Received: 09 June 2021 | Revised: 12 July 2021 | Accepted: 13 July 2021
DOI: 10.37917/ijeee.17.2.3 Vol. 17| Issue 2| December 2021

Open Access

Iraqi Journal for Electrical and Electronic Engineering
Original Article

Backward Private Searchable Symmetric Encryption with Improved Locality

Salim S. Bilbul, Ayad I. Abdulsada*,
Department of Computer science, Education College for Pure Sciences,
University of Basrah, Basrah, 61004, Iraq

Correspondence
* Ayad I. Abdulsada
Department of Computer science
Education College for Pure Sciences,
University of Basrah, Basrah, Iraq
Email: ayad.abdulsada@uobasrah.edu.iq

Abstract
Searchable symmetric encryption (SSE) enables clients to outsource their encrypted documents into a remote server and allows them to search the outsourced data efficiently without violating the privacy of the documents and search queries. Dynamic SSE schemes (DSSE) include performing update queries, where documents can be added or removed at the expense of leaking more information to the server. Two important privacy notions are addressed in DSSE schemes: forward and backward privacy. The first one prevents associating the newly added documents with previously issued search queries. While the second one ensures that the deleted documents cannot be linked with subsequent search queries. Backward has three formal types of leakage ordered from strong to weak security: Type-I, Type-II, and Type-III. In this paper, we propose a new DSSE scheme that achieves Type-II backward and forward privacy by generating fresh keys for each search query and preventing the server from learning the underlying operation (del or add) included in update query. Our scheme improves I/O performance and search cost. We implement our scheme and compare its efficiency against the most efficient backward privacy DSSE schemes in the literature of the same leakage: MITRA and MITRA*. Results show that our scheme outperforms the previous schemes in terms of efficiency in dynamic environments. In our experiments, the server takes 699ms to search and return (100,000) results.

KEYWORDS: Searchable encryption, I/O efficiency, Symmetric primitives.

I. INTRODUCTION

Searchable symmetric encryption (SSE) schemes are efficient solutions that allow a client to store his encrypted documents at a remote untrusted server while enabling him/her to search and retrieve documents that match the search queries without violating the privacy of documents and the underlying search keywords. The basic approach of SSE schemes is to extract an inverted index from the entire document collection that maps each keyword *w* to the set of document identifiers whose documents include *w*. The index is encrypted and outsourced along with the encrypted documents to the server. To search on the encrypted index, the client provides search tokens by encrypting certain keywords using a secret key. The server runs a search algorithm on the search tokens and the encrypted index to find the matched entries. However, search tokens are constructed in a deterministic way, thus the server can learn when multiple searches share a common keyword. This leakage is called the search pattern [3, 4]. Efficient SSE schemes [1, 2, 3, 5, 6] allow leaking also the access pattern [3], which is the set documents that include the search keyword. Several studies [4, 7- 9] have shown how to exploit the above-mentioned leakages by the adversarial servers to perform specific attacks to violate documents and search queries privates, making traditional SSE schemes unsuitable for practical applications. Such an attack inspires the importance of forward privacy, which ensures that newly inserted documents should never be linked with previous search queries. The early schemes of SSE are suitable to deal with static collections. Dynamic SSE schemes support update operations on the collection [10–12]. The functionality of dynamic SSE introduces new privacy challenges. For example, insert query of specific document d to the collection might be related with previous search queries for a given keyword w to reveal that w is included in d. Schemes that resist such leakage are called forward private and are first introduced in [10], and then addressed in many works like [1, 5, 13–15] to avoid file-injection attacks [9]. Another privacy challenge is raised when a search query for a given keyword w might be associated with deleted documents to infer that such documents were continuing w, in the past. Schemes that minimize this leakage are called backward private. Unfortunately, apart from being mentioned in [1], neither real constructions were presented to achieve this property nor a formal definition was provided. The first formal definition of backward privacy is due Bost et al. [2] where three different types of leakage are proposed: Type-I, Type-II, and Type-III.

Contributions. In this paper, we design and implement a single-keyword SSE dynamic scheme that achieves both forward and Type-II backward privacy with two rounds for search queries. The security of our scheme is proved formally under the model of a random oracle. Our scheme enjoys the following properties:

1- *Forward privacy*: where a fresh key is used for generating new search tokens. Thus, the server is no longer able to associate previous search tokens with subsequent update tokens.

2- *Type-II backward privacy*: where update operations (del or add) are never revealed to the server during the execution of update queries.

3- *Efficient index*: During search, the accessed entries of the encrypted index are marked and removed, this prevents increase in index size.

4- *Optimized I/O*: the results of the previous search are stored in the server in their plaintext format as such results are already leaked to the server and re-encrypting them again will not provide any security advantages. Therefore, plaintext results can be read continuously in an optimal locality.

5- *Efficient search and update*: The server needs to evaluate $O(o'_w)$ hash functions for a search a keyword w, where $o'_w$is the total number of updates on w since the last search. For update, the server only evaluates $O(1)$ storage operation. All operations are performed using light-symmetric cryptographic primitives.

6- *Comparison results*: we compare our scheme against MITRA and MITRA*, the most efficient backward private schemes to data, in terms of computation cost and I/O efficiency.

The remaining of the paper is organized into the following sections. Section **II** reviews the most relevant SSE schemes. Section **III** introduces the basic notations and cryptographic primitives that will be utilized in the current work. Section **IV** presents the proposed. Section **IIV** shows the results of our scheme. Section **IIIV** concludes the whole paper.

II. Related Works

Oblivious RAM-based SSE schemes. The functionality of searching the outsourced data was first achieved using the Oblivious RAM (ORAM) [16] tool that allows accessing the outsourced memory without revealing the access pattern. Several ORAM-based SSE schemes with different leakages have been proposed [1, 13]. However, Naveed et al. [17] have stated that such schemes do not hide the access pattern properly, since document identifiers of search results are revealed to the server for retrieving their corresponding documents.

Static SSE schemes. Static schemes allow performing search operation only over the encrypted data. The first linear-time search SSE scheme is due to the seminal work of Song et al. [18]. Curtmola et al. [3] defined the first formal definition of the security of SSE schemes. Particularly, they introduced the notion of leakage function and successfully proposed an inverted-index-based SSE scheme with sub-linear search time. Boolean SSE was introduced in [19]. Disjunctive SSE is developed by Kamara and Moataz [12].

Dynamic SSE schemes. Dynamic SSE (DSSE) schemes allow updating the encrypted data and were first introduced by Kamara et al. [20] and enhanced in [11]. To deal with large-scale datasets, Cash et al. [6] proposed an optimized DSSE scheme. However, no one of these schemes supports forward privacy.

Forward privacy. The first formal definition of forward privacy is introduced by Bost in [5] along with an optimal communication DSSE scheme, Sophos. Diana [2], a forward private DSSE scheme, is based on symmetric cartographic primitives GGM-PRF [21] to support parallel search with higher communication overhead. Improved forward private DSSE schemes are presented in [14, 15, 22]. The strategy of [22] to achieve forward privacy was to re-generate the keys of update operation after each search query. However, this method requires re-encrypting the search results by the client and re-sending them back to the server, which incurs more I/O overhead and one additional round of interaction. FASTIO [14] has improved the locality [6] of DSSE schemes.

Backward privacy. The notion of backward privacy was first introduced by Stefanov et al. in [1] without providing a formal definition. In their work, an ORAM-based SSE scheme was released to capture only forward privacy. Such a construction has the ability to deal with deleting entries, where such entries are skipped elegantly by the server. Unfortunately, this work focuses on improving the search performance rather than security concerns. Bost [2] introduced the first formal definition of backward privacy with three different types of information leakage ordered from strong to weak security requirements: Type-I, Type-II, and Type-III. Furthermore, the author introduced four backward private DSSE schemes with variable privacy/performance balance: Fides, Moneta, Diana$_{\mathrm{del}}$, and Janus. Fides is a Type-II scheme that employs two instances of the forward private DSSE scheme, Sophos [5]. The first instance is used to store addition operations, while the second one is used to store deletion operations. During the search operation, the two instances are inspected to retrieve all entries, the deleted ones are filtered at the queried side, and the server is asked to return the documents of the remaining identifiers. In this case, the server is not able to know the deleted documents. Monita is a Type-I DSSE scheme which is constructed using TWORAM [13], an ORAM-based forward private DSSE scheme that uses heavy communication operations making it suitable for only theoretical instances of Type-I schemes. Diana$_{\mathrm{del}}$ and Janus are Type-III DSSE schemes that leak more information to the server for better performance. Diana$_{\mathrm{del}}$ is constructed from constrained pseudorandom functions (CPRF) [23], which incurs high communication and computation costs. Janus employs puncturable encryption [24] that supports the property of incremental update, to release an optimum DSSE scheme with a single-round search and optimal communication cost. However, the search time of Janus is not optimal. Such that, after hundreds of deletion operations, Janus will not be practical, mainly due to the computation and storage costs. Furthermore, $\text{Diana}_{\text{del}}$ and Janus disallow the reinsertion of the previously deleted keyword-document pairs. Recently, Chamani et al. [25] proposed MITRA, the most practical Type-II backward private DSSE scheme in the literature. An improved version of MITRA, which is called MITRA*, can handle deletion operations efficiently. Orion and Horus [25] are two backward private schemes with quasi-optimal search time (i.e. $O(n_w \text{polylog}(N))$), which is far from being optimal search time $O(O(n_w)$, where n_w is the number of current documents that share the keyword w. Orion [25] is a Type-I scheme that requires $O(log\ N)$ round of interactions, where N is the number of document-keyword pairs. Horus is a Type-III scheme that achieves better search performance than Orion and requires few rounds of interaction. Table 1 summarizes the most relevant DSSE schemes. All of the listed schemes support forward and backward privacy.

III. Background

This section introduces the basic notations and cryptographic primitives that will be utilized in the current work.

A. Notations

The security parameter is denoted by $\lambda \in \mathbb{N}$, which is used as input in all algorithms of our scheme. Probabilistic polynomial-time is referred to as PPT. $\text{negl}(\lambda)$ stands for a negligible function in λ, where $f: \mathbb{N} \to \mathbb{R}$ is considered a negligible function only iff for all $c > 0, \exists n\_0 \in N$ s.t $\forall n \geq n_0, f(n) < n^{-c}$. In the context of the client-server setting, the notation $P(x; y)$ indicates that the protocol P is executed by the input x of the client and the input y of the server. The cardinality of a set X is denoted by $|X|$. $x \overset{\$}{\leftarrow} X$ notation stands for sampling x uniformly from X. Assigning the value y to variable x is denoted by $x \leftarrow y$. The notation $||$ refers to the concatenation operation. The notion $\{0,1\}^{\ell}$ denotes the set of all strings of length ℓ, and $\{0,1\}^{*}$ denotes all strings of arbitrary lengths.

Consider a collection of D documents that includes textual keywords derived from a defined alphabet Σ, with each document d_i is identified by its identifier $id_i \in \{0,1\}^{\ell}$. The database DB includes a set of pairs of document identifiers and keywords (id, w) such that the keyword w appears in the document with identifier id. The set of all unique keywords that appear in DB is denoted by W of size $K\ (i.e, K = |W|)$, N stands for the number of pairs in DB (i.e., $N = |DB|$). The set of documents that include w is denoted by $DB(w)$.

TABLE 1

Comparison of existing forward and backward private DSSE schemes with our scheme. N is the total number of (keyword, identifier) mappings. For keyword w, a_w is the total number of addition operations on w, d_w is the number of delete operations performed on w, o_w is the total number of updates on w (i.e $o_w = a_w + d_w$), o'_w is the total number of updates since the last search, n_w is the number of documents currently sharing w, RO stands for the number of rounds for search. $\hat{O}$ hides the loglogN factors.

	Computation		Communication			Backward
Schemes	Search	Update	Search	Update	Search RO	Private
Moneta [2]	$\hat{O}(a_w \log N + \log^3 N)$	$\hat{O}(\log^2 N)$	$\hat{O}(a_w \log N + \log^3 N)$	$\hat{O}(\log^3 N)$	3	I
Fides[2]	$O(a_w)$	$O(1)$	$O(a_w)$	$O(1)$	2	II
Diana_{del}[2]	$O(a_w)$	$O(\log a_w)$	$O(n_w + d_w \log(a_w))$	$O(1)$	2	III
Janus[2]	$O(n_w d_w)$	$O(1)$	$O(n_w)$	$O(1)$	1	III
MITRA[25]	$O(a_w)$	$O(1)$	$O(n_w)$	$O(1)$	2	II
Orion[25]	$O(n_w \log^2 N)$	$O(\log^2 N)$	$O(n_w \log^2 N)$	$O(\log^2 N)$	$O(\log N)$	I
Horus[25]	$O(n_w \log d_w \log N)$	$O(\log^2 N)$	$O(n_w \log d_w \log N)$	$O(log^2 N)$	$O(\log d_w)$	III
Our scheme	$O(o'_w)$	$O(1)$	$O(o'_w)$	$O(1)$	2	II

B. Dynamic Searchable symmetric encryption (DSSE)

DSSE scheme π =(Setup, Search ,Update) includes one algorithm $Setup$ and two protocols $Search$, $Update$ that are executed between a client and a server. The client holds the databased DB and outsources the encrypted database EDB to the server.

- $Setup(\lambda, DB; \perp)$ is an algorithm that receives the security parameter (λ) and the data base DB as input and returns $(sk, \sigma;$ EDB) to the client where sk is a secret key, σ is the local state and EDB is an empty database that outsourced in the server.
- $Search(sk, q, \sigma;\ EDB)$ is the protocol of searching the database. In the current work, we deal with only search queries of single keywords $(q = w \in \Sigma^{*})$. The output of this protocol to the client is $DB(w)$ (or $\perp$ when $w \notin W$). It also may change σ and EDB.
- $Update(sk, op, in, \sigma;\ EDB)$ is the protocol for updating the database. op stands for update operation which could be either del or add. Input in is a pair (id, w). This protocol may also change the values of σ and EDB.

Given the security parameter λ and the database DB, the client starts to run Setup to get the secret key sk then he/she adds the N entries of DB into EDB by calling Update protocol N times. Search protocol returns for the client only the document identifiers of $DB(w)$. Actual documents are returned to the client with an additional round. We consider a semi-honest server, where it follows precisely the steps of the protocol but tries to get additional information from the messages (transcripts) it receives during the execution of the protocol.

Forward privacy. Forward privacy ensures cutting the link between earlier search queries and the current update operation. This feature is useful when we want to hide whether the current addition operation is related to a new keyword or an existing one.

Backward privacy. The objective of backward privacy is to minimize the information leaked to the server when a search query is issued for a keyword w which some of its entries have been deleted previously. Informally DSSE scheme is considered backward-private scheme when the search query on keyword w does not reveal the identifier id that its corresponding pair (w, id) is added into the database and then removed. Observe that the search query reveals id only when it is issued after the addition of (w, id) and before deleting this pair. Bost et al. [2] introduced formally three different types of backward privacy with variable leakage patterns: Type-I, Type-II, and Type-III. Type-I reveals the lowest information, while Type-III reveals the most. Informally, these types are defined as follows:

- Type-I(*Backward privacy revealing insertion pattern (BPIP)*): When BPIP schemes search for keyword w, nothing is revealed to the server beyond: the documents currently matching w, time of inserting documents, and the total number of deletion and addition operations on w.
- Type-II(*Backward privacy revealing update pattern (BPUP)*): This type reveals all information of type-I plus the times at which w is updated.
- Type-III(*Weak backward privacy (WBP)*) In addition to the leakage of Type-II, this type reveals also the deletion history of w (i.e. which deletion canceled which addition)

C. *Scalability of searchable encryption schemes*

Real implementations [3, 6] have shown that the scalability of *SSE* schemes to large databases is mainly affected not by their usage of cryptographic tools but rather by three irreconcilable memory parameters: storage locality, storage cost, and read efficiency. The notion of locality represents the number of non-contiguous memory reads required by the server to retrieve the result items matching search queries, while read-efficiency represents the additional number of memory reads made by the server to retrieve the result items beyond the actual need. The trade-off between these parameters was first observed by Cash et al. [6], where to improve their locality; *SSE* schemes have to read additional entries per query. This hurt practical performance. Most prior *SSE* schemes scatter index entries at pseudorandom locations which causes an increased number of localities and hence degrading search performance. Cash and Tessaro [28] addressed the lower bound of server memory access locality of *SSE* schemes: they demonstrated that, for the sake of security, it is not possible to achieve, at the same time, optimal server storage, read efficiency, and optimal locality. Their lower bound states that if a scheme holds optimal locality and storage space, then when the adversary server knows the locations of results for previous search queries, then it can infer from the non-accessed locations some statistical information about the underlying data collection. Asharov et al. [29] give a tighter lower bound of storage locality. Furthermore, Demertzis et al.[30] have shown how locality notion can be tuned. However, all of the aforementioned studies hold for static schemes, and the locality of dynamic locality is not studied formally. Thus, an optimized I/O performance is required for dynamic SSE schemes while preserving their security requirements.

IV. Proposed Scheme

In this section, we introduce our proposed scheme that supports forward and backward privacy. Our scheme follows the definition of Type-II for backward privacy in a manner that it reveals nothing to the server during the execution of update queries other than the time at which such updates happened during searches. The proposed scheme follows the two-party model, where the first party is the data owner (client) and the second party is the server that provides large storage and efficient computation power. Our scheme relays, as almost all current DSSE schemes [14, 19, 22, 25], on a key-value index to capture the relation of each keyword with its corresponding identifiers. Such an index is encrypted and outsourced to the server to enable it for answering the requests of clients efficiently. The client formulates his requests in the form of encrypted tokens, which are sent to the server to perform its job. Index stores, for a given keyword w, the encryption of (id, op) in the corresponding key that is derived from w, where op is the update operation, and id is the identifier of a document involved in this operation. Keys represent the addresses for storing the corresponding values at index and are obtained utilizing pseudorandom and secure hash functions. Keys should be derived in a way that ensures create the ability for the client to generate the same address related to the relevant keyword w during the search. The encrypted index is maintained by using synchronized data structures stored at client and server. Note that, our work focuses on the encrypted index and does not consider the encrypted documents. This means that encryption of actual documents is not included in the steps of our work. Such a simplification is common in the literature of SSE schemes, for example [2, 6, 19, 22]. The reason behind this treatment is that the actual documents are commonly protected by CPA-encryption schemes like AES and thus no information is leaked from the ciphertext except document size.

Server data structures. The server uses two data structures: S_e of size $O(N)$ and S_r of size $O(K)$. S_e is used to associate each keyword to the set of document identifiers which includes it. If document d_j contains the keyword w, then the encryption of its identifier id_j along with its update operation op is stored in S_e at address determined by K_w (derived from w) and the index j. In this setting, given K_w and the number of documents sharing w, the server identifies all relevant addresses, and returns the corresponding encrypted values, which will be refined by the client to get the document identifiers currently matching w. S_e is used to store the set of all plaintext document identifiers that match w. The address of each entry in S_e is random-looking obtained by using a secure hash function. Thus, entries of a given keyword are scattered at random locations in S_e. As we process the keywords sequentially to construct S_e, this prevents leaking any information about the entries of each keyword. However, this is true only when keywords are outsourced at the beginning. Later insertion of a specific keyword conceals the number of documents sharing it, and hence Se does not prevent such leakage.

Client Data structures. The client has to store two data structures: $Cnt[w]$ that capture the number of documents sharing the keyword w (as in Cash et al. [6]), and $SrchCnt[w]$ that counts the number of search operations on a keyword w. This counter is utilized to achieve forward privacy by generating new keys after each search. Both Cnt and $SrchCnt$ data structures are of size $O(K)$ and are initialized with zeros. The main advantage of our scheme over FASTIO of [14], which supports only forward privacy, is that, instead of outsourcing to the server only masked values, we outsource encrypted values that are decrypted locally at the client-side. To improve efficiency, we store the search query result of previous searches in their plaintext, such that when a new search query is issued, the server can leverage the cashed result and only needs to investigate the new encrypted update entries since the previous search query. By this procedure, our scheme outperforms the most efficient DSSE scheme of the same leakage, Mitra* [25]. Figure 1 illustrates our proposed scheme.

Assumptions. Let λ be security parameter, $\mathrm{GenPRF}(1^\lambda), \mathrm{GenPRP}(1^\lambda)$, be key generation functions, $\mathrm{F}: \{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^\lambda$ be a pseudorandom function PRF, $\mathrm{h}: \{0,1\}^* \rightarrow \{0,1\}^\lambda$ be a Hash function modeled as random oracle, and $G: \{0,1\}^\lambda \times \{0,1\}^\lambda \rightarrow \{0,1\}^\lambda$ be a pseudorandom permutation function PPF.

```
    **Setup** (λ; ⊥)
    Client:
1-  $K_t, K_f \overset{\$}{\leftarrow} GenPRF(1^\lambda)$
2-  $K_g \overset{\$}{\leftarrow} GenPRP(1^\lambda)$
3-  $SrchCnt, Cnt \leftarrow$ empty map
4-  $\sigma \leftarrow \{SrchCnt, Cnt\}$
    Server:
5-  $S_e, S_r \leftarrow$ empty map
    **Update** $(K_g, K_f, \sigma, id, w, op;\ S_e)$
    Client:
6-  If $(\sigma[w] = \perp)$ then
7-    $SrchCnt[w] \leftarrow 0$,
8-    $Cnt[w] \leftarrow 0$
9-  End if
10- $Cnt[w] \leftarrow Cnt[w] + 1$
11- $K_w \leftarrow F_{K_f}(w||SrchCnt[w])$
12- $Addr \leftarrow h(K_w||Cnt[w]||0)$
13- $mask \leftarrow h(K_w||Cnt[w]||1)$
14- $sk \leftarrow F_{K_g}(w||SrchCnt[w])$
15- If $op =$ "add" then $d \leftarrow 1$
16- else $d \leftarrow 0$
17- $Val \leftarrow G_{sk}(id||d) \oplus mask$
18- Send $(Addr, Val)$ to Server
    Server:
19- $S_e[Addr] \leftarrow Val$
    **Search** $(K_g, K_f, K_t\ \sigma, w;\ S_e, S_r)$
    Round1 Client:
20- if $(\sigma\ [w] = \perp)$ then
21-   Return $\emptyset$
22- End if
23- $t_w \leftarrow F_{K_t}(w)$
24- $Cn =\ cnt[w]$
25- if $(Cn = 0)$ then
26-   $K_w \leftarrow \perp$
27- else
28-   $K_w \leftarrow F_{K_f}(w||SrchCnt[w])$
29- End if
30- send $(K_w, t_w, Cn)$ to the server
    Server:
31- $ID_1, ID_2 \leftarrow \{\}$
    Retrieve all last search result
32- $ID_1 \leftarrow S_r[t_w]$
33- if $(K_w = \perp)$then
34-   Return $ID_1$
35- End if
36- for $i = 1$ to $|Cn|$ do
37-   $Addr \leftarrow h(K_w||Cn||0)$
38-   $mask \leftarrow h(K_w||Cnt[w]||1)]$
39-   $ID_2 \leftarrow ID_2 \cup [S_e[Addr] \oplus mask]$
40-   Delete $S_e[Addr]$
41- End for
42- send $ID_1, ID_2$ to Client
43- Round2 Client:
44- if $(|\mathrm{ID}_2| \neq 0)$ then
45-   $Cnt[w] \leftarrow 0, SrchCnt[w] + 1$
46- End if
47- $sk \leftarrow F_{K_g}(w||\ SrchCnt[w])$
48- for $i = 1$ to $|\mathrm{ID}_2|$ do
49-   $(id||d) \leftarrow G_{sk}^{-1}(ID_2[i])$
50-   if $(\mathrm{d} = 1)$ then
51-     $ID_1 \leftarrow ID_1 \cup \{id\}$
52-   else
53-     $ID_1 = ID_1 - \{id\}$
54-   End if
55- End for
56- Send $\mathrm{ID}_1$ to server
    Server:
57- $S_r[t_w] \leftarrow ID_1$
```

A. *Details of the proposed scheme*

Setup. The setup algorithm generates randomly three secret keys K_f, K_g, and K_t all of size bounded by the security parameter λ. K_f is considered as long term key, in $PRF\ F$, that prevents the server from generating valid search tokens, K_g is used in $PRP\ G$ to protect (op, id) pairs, and K_t is used to construct a tag ,t_w, for each keyword w. The client initiates one empty state $Clntstate$ which is stored locally, while the server initiates two empty maps (S_e, Sr). S_e is used to store the encrypted entries, while S_r is used to store the cashed plaintext search results.

Update. In update protocol (lines 6-19), the client provides a keyword w, a document identifier id, and an update operation op, which is either add or del. For instance, the update query (add ,"book", 10) tells the server to add a new entry for keyword "book" in document 10. The local state σ stores for the keyword w, two counters. The first counter, $[w]$, accumulates the number of update operations that are related to w, while the second counter, $SrchCnt[w]$, denotes the total number of search operations that take place to w.

First, the client accesses the state σ of w to check its initialization. If it is not initialized yet, he/she sets the two counters related to w to 0. In both cases, counter $Cnt[w]$ is incremented by 1. Next, the client executes the PRF F with key K_f to calculates $\mathrm{F}_{\mathrm{K}_f}(\mathrm{w}, \mathrm{SrchCnt}[\mathrm{w}])$. The output, k_w, is considered as a secret key for the hash function, which will be executed twice. The output of the first hash is used as the key Addr that determines the location of storing the encrypted pair $(id||op)$ at the server, whereas the second output is XORed with the encrypted entry to get the masked value Val which will be uploaded to the server. The encryption key of the pair is derived from applying the function F on the inputs w and $SrchCnt[w]$ with a secret key K_g.

Search. Search protocol (lines 20-57) includes two rounds. In the first round, the client starts to check the initialization of ClntStat for w. If it is not initialized, then this means that the corresponding keyword w was not inserted before. So search operation will terminate. Otherwise, the client generates from w a tag value, t_w, to give the server the ability to return the set of document identifiers (if exists) that match w from the last search query. Next, the client checks the value of Cnt[w] to construct valid values for the search token $(\mathrm{t}_w, \mathrm{k}_w, v, \mathrm{Cnt}[\mathrm{w}])$. In this context, if the counter Cnt[w] value is set to zero, which means there are no update operations happen since the last search query, then the search results will remain the same as the previous search query. For this reason, the client unravels the new value of SrchCnt[w] and reveals only the tag tw, which is enough, in this case, to retrieve the correct results. If Cnt[w] is not equal to zero, the client will set v to the current SrchCnt[w]. Search on the server-side is comprised of two main steps. In the first step, the server uses t_w to retrieve the plaintext results from the recent search query (line 32). Then the server may move to the second step (line 33-42). If there is at least one update operation, since the last search query, the server will retrieve the new encrypted entries using the values of k_w, the counter Cnt[w] and the state v. Then the server removes the update entries from S_e to keep the index up-to-date and obviates the need for an index cleaning procedure. When all updates have been processed, the server returns the plaintext results and the encrypted update set to the client. In the second round, the client checks the received encrypted set, ID_2. If it is not empty, the client resets Cnt[w] and increments the counter SrchCnt[w] to make the future update operations are unlikable to the past issued search queries, which is a necessary condition for forward privacy. Then, the client proceeds over the received encrypted entries. In each iteration, he/she decrypts the entry, determines to keep its corresponding identifier depending on whether such identifier is removed later or not. Finally, the refined set is sent to the server, which stores them directly in S_r map.

B. Efficiency investigation

Search and update efficiency. We discuss the efficiency of our proposed scheme in terms of communication and computation complexity costs. Observe that update queries are only $O(1)$ for both parties as they require a fixed number of operations. The communication cost is also of $O(1)$ since only a single pair of values is sent from client to server. For search queries, our scheme requires $O(1)$ operations from the client in the first round, and $O(o'_w)$ of decryption operations, where o'_w represents the total number of update operations (add and del) that took place after the last search operation and also $O(o'_w)$ of look-ups and XOR operations from the server. Recall that the counter Cnt[w] is reset after each search operation, thus it catches only the number of update operation on w that happens after the last search, which is denoted by o'_w. Therefore, the computation and computation overheads of the search operation are $O(o'_w)$. When N update operations are executed, then they entail $O(N)$ storage at the server, since each update requires one storage in $\boldsymbol{S_e}$ map. The client should provide 2. $O(|W|)$ for Cnt and SrchCnt maps, and 3 λ-bit keys.

Deletion cost. Unlike the previous DSSE schemes [1, 2, 5, 1, 22, 25], which grows the encrypted index after each update operation, our work avoids this, where all accessed entries are removed from the encrypted index. Additionally deleted documents are filtered locally by the client.

Roundtrips. Our scheme requires two roundtrips to return the document identifiers matching w. An additional round is needed when we want to retrieve the actual documents. Note that our scheme relies on only light symmetric cryptographic primitives, making it attractive for practical applications.

I/O efficiency. Observe that locality of results and security are two conflicting notions. This is because, security requires update operations for a keyword w, to generate random locations in the encrypted index that are unrelated to already known locations relevant to w. To resolve such a conflict, we cache the results of previous search operations, at the server, in their plaintext. This reduces locality since result items will be organized together so that they can be returned continuously without violating privacy as such results do not provide additional leakage to the server. More precisely, when o'_w new update operations are performed after the last search query, a total of o'_w noncontiguous reads are needed. Thus, the overall locality is reduced from o_w to $o'_w + 1$.

Storage cost and read efficiency. Our scheme enjoys a near-optimal storage cost and read efficiency. Particularly, the index of our scheme is composed of two maps S_e and S_r In S_e, each entry is λ bits where λ is the security parameter. In S_r, each entry is a set of plaintext $\ell - bit$ document identifiers. Consider an index of N_t entries, the size of such index ranges between $N_t . \ell$ to $N_t . \lambda$ bits. To serve a plaintext search query, the server needs to read at least $|\mathrm{DB}(w)|. \ell$ bits for the relevant document identifiers. In our scheme, some identifiers are read from S_e and some are read from S_r. Entries in S_e adds additional λ - ℓ bits per identifier, while the entries in S_r add no cost. In practice, the document identifiers are represented by large numbers ($\lambda \approx \ell$) to ensure distinct identifiers. Hence, the storage cost and read efficiency in our scheme is near-optimal.

C. Security analysis

Our scheme is designed to support forward privacy and Type-II of backward privacy. Forward privacy is achieved since the two elements $(Addr, Val)$ are generated using a pseudorandom function F that receives a new input for each update, making it difficult for the server to distinguish them from random. Furthermore, even the update operation that is evaluated at the server, still hidden which leads to an empty update leakage. For backward privacy, the client sends to the server a search token that enables it to generate random locations which it observes during the previous updates. This allows for the server to know, for each w, the timestamp for each $update$ operation. Except for this, the server does not learn anything. Particularly deletions that cancel specific additions are not revealed to the server. This immediately leads to backward privacy of Type-II. For the search queries, only the access pattern and query pattern are leaked, which is standard leakage in the literature. Notice that, even though a fresh key is generated after each search, our scheme still leaks the search pattern since the searched keywords are not re-encrypted again. The fresh key is used to achieve forward privacy, but not protecting the search pattern. However, the actual keywords under search queries are protected; making the server unable to know them.

V. Experimental Results

We released a prototype for our scheme using Java programing language. Particularly, SHA-256 (of 160 bits long) is employed as the cryptographic hash function h. Using stronger hash functions like SHA-3 does not affect considerably the efficiency of our scheme, as such hashing is not the main cornerstone for our design. Index data structures are implemented as Java $HashMaps$, since it does not retain the order of inserted pairs. This is an interesting property for the security of S_e, so that the insertion operations leak no information about the order of inserted pairs. When a set of keywords are stored initially in S_e, all of their relevant entries are outsourced altogether. Thus, the server is not able to learn how many documents each keyword appears in. However, such information is leaked gradually through successive search queries. When keywords are outsourced one by one, the server notices the number of their corresponding entries in S_e. Since addresses in S_e are generated by hash functions, collisions may occur but with a very small probability. Consider the number N of stored pairs is bounded by 2^{80}, and output of the hash function is 256-bit. Then there are at most 2^{80} entries conflicting on 2^{256} address space. According to the birthday attack, stored entries collide with probability $\frac{N^2}{2*2^{256}} = \frac{2^{160}}{2^{257}} = 2^{-97}$, which is negligible. When a new entry causes a collision, this collision could be handled by the server, where it informs the client to increment the $SrchCnt[w]$ and regenerate its corresponding address. We release the $PRF\ F$ [31] with AES-128/256 for and $PPF\ G$ using AES-128/128. Our prototype is implemented on a desktop computer with a single Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz (with 8 logical cores), 8GB of RAM, and hard disk 512 GB SSD running Windows 10 64-bit operating system, x64-based processor. Our experiments were performed on a synthetic database DB of size $|DB| = 3 * 10^5$ entries, where entries are randomly generated. During the experiments, we constructed variable result sizes between 10– 10^5 documents. From the update operations, delete operations are performed with probability p of 0.1 to simulate the effect of deletions on performance. Throughout the experiments, results are reported as the average of 10 running times. We compare the performance of the proposed scheme with the most efficient backward private schemes in literature: Mitra and Mitra*. All of the compared schemes employ the same cryptographic symmetric primitives, which gives a fair comparison. To simulate the dynamic setting, we construct a sequence Seq of 100,000 interleaved search and update queries on a certain keyword. The sequence is constructed according to a search parameter δ that represents the probability of $search$ queries in the sequence, i.e. $(1 - \delta)$ is the probability of update queries.

A. Search efficiency

To locate the matched documents for a given search query the server is required to process a set of encrypted index entries. Figure 1 shows the search performance for two schemes. In this experiment, we record the total time at the server side to search for keywords having a variable number of matching documents. To get the time for processing a single entry we divide the total time by the number of matching documents. Notice that, for both schemes, the search time for processing a single entry is decreased when the number of matched documents increases. MITRA is slightly better than our scheme in terms of search time since it uses fewer processing steps than our scheme but at the expense of longer search tokens (see Figure 5) and more time for search token generation at the client-side (see Figure 2). However, this experiment does not consider the real scenario of search operations, where it considers searching for a given keyword only for one time. Such a setting ignores the locality improvement of our scheme, which will be explained later in this section.

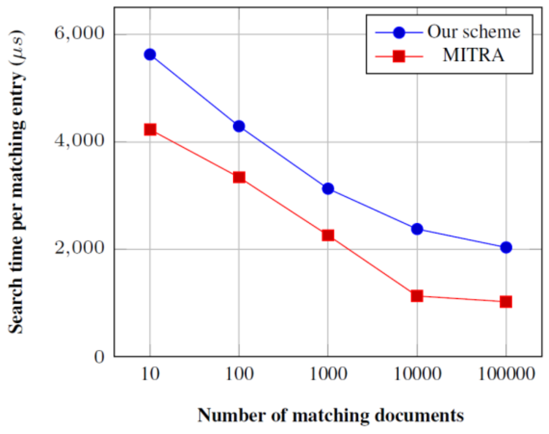

Fig. 1: Average per entry search time

B. Search tokens

Figure 2 compares our scheme with other schemes in terms of search token generation time at the client-side. Notice that our scheme outperforms the competent schemes. MITRA and MITRA* require more time to generate the search tokens when the matched documents grow.

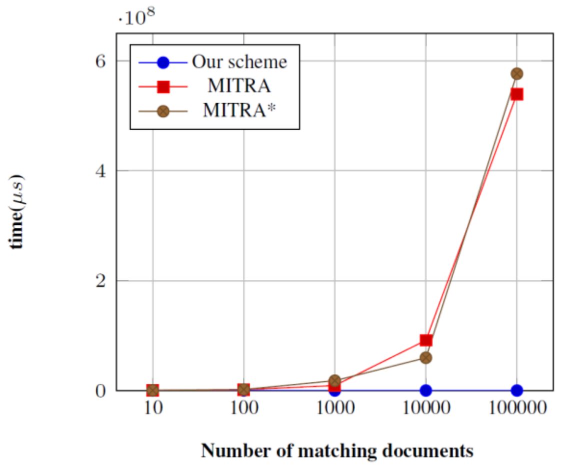

Fig. 2: Times for building trapdoor

C. *Effect of dynamic operations*

In this experiment, we show the effect of the dynamic environment on the performance of search queries. During this experiment, we accumulate the search time on the server-side for search queries in *Seq* sequence. Figure 3 illustrates the search time for different δ values. For MITRA and MITRA*, the search time grows linearly according to the number of executed update queries. Notice that MITRA* requires more time than MITRA since it re-encrypts the search results. In both schemes, search operation requires to touch index entries that are scattered among random locations, which decreases search locality. On the other hand, our scheme incurs much better search performance since it touches only the inserted entries after the last search. Thus we read the search results in lower locality than in the other competitor schemes.

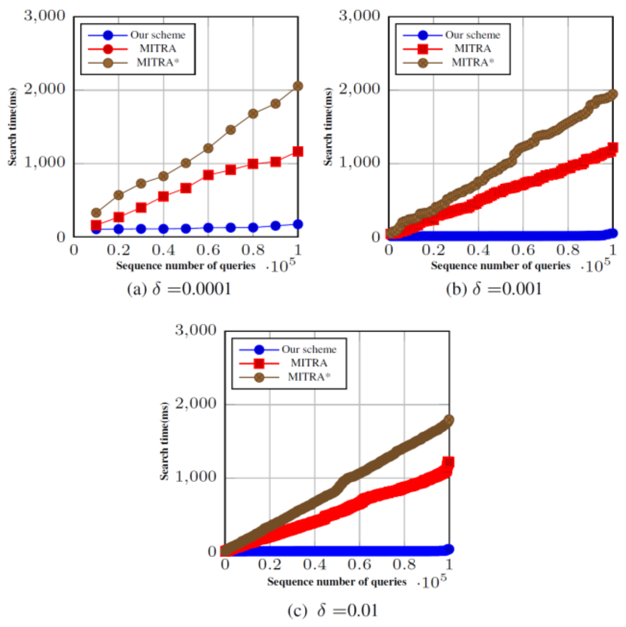

Fig. 3: Performance of dynamic search

D. *Communication cost*

In this experiment, we compare the communication cost of our proposed scheme against MITRA and MITRA*. Figure 4 demonstrates the communication cost as a function of the number of matched documents in sequence Seq. It is easy to see that our scheme incurs lower communication costs than the other schemes since it returns some results in plaintext forms. Recall that MITRA retrieves all the matched documents in their encrypted form and filters the deleted documents at the client-side. Thus it requires more communication cost. MITRA* is worse than MITRA since it re-encrypts the filtered results and sends them back to the server with an extra communication cost.

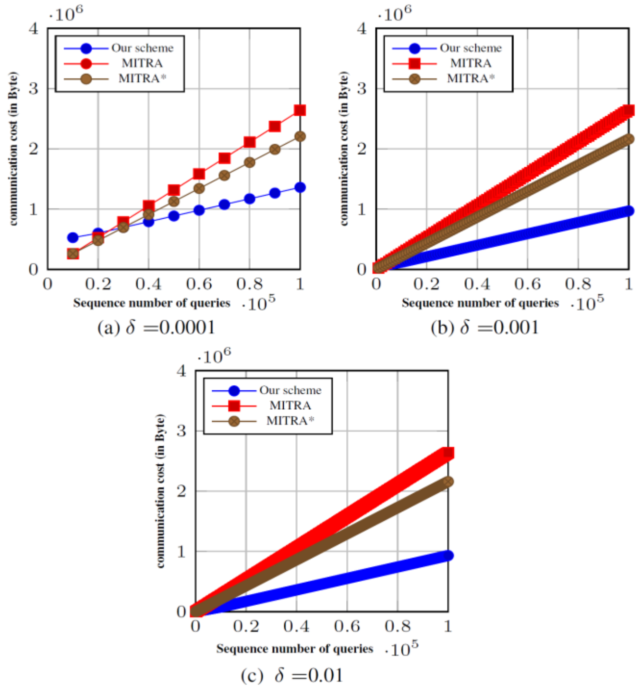

Fig. 4: Communication cost of matched results**.**

VI. Conclusion

Forward and backward privacy are two important properties to thwart serious attacks on SSE. In this paper, we introduce an efficient dynamic SSE scheme that achieves forward and backward privacy from only symmetric cryptographic primitives. Our scheme achieves optimal communication and computational complexities. Our treatment to cash the results of the previous search enhanced the I/O efficiency. Excremental results show that the proposed scheme is both efficient and practical.

The limitations of the current work can be illustrated as follows: first, we use one server to store the outsourced data and answer user's queries. Second, our scheme maintains a state of two counters for each keyword; our ongoing work is to deploy our work on a distributed setting, where multiple servers are used to store the encrypted data and jointly answer the provided queries and we would like to minimize the client storage cost into only a constant permanent cost.

Conflict of Interest

The authors have no conflict of relevant interest to this article.

References

[1] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. Practical dynamic searchable encryption with small leakage. In NDSS, volume 71, pages 72–75, 2014.

[2] Raphël Bost, Brice Minaud, and Olga Ohrimenko. *Forward and backward private searchable encryption from constrained cryptographic primitives*. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1465–1482, 2017.

[3] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. 2006. *Searchable symmetric encryption: improved definitions and efficient constructions*. In ACM CCS 2016. 79–88.

[4] C. Liu, L. Zhu, M. Wang, and Y.-a. Tan, "*Search pattern leakage in searchable encryption: Attacks and new construction,*" Information Sciences, vol. 265, 2014.

[5] Raphael Bost. $\Sigma o\phi o\varsigma$: Forward secure searchable encryption. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1143–1154, 2016.

[6] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S Jutla, Hugo Krawczyk, Marcel Catalin Rosu, and Michael Steiner. Dynamic searchable encryption in very-large databases: data structures and implementation. In NDSS, volume 14, pages 23–26. Citeseer, 2014.

[7] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. *Leakage-abuse attacks against searchable encryption*. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 668–679, 2015.

[8] M. Islam, M. Kuzu, and M. Kantarcioglu, "*Access pattern disclosure on searchable encryption: Ramification, attack and mitigation*," in Network and Distributed System Security (NDSS), 2012.

[9] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 707–720, 2016.

[10] Y.-C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in International conference on applied cryptography and network security. Springer, 2005, pp. 442–455.

[11] S. Kamara and C. Papamanthou, "Parallel and dynamic searchable symmetric encryption," in International Conference on Financial Cryptography and Data Security. Springer, 2013, pp. 258–274.

[12] S. Kamara and T. Moataz, "Boolean searchable symmetric encryption with worst-case sub-linear complexity," in Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, 2017, pp. 94–124.

[13] S. Garg, P. Mohassel, and C. Papamanthou, "Tworam: efficient oblivious ram in two rounds with applications to searchable encryption," in Annual International Cryptology Conference. Springer, 2016, pp. 563–592.

[14] X. Song, C. Dong, D. Yuan, Q. Xu, and M. Zhao, "Forward private searchable symmetric encryption with optimized i/o efficiency," IEEE Transactions on Dependable and Secure Computing, 2018.

[15] K. S. Kim, M. Kim, D. Lee, J. H. Park, and W.-H. Kim, "Forward secure dynamic searchable symmetric encryption with efficient updates," in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 1449–1463.

[16] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious rams," Journal of the ACM (JACM), vol. 43, no. 3, pp. 431–473, 1996.

[17] M. Naveed, S. Kamara, and C. V. Wright, "Inference attacks on property-preserving encrypted databases," in Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, 2015, pp. 644–655.

[18] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in Proceeding2000 IEEE Symposium on Security and Privacy. S&P 2000. IEEE, 2000, pp. 44–55.

[19] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Ros¸u, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for boolean queries," in Annual cryptology conference. Springer, 2013, pp. 353–373.

[20] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in Proceedings of the 2012 ACM conference on Computer and communications security, 2012, pp. 965–976.

[21] D. Boneh and B. Waters, "Constrained pseudorandom functions and their applications," in International conference on the theory and application of cryptology and information security. Springer, 2013, pp. 280–300.

[22] M. Etemad, A. K¨upc¸ ¨u, C. Papamanthou, and D. Evans, "Efficient dynamic searchable encryption with forward privacy," Proceedings on Privacy Enhancing Technologies, vol. 2018, no. 1, pp. 5–20, 2018.

[23] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct randolli functions," in 25th Annual Symposium On Foundations of Computer Science, 1984. IEEE, 1984, pp. 464–479.

[24] M. D. Green and I. Miers, "Forward secure asynchronous messaging from puncturable encryption," in 2015 IEEE Symposium on Security and Privacy. IEEE, 2015, pp. 305 320.

[25] J. Ghareh Chamani, D. Papadopoulos, C. Papamanthou, and R. Jalili, "New constructions for forward and backward private symmetric searchable encryption," in

Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018, pp. 1038–1055.

[26] M. Bellare and P. Rogaway, “Introduction to modern cryptography,” Ucsd Cse, vol. 207, p. 207, 2005.

[27] E. Stefanov, M. Van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas, “Path oram: an extremely simple oblivious ram protocol,” in Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, 2013, pp. 299–310.

[28] D. Cash and S. Tessaro, “The locality of searchable symmetric encryption,” in Annual international conference on the theory and applications of cryptographic techniques. Springer, 2014, pp. 351–368.

[29] G. Asharov, M. Naor, G. Segev, and I. Shahaf, “Searchable symmetric encryption: optimal locality in linear space via two-dimensional balanced allocations,” in Proceedings of the forty-eighth annual ACM symposium on Theory of Computing, 2016, pp. 1101–1114.

[30] I. Demertzis and C. Papamanthou, “Fast searchable encryption with tunable locality,” in Proceedings of the 2017 ACM International Conference on Management of Data, 2017, pp. 1053–1067.

[31] F. Saad Muhi, "A Pseudorandom Binary Generator Based on Chaotic Linear Feedback Shift Register," Iraqi Journal for Electrical and Electronic Engineering vol. 12, pp. 155-160, 2016.