mathcal{A},\mathcal{S}}(\lambda)$. In receiving S from the challenger, $\mathcal{A}$ generates a set of documents $\mathbf{D}$ and a database index $\texttt{DB}$ and submits $(\texttt{DB}, \mathbf{D}, \mathbf{S}')$ to the challenger. The challenger then sends $\mathcal{L}^{setup}_{\texttt{ms-SSE},S_j}(\texttt{DB}, \mathbf{D}, \mathbf{S}')$ for $S_j \in \mathbf{S}'$ to the simulator $\mathcal{S}$. Then $\mathcal{S}$ produces $(\mathbf{I}', \mathbf{c}')$ where $\mathbf{I}' = \{\gamma_{S_j}\}$ and $\mathbf{c}' = \{\mathbf{c}_{S_j}\}$ for all $S_j \in \mathbf{S}'$ and this is returned to $\mathcal{A}$. After that, $\mathcal{A}$ makes polynomial number of adaptive queries for each of which $\mathcal{S}$ is given $\mathcal{L}^{query}_{\texttt{ms-SSE},S_j}(q_i)$ or $\mathcal{L}^{update}_{\texttt{ms-SSE},S_j}(q_i)$ for all $S_j \in \mathbf{S}'$. Outputs of $\mathcal{S}$ is given to $\mathcal{A}$ after each query. Finally $\mathcal{A}$ returns a bit b as the output of the experiment.

We define the advantage of the adversary as

$$\mathbf{Adv}_{\texttt{ms-SSE},\mathcal{A},\mathcal{S}}(\lambda) = |\Pr[\mathbf{Real}\ _{\texttt{ms-SSE},\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{Ideal}_{\texttt{ms-SSE},\mathcal{A},\mathcal{S}}(\lambda) = 1|.$$

We say that $\texttt{ms-SSE}$ is $\mathcal{L}_{\texttt{ms-SSE}}$-secure against adaptive chosen keyword attacks by colluding servers if for all PPT adversaries $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that

$$\mathbf{Adv}_{\texttt{ms-SSE},\mathcal{A},\mathcal{S}}(\lambda) \le \texttt{negl}(\lambda).$$

The above definitions can also be seen as a generalisation of $\texttt{SSE}$ to multiple servers, in which we define $\texttt{ms-SSE}$ for single server by setting $\mathbf{S}$ to one server, $\mathbf{D}$ to an empty set, or the set of documents and/or any other auxiliary information.

Security-wise we argue that an $\texttt{ms-SSE}$ scheme is at least as secure as an $\texttt{SSE}$ scheme. Consider $\mathcal{A}$ who controls all servers $S_j \in \mathbf{S}$. Then $\mathcal{A}$ has access to the whole of $(\mathbf{I}_{S_j}, \mathbf{c}_{S_j})_{S_j \in \mathbf{S}}$. Suppose that the token keys and encryption keys for all servers are equal. This allows $\mathcal{A}$ to consolidate $(\mathbf{I}_{S_j})_{S_j \in \mathbf{S}}$ to form one index table I. Similarly, $\mathcal{A}$ can consolidate all responses to the queries. Consequently, $\mathcal{A}$ has the same information as an adversary of the corresponding $\texttt{SSE}$ scheme, and hence gain equal advantage. However, if $\mathcal{A}$ controls only $\mathbf{S}' \subset \mathbf{S}$, the consolidated index table and query results are not complete. Additionally, the $\texttt{ms-SSE}$ scheme uses different keys for each server which implies $\mathcal{A}$ has insignificant probability to correctly consolidate the index table or query results. Therefore, $\mathcal{A}$ has less information and hence less advantage than a single server SSE adversary.

5 $\texttt{ms-SSE}_G$: a generic construction

In this construction, $\texttt{ms-SSE}_G$, our intuition is to design steps in the $\texttt{Setup}$ algorithm that prepare documents so that one may deploy existing $\texttt{SSE}$ for multiple servers with minimal modification. The general idea is to divide the set of documents into subsets of documents in such a way that an existing single keyword scheme can be used directly for index creations,

```
$(\mathbf{K}, \mathbf{I}, \mathbf{c}) \leftarrow \mathtt{mSetup}(\mathtt{DB}, \mathbf{D}, \mathbf{S})$:
1. Initialise empty index table $(\mathtt{DB}_{S_j})_{j=1}^{s}$ for $S_j \in \mathbf{S}$, empty list $\mathbf{c}$ and $\mathbf{I} = (\gamma_{S_1}, \ldots,$
   $\gamma_{S_s})$.
2. Partition $\mathbf{D}$ into $s$ subsets $\{\mathbf{D}_{S_j}\}_{j=1}^{s}$.
3. For $w_q \in \mathbf{W}$, $q = 1$ to $|\mathbf{W}|$:
       For each $id_{D_f} \in \mathtt{DB}(w_q)$, $f \in [n]$:
           For $j = 1$ to $s$: if $D_f$ in $\mathbf{D}_{S_j}$ add $id_{D_f}$ to $\mathtt{DB}_{S_j}(w_q)$.
4. For $j = 1$ to $s$:
   (a) $(K_{S_j}, \gamma_{S_j}, \mathbf{c}_{D_{S_j}}) \leftarrow \mathtt{SSE.Setup}(\mathtt{DB}_{S_j}, aux = \mathbf{D}_{S_j})$.
5. For $j = 1$ to $s$, send $(\gamma_{S_j}, \mathbf{c}_{D_{S_j}})$ to $S_j$.
6. Set $\mathbf{K} = \{K_{S_j}\}_{j=1}^{s}$, $\mathbf{I} = \{\gamma_{S_j}\}_{j=1}^{s}$ and $\mathbf{c} = \{\mathbf{c}_{D_{S_j}}\}_{j=1}^{s}$.
$\mathbf{J}^w := \mathtt{mSearch}(\mathbf{K}, w, \mathbf{I}, \mathbf{S})$:
1. For $j = 1$ to $s$: $\mathbf{J}_{S_j} \leftarrow \mathtt{SSE.Search}(K_{S_j}, w, \gamma_{S_j})$.
2. Set $\mathbf{J}^w = \{\mathbf{J}_{S_j}\}_{j=1}^{s}$.
$\varsigma := \mathtt{mUpdate}(\mathbf{K}, \mathtt{op}, \mathtt{in}, \mathbf{I}, \mathbf{S})$:
1. For $\mathtt{op} = \mathtt{Doc+}$: $j \xleftarrow{R} [s]$. $\varsigma \leftarrow \mathtt{SSE.Update}(K_{S_j}, \gamma_{S_j}, \mathtt{Doc+}, \mathtt{in})$.
2. For $\mathtt{op} = \mathtt{Doc-}$: For $j = 1$ to $s$: $\varsigma \leftarrow \mathtt{SSE.Update}(K_{S_j}, \gamma_{S_j}, \mathtt{Doc-}, \mathtt{in})$.
```

Fig. 1 $\mathtt{ms\text{-}SSE}_G$– A generic construction

queries, and updates for each subset of documents. It mirrors the straightforward approach discussed in Section 4. Figure 1 describes the construction.

We define `mSetup` as an algorithm that divides $\mathbf{D}$ into s subsets $\mathbf{D}_{S_j}$, and build the keyword index $\mathtt{DB}_{S_j}$. Based on the number of servers s, and given the tuple $(\mathbf{D}_{S_j}, \mathtt{DB}_{S_j})$, it runs `SSE.Setup` as if creating an index for each single server S_j. Similar concept applies to `mSearch` and `mUpdate`. In `mSearch`, the search token is broadcast to all servers in order to retrieve document identifiers that match the search token. As for `mUpdate`, a server is randomly selected to update a document, or a document identifier is broadcast to remove the document. $\mathtt{ms\text{-}SSE}_G$ can be setup based on existing schemes, such as using [5, 7, 28]. In brief, we replace the generic functions (e.g. `SSE.Setup`) with similar functions in the existing schemes. For example, using Cash et al.'s Π_{bas}^{dyn} [5, Section 4], we can construct a multiserver scheme by mapping `SSE.Setup` to Π_{bas}^{dyn}'s setup function and the underlying document encryption process, `SSE.Search` to the search function and `SSE.Update` to the update function, also coupled with the encryption process if the operation is to add a document.

Leakage, $\mathcal{L}_{\mathtt{ms\text{-}SSE}}$. Leakage is straightforward when servers *do not collude*.[1] It consists of the leakage of the underlying `SSE` scheme since a server only sees what is stored and the outputs generated from the `SSE` scheme. Nevertheless stored items and outputs are restricted to the subset of documents. In contrast, *colluding servers* learn the total number of documents by summing the number of encrypted documents stored in each server, since we divide $\mathbf{D}$ into s subsets $\mathbf{D}_{S_j}$. They also learn the total index size, in addition to the leakage of each of the server. During search and update query, these servers collectively learn the search and update results of one another and no other information is leaked except which server is holding which documents. If we consider these servers as partitions of the storage of a single server then the leakage is equivalent to that of a `SSE` scheme. So for colluding servers, $\mathcal{L}_{\mathtt{ms\text{-}SSE}} = (n, \{|\gamma_{S_j}|, \mathcal{L}_{\mathtt{SSE}} \text{ for } S_j | j = 1, \ldots, s\})$.

[1] We believe this is the case in most of the practical scenario, as a provider would not simply share customers' data with other providers.

Theorem 1 *Let* $\texttt{ms-SSE}_G$ *and* $\mathcal{L}_{\texttt{ms-SSE}}$ *be as defined above. Suppose* $\texttt{SSE}$ *is* $\mathcal{L}_{\texttt{SSE}}$*-secure against adaptive chosen keyword attack. Then,* $\texttt{ms-SSE}_G$ *is* $\mathcal{L}_{\texttt{ms-SSE}}$*-secure against adaptive attack by colluding servers.*

Proof This proof uses the game hopping strategy beginning with the single server $\mathcal{L}_{\texttt{SSE}}$-security game, followed by $\mathcal{L}_{\texttt{ms-SSE}}$ security game for non-colluding server and finally the game for colluding server. Here, $Pr\,[\mathsf{Game_i}]$ denotes the probability of the adversary winning $\mathsf{Game_i}$.

$\mathsf{Game_0}$: This game is exactly the $\mathcal{L}_{\texttt{SSE}}$-security game on $\texttt{SSE}$. Since we assume $\texttt{SSE}$ is $\mathcal{L}_{\texttt{SSE}}$-secure, for any PPT adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}^{\text{single}}$ such that $\mathbf{Adv}_{\;\texttt{SSE},\mathcal{A},\mathcal{S}^{\text{single}}}(\lambda) \leq \texttt{negl}(\lambda)$. So,

$$Pr\,[\mathsf{Game_0}] = \mathbf{Adv}_{\texttt{SSE},\mathcal{A},\mathcal{S}^{\text{single}}}(\lambda).$$

$\mathsf{Game_1}$: This is the $\mathcal{L}_{\texttt{ms-SSE}}$ security game where the adversary $\mathcal{B}_1$ controls one of the s servers, $S_b \in \mathbf{S}$. At the beginning of the game, $\mathcal{B}_1$ submits a set of n documents, $\mathbf{D}$. Let server S_b be assigned with subset $\mathbf{D}_{S_b}$ containing $n_b < n$ documents. Then the challenger sends to the simulator leakages of S_b, which is related to index, ciphertext and search results for $\mathbf{D}_{S_b}$ only.

Although the leakage is less than in the single server game, from the simulator's perspective the leakage is from a set of document, which is the same view as in the single server game. The difference in a multiserver setting is the fact that the keywords set is not disjoint and may be equal for all servers. This affects the index keys and the search token. However, in this scheme, every server executes an independent instance of $\texttt{SSE}$ including generating and using different keys. As a result, the index keys or search tokens of a keyword are different for different servers, and hence the index tables for the servers are disjoint. Outputs for search queries from different servers are independent because they search disjoint set of documents. It follows that, the simulated index table and search results for S_b can be constructed fully based on the leakages and does not need any information about outputs of other servers. Hence, the simulator $\mathcal{S}^{\text{single}}$ can be used in this game to simulate outputs for S_b with indistinguishable distribution from outputs of $\texttt{ms-SSE}_G$.

Since $\mathcal{B}_1$ obtain outputs related to $\mathbf{D}_{S_b} \subset \mathbf{D}$ only, he has less information than $\mathcal{A}$ in $\mathcal{L}_{\texttt{SSE}}$-security game. Hence,

$$Pr\,[\mathsf{Game_1}] < Pr\,[\mathsf{Game_0}]\,.$$

$\mathsf{Game_2}$: This is the $\mathcal{L}_{\texttt{ms-SSE}}$ security game where the adversary $\mathcal{B}_2$ controls a set of servers $\mathbf{S}' \subseteq \mathbf{S}$ of its choice. Let $\mathbf{S}' = \{S_1, \ldots, S_t\}$ where $t \leq s$. Since the instances of $\texttt{SSE}$ for the servers are independent, the simulator $\mathcal{S}^{\text{multi}}$ is a collection of $\texttt{SSE}$ simulators $\left\{\mathcal{S}_1^{\text{single}}, \ldots, \mathcal{S}_t^{\text{single}}\right\}$ taking as inputs the leakages for the respective servers. Here $\mathcal{B}_2$ obtains outputs related to t subsets of documents, which is more than that obtained by $\mathcal{B}_1$ in $\mathsf{Game_1}$. The adversary $\mathcal{B}_2$ would gain the same amount of information from the challenger as $\mathcal{A}$ in $\mathsf{Game_0}$ if it controls all servers, $\mathbf{S}' = \mathbf{S}$. By the independence of $\texttt{SSE}$ instances on every server, we have that

$$Pr\,[\mathsf{Game_2}] = \sum_{i=1}^{t} Pr\,[\mathsf{Game_1}] \leq Pr\,[\mathsf{Game_0}]\,.$$

Hence,

$$\mathbf{Adv}_{\texttt{ms-SSE},\mathcal{B}_2,\mathcal{S}^{\text{multi}}}(\lambda) = Pr\,[\mathsf{Game}_2] \leq \mathbf{Adv}_{\texttt{SSE},\mathcal{A},\mathcal{S}^{\text{single}}}(\lambda) \leq \texttt{negl}(\lambda).$$

Therefore, $\texttt{ms-SSE}_G$ is $\mathcal{L}_{\texttt{ms-SSE}}$-secure against adaptive attack by colluding servers. □

6 $\texttt{ms-SSE}_M$: a construction with smaller leakage

The previous scheme, $\texttt{ms-SSE}_G$, leaks the size and the total number of documents when servers collude. Even if the servers do not collude, the server learns content of the stored documents in the event of successful statistical analysis or key leaks. In order to address these limitations, we propose a scheme based on similar index structures proposed in [5, 7]. We first give a static scheme and then extend it to a dynamic one. The general idea is to divide documents into blocks, encrypt these blocks, distribute the blocks randomly to the servers. A masked index table γ_{S_j} is also created for every server S_j, which stores the linkage between the blocks and the servers. Then through the masked index tables a user queries for documents. Figure 2 provides details of the construction.

In brief, there is a two part process in the setup phase (mSetup). The first is to randomly assign blocks of documents to the list of servers. Given a set of documents, the scheme generates a master key, k. This master key is then used to generate a set of server keys K_{S_j} for all servers listed in the servers' list $\mathbf{S}$. Next, each document D_f in $\mathbf{D}$ is divided into a set of blocks, in which the number of blocks of D_f is denoted as h_f. Once this is done, each block of D_f is randomly assigned to a server S_j. The assignment is held by a temporary index T, in the form of a tuple of the document identifier id_{D_f}, the block identifier $id_{d_{f,i}}$,

```
$(\mathbf{K}, \mathbf{I}, \mathbf{c}) \leftarrow$ mSetup( DB, $\mathbf{D}$, $\mathbf{S}$):
1. Generate a random binary sequence of length $\lambda$ as the master key $k$.
2. For all $S_j \in \mathbf{S}$, generate server key $K_{S_j} \leftarrow F(k, id_{S_j})$.
3. Set $\mathbf{K} = \{k, \mathbf{K}_S\}$ for $\mathbf{K}_S = \{K_{S_1}, \ldots, K_{S_s}\}$.
4. For $f = 1$ to $n$, divide $D_f$ into blocks: $D_f = \{d_{f,1}||1,\ d_{f,2}||2, \ldots, d_{f,h_f}||h_f\}$.
5. Create a temporary index table $T$. For each $id_{D_f}$, $1 \leq f \leq n$:
       Given $d_{f,i}||i \in D_f$, $i = 1$ to $h_f$:
           $j \xleftarrow{R} [s]$. Set $T[id_{D_f}, i] = (id_{D_f}, id_{d_{f,i}}, id_{S_j})$.
6. Create empty index tables $\mathbf{I} = (\gamma_{S_1}, \ldots, \gamma_{S_s})$, and $\mathbf{c}$.
7. For $w_q \in \mathbf{W}$, $q = 1$ to $|\mathbf{W}|$:
       $K_{tok} \leftarrow F(k, 1||w_q)$ and $K_{eid} \leftarrow F(k, 2||w_q)$.
       Create temporary empty lists $\mathbf{t}_{S_1}, \ldots, \mathbf{t}_{S_s}$.
       For all $S_j \in \mathbf{S}$, $l_{q,j} \leftarrow F(K_{tok}, id_{S_j})$.
       For each $id_{D_f} \in \mathrm{DB}(w_q)$, $f \in [n]$:
           Given $D_f$ and $T[id_{D_f}, i]$, $i = 1$ to $h_f$:
               $c_{d_{f,i}} \leftarrow \mathcal{E}.\mathtt{Enc}(K_{S_j}, d_{f,i}||i)$. Set $\mathbf{c}[id_{D_f}, i] = c_{d_{f,i}}$.
               $j \xleftarrow{R} [s]$. $p_{d_{f,i}} \leftarrow \mathcal{E}.\mathtt{Enc}(K_{eid}, T[id_{D_f}, i])$.
               Append $p_{d_{f,i}}$ to $\mathbf{t}_{S_j}$.
       For every $S_j$, set $\gamma_{S_j}[l_{q,j}] = \mathbf{t}_{S_j}$.
8. Set $\mathbf{I} = (\gamma_{S_j})_{j=1}^{s}$. Submit $\gamma_{S_j}$ to $S_j$.
9. For all elements in $\mathbf{c}$, submit $\mathbf{c}[id_{D_f}, i]$ to $S_j$ based on $T[id_{D_f}, i]$.
$\mathbf{J}^w$ := mSearch($\mathbf{K}, w, \mathbf{I}$, $\mathbf{S}$):
1. $K_{tok} \leftarrow F(k, 1||w)$ and $K_{eid} \leftarrow F(k, 2||w)$. Broadcast $K_{tok}$ to all servers.
2. For each server $S_j$ $1 \leq j \leq s$: $\mathbf{p}_{w,j} \leftarrow \gamma_{S_j}[F(K_{tok}, id_{S_j})]$.
3. Set $\mathbf{J}^w = (\mathbf{p}_{w,1}, \ldots, \mathbf{p}_{w,s})$. (User uses $K_{eid}$ to decrypt element in $\mathbf{p}_{w,j}$ to retrieve
   document, block and server identifiers.)
```

Fig. 2 $\texttt{ms-SSE}_M$ – A construction with smaller leakage and the server identifier, id_{S_j}. Such a tuple effectively creates a "pointer" linking a block to the server that will eventually store the ciphertext of this block.

The second process is to encrypt blocks of documents, and to create masked indexes that link the keywords of the documents to the tuple in T in encrypted form, thus producing searchable encrypted indexes. This is realised by first creating a set of empty index tables $\mathbf{I}$ and list $\mathbf{c}$ that will be used to hold the indexes and ciphertexts. The index $\gamma_{S_j} \in \mathbf{I}$ for server S_j has a $\gamma_{S_j}[key] = value$ structure. The steps to generate γ_{S_j} are as follows. For a keyword w_q, a search token key, K_{tok}, and an identifier encryption key, K_{eid}, are created using F with the master key k and w_q as input. A list of empty vectors $\{\mathbf{t}_{S_1}, \ldots, \mathbf{t}_{S_s}\}$ are also created. $\mathbf{t}_{S_j}$ is used to temporarily hold encrypted index entries of T for server S_j until all entries in T have been processed. Entries in $\mathbf{t}_{S_j}$ serve as the *value* in γ_{S_j}. Next, search tokens, $l_{q,j}$, for w_q are generated for every server S_j through F with K_{tok} and the identifier of S_j as input. The search token serves as the *key* in γ_{S_j}. After that, for every document that contain w_q, which is identified based on the keyword index DB, each block of the document is encrypted using the server key K_{S_j} of server S_j that will eventually store the encrypted block. The selection of server key is done by referring to the entry $T[id_{D_f}, i]$, where i refers to block i. After this the entry $T[id_{D_f}, i]$ is encrypted using K_{eid}. The encrypted entry $p_{d_{f,i}}$ is randomly assigned to a server S_j, and appended to the vector $\mathbf{t}_{S_j}$. Once all documents containing w_q has been processed, the scheme sets $\gamma_{S_j}[l_{q,j}] = \mathbf{t}_{S_j}$. The above process is repeated for all keywords in $\mathbf{W}$.

Given $\gamma_{S_j}[l_{q,j}] = \mathbf{t}_{S_j}$, to search (mSearch), a user supplies $l_{q,j}$, generated using the master key k, w_q and the server identifier id_{S_j}. Each server S_j returns the respective list of encrypted indexes $\mathbf{p}_{w,j}$ ($\mathbf{t}_{S_j}$) if there is a match.

We note that random selections of servers ($j \overset{R}{\leftarrow} [s]$) during block assignments in index table T determine where each of the encrypted blocks is to be stored, while random selections during index creation chooses which index table γ_{S_j} the linkages are kept. It means a server i might, in its index, store an entry of a $(b\|D_f\|j)$ linkage, in which block b is in fact at server j instead of server i. This allows us to decouple the mapping between some of the entries in the index and the actual stored blocks. It prevents a server from learning the group of encrypted identifiers that matches the group of stored blocks during search, where the number of returned identifiers can be more/less than the actual blocks to be retrieved. However, this does allow a server to learn, if any, the number of blocks not stored in it, or the excess of stored blocks based on the differences between the number of encrypted identifiers in γ_{S_j} and the number of actual stored blocks. Nevertheless, we may pad the identifiers/blocks to hide this information if necessary.

Leakage, $\mathcal{L}_{\texttt{ms-SSE}}$ If *servers do not collude*, a server S_j possesses only the stored blocks, while γ_{S_j} reveals the number of keywords, $|\mathbf{W}|$, the number of block identifiers per keyword, $|\mathbf{p}_{q,j}|$, and the total number of block identifiers $P_j = \Sigma_{q=1}^{|\mathbf{W}|}|\mathbf{p}_{q,j}|$ in the server. In addition the server learns the differences (if any) between the number of identifiers and the number of stored blocks, which we denote as Δ. This means $\mathcal{L}_{\texttt{ms-SSE}}^{setup} = (|\mathbf{W}|, P_j, |\mathbf{p}_{q,j}|, \Delta)$. When a query is submitted, a server S_j learns $|\mathbf{p}_{q,j}|$ for a query w_q, and also the search pattern $\mathrm{SP}_{S_j}(w_q)$ by recording the number of times an identical keyword is queried. The server S_j also learns the intersection patterns $\mathrm{IP}_{S_j}(w_q)$ where the same block identifiers are submitted during retrieval of blocks when different keywords are queried. This means $\mathcal{L}_{\texttt{ms-SSE}}^{query} = (|\mathbf{p}_{q,j}|, \mathrm{SP}_{S_j}(w_q), \mathrm{IP}_{S_j}(w_q))$.

In the case where *servers collude*, the servers learn leakages of one another in addition to the single server leakage of γ_{S_j}. This means leakages of the total number of block identifiers

$\Sigma_{j=1}^{s} P_j$ based on γ_{S_j} for all j. However since keywords are differentiated as $F(K_{tok}, id_{S_j})$ for different server S_j, the servers will not be able to combine the encrypted block identifiers to link to the same keyword, until a keyword is queried. In summary, leakge for non-colluding server is $\mathcal{L}^{setup}_{\texttt{ms-SSE}} = (|\mathbf{W}|, \{(P_j, |\mathbf{p}_{q,j}|, \Delta)|j = 1, \ldots, s\})$ and the leakage for colluding servers is $\mathcal{L}^{query}_{\texttt{ms-SSE}} = (\{(|\mathbf{p}_{q,j}|, \mathrm{SP}_{S_j}(w_q), \mathrm{IP}_{S_j}(w_q))|j = 1, \ldots, s\})$.

Theorem 2 *Let* $\texttt{ms-SSE}_M$ *and* $\mathcal{L}_{\texttt{ms-SSE}} = (\mathcal{L}^{setup}_{\texttt{ms-SSE}}, \mathcal{L}^{query}_{\texttt{ms-SSE}})_{S_j \in \mathbf{S}}$ *be as defined above. Suppose* F *is a secure pseudorandom function and* $\mathcal{E}$ *is an IND-CPA symmetric encryption scheme. Then* $\texttt{ms-SSE}_M$ *is* $\mathcal{L}_{\texttt{ms-SSE}}$*-secure against adaptive attack by colluding server.*

Proof In $\texttt{ms-SSE}_M$ independent key sets are used for different servers which means related information has been rendered independent as it is distributed to different servers. This allows for independent simulators for each server to produce the required outputs in the colluding servers attack. Thus, this proof begins by defining a simulator for one of the servers (under non-colluding server attack) and shows that its output is indistinguishable from the output of $\texttt{ms-SSE}$. Finally, multiple instances of this simulator form the simulator for the colluding server attack.

First, we define a simulator $\mathcal{S}^{\mathrm{single}}$ for non-colluding server, $\mathcal{B}$, attack with server controlled by $\mathcal{B}$ being S_b.

For input $\mathcal{L}^{setup}_{\texttt{ms-SSE}} = (|\mathbf{W}|, P_b, |\mathbf{p}_{q,b}|, \Delta)$ $\mathcal{S}^{\mathrm{single}}$ produces $(\gamma'_b, \mathbf{c}'_b)$:

- Generate index γ'_b by first generating an encryption key $k'_b \xleftarrow{R} \mathcal{E}.\texttt{Gen}(1^\lambda)$. Then for each $q \in [|\mathbf{W}|]$, generate random strings $r_{q,0} \xleftarrow{R} \{0,1\}^\lambda$, and $\mathbf{R}_q = \{r_{q,i} \xleftarrow{R} \{0,1\}^{3(\lambda)} | i \in [|\mathbf{p}_{q,b}|]\}$. Then set $\gamma'_j[r_{q,0}] = \{\mathcal{E}.\texttt{Enc}(k', r_i) \,|\, r_i \in \mathbf{R}_q\}$.
- Generate ciphertexts by generating an encryption key $k'_{eid} \leftarrow \mathcal{E}.\texttt{Gen}(1^\lambda)$ and generating $|\mathbf{p}_{q,b}| + \Delta$ random binary strings, $r_i \xleftarrow{R} \{0,1\}^{\phi(\lambda)+\log_2 l}$.
- Then compute $\mathbf{c}'_b = \{\mathcal{E}.\texttt{Enc}\left(k'_{eid}, r_i\right) \,|\, i \in [|\mathbf{p}_{q,b}| + \Delta]\}$.

For input $\mathcal{L}^{query}_{\texttt{ms-SSE}} = (|\mathbf{p}_{q,b}|, \mathrm{SP}_{S_b}(w_q), \mathrm{IP}_{S_b}(w_q))$ $\mathcal{S}^{\mathrm{single}}$ produces query results which consists of set of block identifier ciphertexts, $\mathbf{J}'_b$, and a set of block ciphertexts, C'_b which the user retrieves:

- Prepare $\mathbf{J}'_b$ by first referring to $\mathrm{SP}_{S_b}(w_q)$ to determine whether the query has been made previously. If so, set $\mathbf{J}'_b$ with the same list as before. Otherwise, choose an unused entry in γ'_b containing a list of $|\mathbf{p}_{q,b}|$ items, and set $\mathbf{J}'_b$ as the list.
- Prepare C'_b by again referring to $\mathrm{SP}_{S_b}(w_q)$ to determine whether the query has been made previously. If so, set C'_b with the same set as before. Otherwise, prepare a new set. Refer $\mathrm{IP}_{S_b}(w_q)$ to determine ciphertexts in this query which has been returned as results in previous queries. If there is any, include the same ciphertexts in C'_b. Lastly, include in C'_b unused ciphertexts from $\mathbf{c}'_b$ until $|C'_b|$ equals the number of block identifiers submitted for retrieval.

Game_0: This is the $\mathbf{Real}_{\texttt{ms-SSE},\mathcal{B}}(\lambda)$ game with the adversary $\mathcal{B}$ controlling one server, S_b. So, $Pr\,[\mathsf{Game}_0] = Pr\left[\mathbf{Real}_{\texttt{ms-SSE},\mathcal{B}}(\lambda) = 1\right]$.

Game_1: This game is Game_0 with $F(k, \cdot)$ and $F(K_{tok}, \cdot)$ in the $\texttt{mSetup}$ replaced by independent random functions. Let where $\mathcal{B}_1$ be the adversary of the pseudorandom function.Then, we have that

$$Pr\,[\mathsf{Game}_1] - Pr\,[\mathsf{Game}_0] \leq 2\mathbf{Adv}^{prf}_{F,\mathcal{B}_1}(\lambda).$$

Game_2: This game is Game_1 with the encryption algorithm $\mathcal{E}.\mathtt{Enc}(K_{S_b}, r_1)$ and $\mathcal{E}.\mathtt{Enc}(K_{eid}, r_2)$ in Step 7 where r_1 and r_2 are random strings instead of the document block $d_{f,i}\|i$ and the identifiers $id_{D_f}\|id_{d_f,i}\|id_{S_b}$. Let $\mathcal{B}_2$ be the IND-CPA adversary of $\mathcal{E}$. We have that

$$Pr\left[\mathsf{Game}_2\right] - Pr\left[\mathsf{Game}_1\right] \leq 2\mathbf{Adv}^{ind-cpa}_{\mathcal{E},\mathcal{B}_2}(\lambda).$$

Game_3: This is the **Ideal** $_{\mathtt{ms\text{-}SSE},\mathcal{B},\mathcal{S}^{\text{single}}}(\lambda)$ with the simulator $\mathcal{S}^{\text{single}}$ as defined earlier. Clearly, this game is equivalent to Game_2. Hence,

$$Pr\left[\mathbf{Ideal}_{\mathtt{ms\text{-}SSE},\mathcal{B},\mathcal{S}^{\text{single}}}(\lambda)\right] = Pr\left[\mathsf{Game}_3\right] = Pr\left[\mathsf{Game}_2\right].$$

From Game_0 to Game_3 we have that

$$\begin{aligned}
&Pr\left[\mathbf{Ideal}_{\mathtt{ms\text{-}SSE},\mathcal{B},\mathcal{S}^{\text{single}}}(\lambda) = 1\right] \\
&= Pr\left[\mathsf{Game}_2\right] \\
&\leq \mathbf{Adv}^{ind-cpa}_{\mathcal{E},\mathcal{B}_2}(\lambda) + Pr\left[\mathsf{Game}_1\right] \\
&\leq \mathbf{Adv}^{ind-cpa}_{\mathcal{E},\mathcal{B}_2}(\lambda) + \mathbf{Adv}^{prf}_{F,\mathcal{B}_1}(\lambda) + Pr\left[\mathsf{Game}_0\right] \\
&\leq \mathbf{Adv}^{ind-cpa}_{\mathcal{E},\mathcal{B}_2}(\lambda) + \mathbf{Adv}^{prf}_{F,\mathcal{B}_1}(\lambda) + Pr\left[\mathbf{Real}_{\mathtt{ms\text{-}SSE},\mathcal{B}}(\lambda) = 1\right].
\end{aligned}$$

It follows that,

$$\mathbf{Adv}_{\mathtt{ms\text{-}SSE},\mathcal{B},\mathcal{S}^{\text{single}}}(\lambda) \leq \mathbf{Adv}^{prf}_{F,\mathcal{B}_1}(\lambda) + \mathbf{Adv}^{ind-cpa}_{\mathcal{E},\mathcal{B}_2}(\lambda).$$

By our assumptions, the right hand side of the inequality is $\mathtt{negl}(\lambda)$. Therefore, we conclude that $\mathtt{ms\text{-}SSE}$ is $\mathcal{L}_{\mathtt{ms\text{-}SSE}_M}$-secure against adaptive attack by non-colluding server.

Finally, consider an adversary $\mathcal{A}$ which controls a set of servers, $\mathbf{S}' \subseteq \mathbf{S}$. Let $\mathbf{S}' = \{S_1, \ldots, S_t\}$ where $t \leq s$. For adversary $\mathcal{A}$ consider a simulator $\mathcal{S}^{\text{multi}}$ for the $\mathbf{Ideal}_{\mathtt{ms\text{-}SSE}}$ game which is a collection of t instances of the single server simulator, $\{\mathcal{S}^{\text{single}}_1, \ldots, \mathcal{S}^{\text{single}}_t\}$. Each simulator acts on the given leakages independently. During the game, inputs $\mathcal{L}^{setup}_{\mathtt{ms\text{-}SSE}} = (|\mathbf{W}|, \{(P_j, |\mathbf{p}_{q,j}|, \Delta)|j = 1, \ldots, s\})$ and $\mathcal{L}^{query}_{\mathtt{ms\text{-}SSE}} = (\{(|\mathbf{p}_{q,j}|, \mathrm{SP}_{S_j}(w_q), \mathrm{IP}_{S_j}(w_q))|j = 1, \ldots, s\})$ to $\mathcal{S}^{\text{multi}}$ are distributed accordingly. In particular, for $j = 1, \ldots, t$, the leakage $(|\mathbf{W}|, \{(P_j, |\mathbf{p}_{q,j}|, \Delta)$ and $(\{(|\mathbf{p}_{q,j}|, \mathrm{SP}_{S_j}(w_q), \mathrm{IP}_{S_j}(w_q))$ is passed to $\mathcal{S}^{\text{single}}_j$.

Due to the independent server keys in $\mathtt{ms\text{-}SSE}_M$ and, the secure primitives $\mathcal{E}$ and F, outputs of each storage servers is disjoint. Hence, the collection of independent outputs by $\mathcal{S}^{\text{multi}}$ matches the distribution of $\mathtt{ms\text{-}SSE}_M$ outputs. Since we define $\mathcal{S}^{\text{multi}}$ to consists of simulators which work independently on their respective inputs, we have that

$$\mathbf{Adv}_{\mathtt{SSE},\mathcal{A},\mathcal{S}^{\text{multi}}}(\lambda) \leq s\left(\mathbf{Adv}^{prf}_{F,\mathcal{B}_1}(\lambda) + \mathbf{Adv}^{ind-cpa}_{\mathcal{E},\mathcal{B}_2}(\lambda)\right)$$

with equality achieved when $\mathcal{A}$ controls all s servers. Since the right hand side of the inequality is negligible by our assumptions, we conclude that $\mathtt{ms\text{-}SSE}_M$ is $\mathcal{L}_{\mathtt{ms\text{-}SSE}_M}$-secure against adaptive attack by colluding servers. □

If only a static scheme is required, then instead of encrypting every entry in T separately (required for efficient index updates), we could encrypt the whole list $T[id_f]$ for D_f. This prevents leakage on the number of blocks per keyword and the total number of blocks.

```
Add the following steps after Step 7 of $\texttt{ms-SSE}_M$ (Figure 2):
    Create temporary empty lists $(\mathbf{a}_{S_1}, \dots, \mathbf{a}_{S_s})$ and $(\mathbf{eb}_{S_1}, \dots, \mathbf{eb}_{S_s})$.
    For every $id_{D_f}$ in $T$:
        $K_{tok} \leftarrow F(k, 1||id_{D_f})$ and $K_{eid} \leftarrow F(k, 2||id_{D_f})$.
        $j \xleftarrow{R} [s]$. For every $i$ item in $T[id_{D_f}, i]$: Append $T[id_{D_f}, i]$ to $\mathbf{a}_{S_j}$.
        $\mathbf{e}_{S_j} \leftarrow \mathcal{E}.\texttt{Enc}(K_{eid}, \mathbf{a}_{S_j})$.
        Divides $\mathbf{e}_{S_j}$ into blocks where each block has length $|p_{d_{f,i}}|$ and store all
        blocks in $\mathbf{eb}_{S_j}$.
    For every $S_j$, set $\gamma_{S_j}[F(K_{tok}, id_{S_j})] = \mathbf{eb}_{S_j}$.
    For every $S_j$, permute entries in $\gamma_{S_j}$ to mix entries of keywords and document
    identifiers.
```

Fig. 3 $\texttt{ms-SSE}_M^{dyn}$ – Modification to $\texttt{mSetup}$ in $\texttt{ms-SSE}_M$

6.1 Extension to a dynamic construction

We propose a dynamic construction $\texttt{ms-SSE}_M^{dyn}$ that supports addition and deletion of documents. We first discuss the modification of $\texttt{mSetup}$ as shown in Fig. 3 to accommodate document removals since the existing index structure does not provide for efficient extraction of blocks. Our approach is to treat document identifiers id_{D_f} as keywords and create entries for them in γ_{S_j} through identifier-based keys $F(k, 1||id_{D_f})$ and $F(k, 2||id_{D_f})$. Using the keys, we add an entry to a randomly selected γ_{S_j}, encrypt the complete list of elements in T for id_{D_f} and then divide the list into blocks of length equal to $|p_{d_{f,i}}|$. Encryption of the whole list prevents leakage on the number of blocks in a document. By treating the identifiers as keywords, dividing the encrypted list into blocks and permuting the entries in γ_{S_j}, we ensure a server cannot differentiate between entries of keywords and document identifiers. If the entry has a much smaller number of blocks compared to the average number of blocks for other entries then dummy blocks can be included.

For *adding a document*, $\texttt{Doc+}$ (Fig. 4), the approach proposed in [5] is adapted. We initialise a list of frequency counters on every keyword of the document cnt_{w_q}, which forms

```
$\varsigma \leftarrow \texttt{mUpdate}(\mathbf{K}, \texttt{Doc+}, D_u, \mathbf{I}, \mathbf{S})$:
1. Setup $T$ following Step 4 and 5 of $\texttt{ms-SSE}_M$ (Figure 2).
2. Update index tables $\mathbf{I}$:
   (a) For $w_q \in \mathbf{W}_u$, $q = 1$ to $|\mathbf{W}_u|$:
           If $cnt_{w_q} = \perp$, then initialise $cnt_{w_q} = 0$ else $cnt_{w_q}$++.
           $K_{tok} \leftarrow F(k, 1||w_q)$ and $K_{eid} \leftarrow F(k, 2||w_q)$.
           Create temporary empty lists $\mathbf{t}_{S_1}, \dots, \mathbf{t}_{S_s}$.
           For all $S_j \in \mathbf{S}$, $l_{q,j} \leftarrow F(K_{tok}, id_{S_j}||cnt_{w_q})$.
           Given $D_u$ and $T[id_{D_u}, i]$, $i = 1$ to $h_u$:
               $c_{d_{u,i}} \leftarrow \mathcal{E}.\texttt{Enc}(K_{S_j}, d_{u,i}||i)$. Set $\mathbf{c}[id_{D_u}, i] = c_{d_{u,i}}$.
               $j \xleftarrow{R} [s]$. $p_{d_{u,i}} \leftarrow \mathcal{E}.\texttt{Enc}(K_{eid}, T[id_{D_u}, i])$.
               Append $p_{d_{u,i}}$ to $\mathbf{t}_{S_j}$.
           For every $S_j$, send $(l_{q,j}, \mathbf{t}_{S_j})$.
   (b) For $id_{D_u}$, create temporary empty lists $\mathbf{a}$ and $\mathbf{eb}$.
           $K_{tok} \leftarrow F(k, 1||id_{D_u})$ and $K_{eid} \leftarrow F(k, 2||id_{D_u})$.
           $j \xleftarrow{R} [s]$. For every $i$ item: Append $T[id_{D_u}, i]$ to $\mathbf{a}$.
           $\mathbf{e} \leftarrow \mathcal{E}.\texttt{Enc}(K_{eid}, \mathbf{a})$.
           Divides $\mathbf{e}$ into blocks and store all the blocks in $\mathbf{eb}$.
       For $S_j$, send $(F(K_{tok}, id_{S_j}), \mathbf{eb})$.
3. For $i = 1$ to $h_u$, submit $\mathbf{c}[id_{D_u}, i]$ to $S_j$ based on $T[id_{D_u}, i]$.
$\varsigma = \mathbf{I}' = (\gamma'_{S_1}, \dots, \gamma'_{S_s})$.
```

Fig. 4 $\texttt{ms-SSE}_M^{dyn}$ – Adding a new document

```
$\varsigma \leftarrow$ mUpdate($\mathbf{K}$, Doc-, $id_{D_x}$, $\mathbf{I}$, $\mathbf{S}$):
1. $K_{tok} \leftarrow F(k, 1||id_{D_x})$ and $K_{eid} \leftarrow F(k, 2||id_{D_x})$.
2. Broadcast $K_{tok}$ to all servers.
3. For each server $S_j$, $1 \leq j \leq s$,
       $\mathbf{p}_{D_x,j} \leftarrow \gamma_{S_j}[F(K_{tok}, id_{S_j})]$.
4. For the entry where $\mathbf{p}_{D_x,j} \neq \perp$, remove the entry in $\gamma_{S_j}$.
5. Return to user $(\mathbf{p}_{D_x,1}, \ldots, \mathbf{p}_{D_x,s})$.
6. User uses $K_{eid}$ to retrieves the document, block and server identifiers.
7. User uses the identifiers to instruct the servers to remove the encrypted blocks.
$\varsigma = \mathbf{I}' = \gamma'_{S_j}$ where $\gamma'_{S_j}$ is the index table updated with the entry on $id_{D_x}$ removed.
```

Fig. 5 $\texttt{ms-SSE}_M^{dyn}$ – Removing a document

part of the input to the token entry in γ_{S_j}. So instead of producing a token entry $l_{q,j} \leftarrow F(K_{tok}, id_{S_j})$ for γ_{S_j} (Step 7 of mSetup, Fig. 2), we produce $l_{q,j} \leftarrow F(K_{tok}, id_{S_j}||cnt_{w_q})$, where cnt_{w_q} is incremented each time w_q is processed for a new document. This is so that addition does not cause leakage before queries as some documents may share identical keywords.

For *removing a document,* Doc− (Fig. 5), the idea is to query the servers through document identifier tokens that have been indexed during mSetup and Doc+ as described earlier. Our scheme removes the actual encrypted blocks and the identifier entries in the index. We do not update the keyword/block pair indexes in γ_{S_j} but a deletion strategy similar to that of [28] can be adopted, in which when the searched encrypted blocks cannot be found, their entries in γ_{S_j} can be deleted. Also, communication bandwidth can be saved if it is possible for the user to store an index table on document identifier and server location so that the user only sends K_{tok} to the particular server that holds the index for id_{D_x}.

For query, mSearch requires minimal modification from the static scheme due to how Doc+ and Doc− protocols are constructed. Figure 6 describes the protocol. The only additional step is for each server to generate tokens based on a sequence of counter values $F(K_{tok}, id_{S_j}||cnt_w)$ until there are no more matching token. This is required since for Doc+, we use a cnt_{w_q} to differentiate similar keyword entries for different documents to prevent leakage before queries.

We further discuss how a server can be introduced or removed. Adding a new server Svr+ can be straightforward. The idea is to generate a new server key, and then update the key list $\mathbf{K}_S$ and the server list $\mathbf{S}$. Instead of modifying existing indexes and redistributing the encrypted blocks that have been stored, the new server will only be considered when adding new documents.

Removing an existing server S_j is more computationally intensive, as all blocks from S_j must be re-assigned. This can be done using existing protocol Doc+. The index γ_{S_j} must also be processed as it contains block linkages pointing to locations in other servers.

```
$\mathbf{J}^w$ := mSearch($\mathbf{K}, w, \mathbf{I}, \mathbf{S}$):
1. $K_{tok} \leftarrow F(k, 1||w)$ and $K_{eid} \leftarrow F(k, 2||w)$. Broadcast $K_{tok}$ to all servers.
2. For each server $S_j$, $1 \leq j \leq s$,
       $\mathbf{p}_{w,j} \leftarrow \gamma_{S_j}[F(K_{tok}, id_{S_j})]$, and
       for $cnt_w$ starts with 0 until $S_j$ returns $\perp$,
       $\mathbf{p}_{w,j,cnt_w} \leftarrow \gamma_{S_j}[F(K_{tok}, id_{S_j}||cnt_w)]$ .
3. Return $\mathbf{p}_{w,j}$ and $\mathbf{p}_{w,j,cnt_w}$ to users for $1 \leq j \leq s$.
```

Fig. 6 $\texttt{ms-SSE}_M^{dyn}$ – Modification to mSearch in $\texttt{ms-SSE}_M$

Such linkages can be decrypted, re-encrypted and redistributed randomly to the remaining servers. The mechanism is to append the re-generated linkages to the matching token entries of other servers S_k. Also, the index γ_{S_j} contains entries referencing documents and its blocks' locations. These can also be redistributed to another server. This completes the process but other servers S_k may still contain linkages that point to the blocks stored in S_j. In this case, the user archives the identifier of the removed server. During search if there is a linkage pointing to S_j, the user discards this linkage.

Leakage The leakage for `mSetup` is changed by the inclusion of list of document blocks as an entry in an index γ_{S_j}. It follows that the number of entries in γ_{S_j} no longer indicates the number of keywords, but it is the sum of number of keywords $|\mathbf{W}_j|$ and the number of documents $|\mathbf{D}_j|$ whose encrypted blocks reside on the server. Besides, the total number of block identifiers, P_j, and the difference between the number of block identifiers and the number of encrypted blocks, Δ, are no longer revealed. For server S_j,

$$\mathcal{L}^{setup}_{\texttt{ms-SSE}} = (|\mathbf{W}_j|+|\mathbf{D}_j|, \{|\mathbf{p}_{i,j}| \mid i = 1, \ldots, |\mathbf{W}_j|+|\mathbf{D}_j|\}).$$

The length of entry from the original index $|\mathbf{p}_{q,b}|$ remains in the search query leakage. Due to document addition, the number of index entries retrieved for a search query may be more than one. Since the server computes the additional keyword tokens by incrementing counters, the leakage is a list of index entries and their lengths corresponding to the counters, $(c, l_{c,j}, |\mathbf{p}_{c,b}|)_{c=1}^{cnt_{w_q}}$.

Another piece of leaked information in search query leakage is due to the delayed index update for document removal. When a document is deleted, the encrypted blocks of the document is identified and deleted immediately. The corresponding encrypted block identifiers in γ_{S_j} are deleted only when a keyword of the document is searched. Hence, the keyword belonging to the deleted document is hidden until it is searched. Nevertheless, if multiple documents have been removed before a search of the keyword, updating γ_{S_j} reveals that a deleted document that contained the keyword but does not reveal which document. Since the server finds and omits the encrypted block identifiers as sent by the user, the server would know the token of the entry from which they are omitted. As a result, the query leakage now contains the set of pairs $\{(\tau_i, |\mathbf{r}_i|) \mid 0 \leqslant i \leqslant cnt_{w_q}\}$ where $\mathbf{r}_i$ is the list of encrypted block identifiers omitted from $\gamma_{S_j}[\tau_i]$. The set may be empty and may have at most cnt_{w_q} pairs. Search query leakage for server, S_j is $\mathcal{L}^{query}_{\texttt{ms-SSE}} =$

$$\left(|\mathbf{p}_{q,b}|, (c, l_{c,j}, |\mathbf{p}_{c,b}|)_{c=1}^{cnt_{w_q}}, \mathrm{SP}_{S_j}(w_q), \mathrm{IP}_{S_j}(w_q), \{(\tau_i, |\mathbf{r}_i|) \mid 0 \leqslant i \leqslant cnt_{w_q}\}\right).$$

For document addition, a server S_j possesses partial number of blocks of the added document, $\mathbf{c}^u_{S_j}$, and learns the number of newly stored blocks $|\mathbf{c}^u_{S_j}|$. S_j also learns new keyword/blocks pairs added to γ_{S_j} and that these blocks belong to a document. We denote these pairs as $L_j = (l_{q,j}, |\mathbf{t}_{S_j}|)_{q=1}^{|\mathbf{W}_u|}$. If a newly added document contains a keyword that was searched before, then S_j learns this fact, since S_j can use the list of previously submitted query tokens $K_{tok} \leftarrow F(K, 1||w_q)$ together with cnt_{w_q} to try to match the entry in γ_{S_j}. We denote this pattern due to addition as $\mathbb{A}_j = \{(F(K, 1||w_q), cnt_{w_q}) \mid w_q \in \mathbf{W}_u\}$. We remark that the addition protocol can be readily extended to allow for addition of a batch of documents, in which then the information with regard to relation of blocks and document can be hidden in the case when many documents are added in a session. $\mathcal{L}^{update,\texttt{Doc}^+}_{\texttt{ms-SSE}} = \left(|\mathbf{c}^u_{S_j}|, L_j, \mathbb{A}_j\right)$.

Similarly for document removal, S_j learns the blocks and the number of blocks removed, and that the blocks belong to a document. In addition, when there is a match during deletion, S_j learns that the entry in γ_{S_j} is not a keyword entry. A list $\mathtt{R}_j = (v, l)$ where $v \in \{0, 1\}$ and $l := F(K_{tok}, id_{S_j})$ can be maintained, where if there is a match then we set $v = 1$. In summary $\mathcal{L}^{update,\mathtt{Doc}^-}_{\mathtt{ms\text{-}SSE}} = \left(|\mathbf{c}^x_{S_j}|, \mathtt{R}_j\right)$.

The colluding servers collectively learn all leakages of all servers. Notice that since the query token that is sent to all servers is the same, colluding servers knows when they are searching for the same keyword, and hence would learn the total number of block identifiers returned to the user. Similarly, for document addition, the adversary learns would know all of the documents block ciphertexts and encrypted identifiers. In short, the leakage gained by adversary who controls all of the servers is equal to the leakage obtained by adversary in a single server SSE.

Theorem 3 *Let* $\mathtt{ms\text{-}SSE}^{dyn}_M$ *and* $\mathcal{L}_{\mathtt{ms\text{-}SSE}} = \left(\mathcal{L}^{setup}_{\mathtt{ms\text{-}SSE}}, \mathcal{L}^{query}_{\mathtt{ms\text{-}SSE}}, \mathcal{L}^{update}_{\mathtt{ms\text{-}SSE}}\right)$ *be as defined above. Then,* $\mathtt{ms\text{-}SSE}^{dyn}_M$ *is* $\mathcal{L}_{\mathtt{ms\text{-}SSE}}$*-secure against non-adaptive chosen keyword attacks by colluding servers, assuming* F *is a secure pseudorandom function and* $\mathcal{E}$ *is an IND-CPA symmetric encryption scheme.*

Proof Sketch As before we build a simulator for non-colluding server $\mathcal{S}^{single}$ and extend it to the case of colluding servers $\mathcal{S}^{multi}$.

Let $Q_0, Q_1, \ldots, Q_\kappa$ be the queries submitted by the non-colluding adversary with Q_0 being the setup query, and for $q > 0$, Q_q being search query, document addition query or document removal query.

For Q_0, the simulator $\mathcal{S}^{single}$ generate an index γ' with random strings according to $\{|\mathbf{p}_{i,j}| \mid i = 1, \ldots, |\mathbf{W}_j|+|\mathbf{D}_j|\}$ in $\mathcal{L}^{setup}_{\mathtt{ms\text{-}SSE}}$, similar to the simulator for Theorem 2.

For document addition query Q_q, γ' is modified to match new entries if $\mathtt{A}_j \neq \emptyset$. First, search for the known index keys $l_{q,j}$ from L_j in search queries preceding Q_q. Let Q_i $i < q$ denote the identified query whose leakage includes $(c, l_{q,j}, |\mathbf{p}_{c,b}|)$ for some c and $|\mathbf{p}_{i,b}|$. Find element of $\mathtt{A}_j$ with matching counter, $cnt_{w_q} = c$ and extract its corresponding search token, y_{w_q}. Choose an entry in γ' with length $|\mathbf{p}_{i,b}|$ and move the entry to $\gamma'[F(y_{w_q}, id_{S_j})]$. The new index entries are added to γ' and $\mathbf{c}'_j$ in a similar steps for setup queries.

Document removal query requires $\mathcal{S}^{single}$ to delete the entry with the revealed index key l in R_j from γ' and $|\mathbf{c}^x_j|$ ciphertexts. If the document being deleted, D_x is added after the initial setup, then the index key is known and hence omit entry at $\gamma'[l]$. Otherwise, $\mathcal{S}^{single}$ chooses an entry with length $|\mathbf{c}^x_j|$ to omit from γ' and ensure $\gamma'[l]$ is empty.

For search query Q_q whose leakage shows SP(w_q)=v for some $v < q$. If there are update queries among Q_i for $v < i < q$, then $\mathbf{J}'_q \neq \mathbf{J}'_v$. Hence, in such cases, $\mathcal{S}^{single}$ use the other leakage elements to form $\mathbf{J}'_q$ correctly. Moreover, the index γ' must be updated according to preceding document addition and removal queries.

The simulator for colluding servers $\mathcal{S}^{multi}$ consists of t $\mathcal{S}^{single}$ where t is the number of servers controlled by the adversary. The shared information among $(\gamma'_j)^t_{j=1}$ which includes the same search token for every queried keyword is maintained because of the revealed index keys for document addition and removal, and for search queries.

The simulators described above are able to produce outputs fulfilling the leakages correctly. This implies that the adversary has insignificant probability in distinguishing the **Real** game from the **Ideal** game. □

Doc+ variants with forward privacy The `Doc+` protocol can be modified to achieve forward privacy introduced by [32], in that even if a keyword is searched before, the server will not be able to learn whether a newly added document also contains this keyword. We can use a unique key K^q or maintain a counter cnt_q to generate $K^q_{tok} \leftarrow F(K^q, 1||w_q)$ or $K^c_{tok} \leftarrow F(K, 1||w_q||cnt_q)$ for every occurrence of w_q when adding documents. Nevertheless, both approaches are bandwidth intensive as during search the list of K^q_{tok} or K^c_{tok} must be submitted to all servers, in which the list grows linearly for every document update.

7 Performance

We summarise the performance of $\texttt{ms-SSE}_M$ in Table 1 with comparisons to some recent schemes. In the table, *Leak* measures leakage based on indexes, documents and keywords. *Index size* and *Search time* is measured per server. *Comm.* measures bandwidth for index, query and update submission while *Inter.* measures the number of rounds required to retrieve the documents/blocks.

Our schemes leak total number of blocks Nl and the number of blocks per keyword $|\mathbf{p}_w|$, as compared to [28], which only reveals the total number of blocks and indexes b. We note though, Naveed et al. reveals document size $|D_f|$ after queries and requires more rounds of interaction.

Index size in $\texttt{ms-SSE}_G$ is equivalent to the underlying `SSE` scheme since the index is created by the `SSE` scheme it deploys. In the case of $\texttt{ms-SSE}_M$, the total index size is $O(Nl)$, which for a single server is $O(Nl/s)$. For $\texttt{ms-SSE}^{dyn}_M$, the index size is $O((N+n)l)/s)$, which is higher due to the addition of document identifier entries in the index.

Search and update cost for $\texttt{ms-SSE}_G$ is as before equivalent to the underlying `SSE` scheme, but for communication, all our schemes are with higher overhead $O(s)$ as it has to send the query token to s servers instead of one server. We note that, however, this is optimal when there is more than one server, unless we require the user to store information

Table 1 Comparisons with a few recent SSE schemes

Schemes	Leak	Index size	Search time	Comm.	Inter.
$\texttt{ms-SSE}_G$	=`SSE` [c]	=`SSE`	=`SSE`	$O(s)$	2r
	(`SSE`)$/s$ [nc]				
$\texttt{ms-SSE}_M$	$\lvert\mathbf{W}\rvert$, Nl, $\lvert\mathbf{p}_w\rvert$ [c]	$O(Nl/s)$	$avg(z/s)$	$O(s)$	2r
	$\lvert\mathbf{W}\rvert$, Nl/s, $\lvert\mathbf{p}_w\rvert/s$ [nc]				
	$\lvert\mathbf{W}\rvert+n$, $(N+n)l$, $\lvert\mathbf{p}_w\rvert$ [c]	$O((N+n)l/s)$	$avg(z/s)$	$O(s)$	2r
$\texttt{ms-SSE}^{dyn}_M$	$\lvert\mathbf{W}\rvert+n$, $((N+n)l)/s$,				
	$\lvert\mathbf{p}_w\rvert/s$ [nc]				
[7]	$\lvert\mathbf{W}\rvert$, n, M, $\lvert D_f\rvert$	$O(Mn)$	$O(z)$	$O(1)$	2r
[5]	N, n, $\lvert D_f\rvert$	$O(N)$	$O(z/x)$	$O(1)$	2r
[28]	b, $\lvert D_f\rvert^{@}$	$O(b)$	$O(z)$	$O(1)$	4r

n = number of documents, s = number of servers, b = number of blocks of documents and indexes, $l = \max(h)$, $|\mathbf{p}_w|$ = number of block identifiers per keyword for all servers, $z = |\texttt{DB}(w)|$, $N = \Sigma_{w\in\mathbf{W}}|\texttt{DB}(w)|$, x = number of processors, $M = \max(|\texttt{DB}(w)|)$, avg denotes on average, [c] = colluding servers, [nc] = non-colluding servers, r = rounds of interaction between user and a server. @: For [28], $|D_f|$ is leaked only after query for servers to pull the blocks from, but this then increases storage and computational cost on the user. Search time for each server in $\texttt{ms-SSE}_M$ and $\texttt{ms-SSE}_M^{dyn}$ is averagely z/s. All in all, our proposals provide better hiding properties in terms of document size and number of documents, with trade-offs on storage and communication.

8 Implementation of $\texttt{ms-SSE}_M$

We implemented the $\texttt{ms-SSE}_M$ scheme, simulating multiple servers on a single local machine. The application was written as a single thread application. The workstation is running on an Intel Core i7-4510U (2 GHz) processor with 8 GB of memory and 256 GB of SSD.

The implementation was run using a test file with file size: 1,252,862,080 bytes (i.e., about 1.25 GB). Each chunk of block was the size of 50,000 bytes (50 kB). A total of number of 25,058 blocks was generated.

We measured the time taken for each algorithm. To generate the master key, the time taken was less than 1 millisecond (ms). 5 servers were simulated and the time taken to generate all five server keys was 234 ms. Building the index blocks, which includes block segmentation and encryption, took a total of 121.44 seconds. Searching took 81 ms whereas retrieval of the file, which includes block decryption and merging, took 39.19 seconds.

9 Conclusions

We proposed extensions of SSE to SSE over multiple servers. Given the availability of many providers, our rationale is that a user may choose to outsource encrypted documents to a few of them instead of just one. We defined multi-server SSE and proposed a generic construction that can be used to readily extend existing SSE schemes to work on multiple servers with minimal modification. We further proposed a construction that provides better hiding properties. It divides documents into sets of encrypted blocks and randomly assigns these blocks to different servers. This is done in a way that protects against any server from obtaining the complete set of blocks of a document with high probability, even in the encrypted form assuming servers do not collude, while achieving sublinear search time for each server. On security, we explore leakages under a new setting of non-colluding and colluding servers.

Acknowledgements The authors thank the anonymous reviewers for their constructive comments that have helped to improve this paper.

References

1. Boneh, D., Crescenzo, G.D., Ostrovsky, R., Persiano, G.: Public key encryption with keyword search. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004, Springer Berlin / Heidelberg, LNCS, vol. 3027, pp. 506–522 (2004)
2. Bösch, C., Hartel, P., Jonker, W., Peter, A.: A survey of provably secure searchable encryption. ACM Comput. Surv. **47**(2), 18:1–18:51 (2014)
3. Bösch, C., Peter, A., Leenders, B., Lim, H.W., Tang, Q., Wang, H., Hartel, P.H., Jonker, W., Jøsang, A., García-Alfaro, J.: Distributed searchable symmetric encryption. In: Miri, A., Hengartner, U., Huang, N. (eds.) PST 2014, IEEE, pp. 330–337 (2014)

4. Cash, D., Jarecki, S., Jutla, C.S., Krawczyk, H., Rosu, M.C., Steiner, M.: Highly-scalable searchable symmetric encryption with support for boolean queries. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013 and IACR Eprint Archive (2013:169), Springer, LNCS, vol. 8042, pp. 353–373 (2013)
5. Cash, D., Jaeger, J., Jarecki, S., Jutla, C., Krawczyk, H., Rosu, M.C., Steiner, M.: Dynamic searchable encryption in very large databases: data structures and implementation. In: NDSS 2014 and IACR Eprint Archive (2014:853), Internet Society, vol. 2014 (2014)
6. Chang, Y., Mitzenmacher, M.: Privacy preserving keyword searches on remote encrypted data. In: Ioannidis, J., Keromytis, A.D., Yung, M. (eds.) ACNS 2005, Springer, LNCS, vol. 3531, pp. 442–455 (2005)
7. Chase, M., Kamara, S.: Structured encryption and controlled disclosure. In: Abe, M. (ed.) ASIACRYPT 2010, Springer, LNCS, vol. 6477, pp. 577–594 (2010)
8. Cheng, R., Yan, J., Guan, C., Zhang, F., Ren, K.: Verifiable searchable symmetric encryption from indistinguishability obfuscation. In: Bao, F., Miller, S., Zhou, J., Ahn, G. (eds.) ACM ASIA CCS 2015, 1, pp. 621–626 (2015)
9. Chor, B., Kushilevitz, E., Goldreich, O., Sudan, M.: Private information retrieval. J. ACM **45**(6), 965–981 (1998)
10. Curtmola, R., Garay, J.A., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. In: Juels, A., Wright, R.N., di Vimercati, S.D.C. (eds.) ACM CCS 2006, 1, pp. 79–88 (2006)
11. Faber, S., Jarecki, S., Krawczyk, H., Nguyen, Q., Rosu, M., Steiner, M.: Rich queries on encrypted data: beyond exact matches. IACR ePrint Archive 2015:927. http://eprint.iacr.org/2015/927 (2015)
12. Gentry, C.: A fully homomorphic encryption scheme. PhD thesis. Stanford University (2009)
13. Goh, E.J.: Secure indexes. IACR ePrint Archive, Report 2003/216. http://eprint.iacr.org/2003/216/ (2003)
14. Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious RAMs. J. ACM **43**(3), 431–473 (1996)
15. Hahn, F., Kerschbaum, F.: Searchable encryption with secure and efficient updates. In: Ahn, G., Yung, M., Li, N. (eds.) ACM SIGSAC 2014, 1, pp. 310–320 (2014)
16. Ishal, Y., Kushilevitz, E., Lu, S., Ostrovsky, R.: Private large-scale databases with distributed searchable symmetric encryption. IACR ePrint Archive (and CT-RSA 2016) 2015:1190 (2015)
17. Islam, M.S., Kuzu, M., Kantarcioglu, M.: Access pattern disclosure on searchable encryption: ramification, attack and mitigation. In: NDSS 2012, The Internet Society (2012)
18. Jarecki, S., Jutla, C.S., Krawczyk, H., Rosu, M.C., Steiner, M.: Outsourced symmetric private information retrieval. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013, 1, pp. 875–888 (2013)
19. Kamara, S., Papamanthou, C.: Parallel and dynamic searchable symmetric encryption. In: Sadeghi, A.R. (ed.) FC 2013, Springer, LNCS, vol. 7859, pp. 258–274 (2013)
20. Kamara, S., Papamanthou, C., Roeder, T.: Dynamic searchable symmetric encryption. In: Yu, T., Danezis, G., Gligor, V.D. (eds.) ACM CCS 2012, 1, pp. 965–976 (2012)
21. Katz, J., Lindell, Y.: Introduction to modern cryptography. Chapman & Hall/CRC (2007)
22. Kurosawa, K., Ohtaki, Y., Keromytis, A.D.: UC-Secure searchable symmetric encryption. In: FC 2012, Springer, LNCS, vol. 7397, pp. 285–298 (2012)
23. Kurosawa, K., Ohtaki, Y.: How to construct UC-secure searchable symmetric encryption scheme. IACR ePrint Archive 2015:251 (2015)
24. Kuzu, M., Islam, M.S., Kantarcioglu, M.: Distributed search over encrypted big data. In: Park, J., Squicciarini, A.C. (eds.) ACM CODASPY 2015, 1, pp. 271–278 (2015)
25. Moataz, T., Shikfa, A.: Boolean symmetric searchable encryption. In: Chen, K., Xie, Q., Qiu, W., Li, N., Tzeng, W.G. (eds.) ASIACCS 2013, ACM, pp. 265–276 (2013)
26. Mohamad, M.S., Poh, G.S.: Verifiable structured encryption. In: Kutylowski, M., Yung, M. (eds.) Inscrypt 2012, Springer, LNCS, vol. 7763, pp. 137–156 (2012)
27. Mohamad, M.S., Poh, G.S., Chin, J.J.: Securing outsourced storage. In: Cryptology 2016, Institute for Mathematical Research, UPM, pp. 111–119 (2016)
28. Naveed, M., Prabhakaran, M., Gunter, C.A.: Dynamic searchable encryption via blind storage. In: IEEE S & P 2014, IEEE Computer Society, pp. 639–654 (2014)
29. Orencik, C., Selcuk, A., Savas, E., Kantarcioglu, M.: Multi-keyword search over encrypted data with scoring and search pattern obfuscation. Int. J. Inf. Sec. (Online: 23-05-15) pp 1–19 (2015)
30. Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: IEEE S & P 2000, IEEE Computer Society, p. 44 (2000)
31. Stefanov, E., Shi, E.: ObliviStore: High performance oblivious cloud storage. In: IEEE S & P 2013, IEEE Computer Society, pp. 253–267 (2013)
32. Stefanov, E., Papamanthou, C., Shi, E.: Practical dynamic searchable encryption with small leakage. In: NDSS 2014, The Internet Society (2014). http://www.internetsociety.org/events/ndss-symposium-2014