Forward-Private Dynamic Searchable Symmetric Encryption with Efficient Search

Muslum Ozgur Ozmen* Thang Hoang* Attila A. Yavuz*

Abstract

Dynamic Searchable Symmetric Encryption (DSSE) allows to delegate keyword search and file update over an encrypted database via encrypted indexes, and therefore provides opportunities to mitigate the data privacy and utilization dilemma in cloud storage platforms. Despite its merits, recent works have shown that efficient DSSE schemes are vulnerable to statistical attacks due to the lack of forward-privacy, whereas forward-private DSSE schemes suffers from practicality concerns as a result of their extreme computation overhead. Due to significant practical impacts of statistical attacks, there is a critical need for new DSSE schemes that can achieve the forward-privacy in a more practical and efficient manner.

We propose a new DSSE scheme that we refer to as Forward-private Sublinear DSSE (FS-DSSE). FS-DSSE harnesses special secure update strategies and a novel caching strategy to reduce the computation cost of repeated queries. Therefore, it achieves forward-privacy, sublinear search complexity, low end-to-end delay, and parallelization capability simultaneously. We fully implemented our proposed method and evaluated its performance on a real cloud platform. Our experimental evaluation results showed that the proposed scheme is highly secure and highly efficient compared with state-of-the-art DSSE techniques. Specifically, FS-DSSE is one to three magnitude of times faster than forward-secure DSSE counterparts.

1 Introduction

Cloud computing enables massive computation and storage resources that offer a wide range of services. One of the most important cloud facilities is Storage-as-a-Service (SaaS) to allow the client to outsource their data to the cloud and thereby, reducing the data management and maintaining costs. Despite its merits, this service also brings severe privacy issues. Once the client outsources their data to the cloud, they lose control over the privacy of their data. This may leak critical information to the cloud or malicious entities if the cloud is compromised (e.g., a malware). Although standard encryption techniques such as AES can enable the data confidentiality, it also prevents the user from searching or updating information on the cloud and therefore, completely invalidates the benefits of SaaS services.

To address the aforementioned privacy versus data utilization dilemma, Dynamic Searchable Symmetric Encryption (DSSE) techniques have been proposed, which allow the client to encrypt their own data in such a way that it can be later searched and dynamically updated [13]. This is achieved via the creation of an encrypted index containing a set of keyword-file pairs, which associate search/update tokens with the outsourced files encrypted with standard symmetric encryption (e.g., AES). A number of DSSE schemes have been proposed in the literature, each offering various security, functionality, and

*Oregon State University, {ozmenmu, hoangmin, attila.yavuz}@oregonstate.edu

Table 1: Security and (amortized) asymptotic complexity of some state-of-the-art DSSE schemes.

Property/Scheme	Security		Storage		Complexity		Parallelizable
	Update Size Privacy	Forward Privacy	Client State	Encrypted Index	Search Cost	Update Cost	
$\Pi^{\text{dyn}}_{\text{2lev}}$ [4]	✗	✗	$\mathcal{O}(1)$	$\mathcal{O}(N')$	$\mathcal{O}\left(\frac{r+d_w}{p}\right)$	$\mathcal{O}(m'' + r'')$	✓
SUISE [8]	✗	✗	$\mathcal{O}(m')$	$\mathcal{O}(N')$	$\mathcal{O}(r)$	$\mathcal{O}(m'')$	✗
Stefanov et al. [17]	✗	✓	$\mathcal{O}(N'^{\alpha})$	$\mathcal{O}(N')$	$\mathcal{O}\left(\min\left\{\begin{matrix} d_w + \log N' \\ r \log^3 N' \end{matrix}\right\}\right)$	$\mathcal{O}\left(m'' \log^2 N'\right)$	✗
2D-DSSE [21]	✓	✓	$\mathcal{O}(m+n)$	$\mathcal{O}(m \cdot n)$	$\mathcal{O}\left(\frac{n}{p}\right)$	$\mathcal{O}(m)$	✓
Sophos [1]	✗	✓	$\mathcal{O}(m \log n)$	$\mathcal{O}(N')$	$\mathcal{O}(r + d_w)$	$\mathcal{O}(m'')$	✗
FS-DSSE	✓	✓	$\mathcal{O}(1)$	$\mathcal{O}(m \cdot n)$	$\mathcal{O}\left(\frac{r+d_w}{p}\right)$	$\mathcal{O}(m)$	✓

• m and n denote the maximum number of keywords and files, respectively. $m' < m$ and $n' < n$ denote the actual number of keywords and files, respectively. $N' \leq m' \cdot n'$ is # of keyword-file pairs. m'' = # of unique keywords included in an updated file, r = # of files matching search query, p = # of processors, $0 < \alpha < 1$, d_w = # of historical update (add/delete) operations on keyword w, r'' = (accumulated) # of unique keywords being newly added.
We omitted the security parameter κ for analyzed complexity cost.

efficiency trade-offs (e.g., [4, 21, 1, 2, 20, 14]).

Research Gaps. Recently, several studies have shown that the most efficient (sublinear) DSSE schemes leak significant information and are vulnerable to statistical inference analysis (e.g., [15, 3, 22]). For instance, Zhang et al. [22] has demonstrated a file-injection attack strategy which can recover all keywords being searched or updated in DSSE. It has been identified that the forward-privacy is an imperative security feature for modern DSSE schemes to mitigate the impact of such attacks. Specifically, a DSSE is called forward-private if the search query does not reveal any information that can be exploited to determine the content of the files being added or deleted in the future [17]. However, to the best of our knowledge, a very limited number of forward-private DSSE schemes have been proposed, all of which suffer from the efficiency and practicality concerns. It is due to the fact that they either incur polylogarithmic/linear search overhead (e.g., [17, 21]) or rely on Public Key Cryptography (PKC) which is known to be computational costly (e.g., [1]). Therefore, it is vital to develop a new DSSE scheme that offers forward-privacy in a more efficient and practical manner.

Our contributions. In this paper, we propose a new DSSE scheme that offers important features for practical deployment including forward-privacy, sublinear search time with parallelization support, and low client storage. This is achieved by harnessing a secure update strategy on a special encrypted index structure along with a novel caching strategy using a dictionary data structure to partially store the result of previous search queries. We refer our scheme as *Forward-Private and Sublinear DSSE (FS-DSSE)* scheme, with the following desirable properties:

• *High-Speed Search with Full Parallelization:* The proposed scheme offers the lowest search delay among its counterparts. From the asymptotic point of view, our search complexity is *(i)* equivalent to the most efficient yet *forward-insecure* DSSE scheme [4], and *(ii)* lower than state-of-the-art forward-private DSSE schemes (see Table 1). Our proposed scheme is also fully parallellizable, and therefore, can take advantadge of multi-threading techniques offered by the cloud. The experimental evaluation showed that, our search delay was comparable to the most efficient yet forward-insecure DSSE scheme, while it was one to three orders of magnitude faster than its forward-private counterparts [1] (see Section 6).

• *Low Client Storage Overhead:* The proposed scheme features $\mathcal{O}(1)$ client storage overhead, in which the client only needs to store a few symmetric keys. This property allows the proposed scheme to be deployed on mobile devices where the client has a limited memory capacity.

• *High Security:* Our scheme not only achieves forward-privacy as the important security feature, but also can hide the size information of some operations on the encrypted index similar to [21]. Specifically,

the proposed scheme does not leak the number of actual keyword-file pairs in update operation, and the encrypted index size as defined in [17]. Note that such information is leaked in most state-of-the-arts DSSE schemes (except [21]), which might be exploited in statistical attacks.
• *Full-fledged implementation and evaluation on real infrastructure:* We fully implemented the proposed scheme and extensively evaluated its performance on a real computing infrastructure. The experimental result demonstrated that the proposed scheme is highly efficient, which showed the potentials to be deployed on real-world breach-resilient infrastructure.

Table 1 presents the overall comparison in terms of security, operation complexity and storage overhead between our scheme and some recent DSSE schemes.

2 Related Work

The concept of searchable encryption (SE) was first introduced by Song et al. [16]. Subsequently, Curtmola et al. in [6] defined the standard security notion called IND-CKA2 for searchable encryption primitives and presented a IND-CKA2-secure SSE scheme that achieves a sublinear search time. Such static schemes have limited practical applications as they do not support update functionality (dynamism). Kamara et al. in [13] were among the first to define the concept of DSSE, which supports dynamic operations by using an encrypted index, but leaks information during update and is not parallelizable [13]. Later, Kamara et al. provided an improvement over their previous scheme, which leaks less information and is parallelizable [11]. A number of SE schemes have been introduced, each featuring various tradeoffs between the security, functionality and efficiency [4, 8, 17, 1, 21, 14]. For instance, the schemes in [4, 17, 21] support single keyword search with high security. The scheme in [2] supports multi-keyword ranked search, followed by a refinement which offers higher efficiency [18]. The scheme in [20] is specially designed for location-based services to perform geometric range queries on encrypted spatial data. The scheme in [19] supports multi-dimensional range query functionality that enables efficient searches over multi-dimensional tree-based database, but does not support update capability.

Most SE schemes inevitably leak access patterns and therefore, are vulnerable to statistical inference attacks which are first exploited by Islam et al. in [10] that leverages the access pattern (defined as the files that a keyword appears in). Some DSSE schemes (e.g. [9]) are proposed to prevent these attacks but they are neither efficient nor fully secure. Although these statistical attacks can be prevented by Oblivious Random Access Machine (ORAM) [7] or Private Information Retrieval (PIR) [5] techniques, they are known to be extremely costly for practical deployment.

3 Preliminaries

Notation. We denote $\mathcal{E}$ = (Setup, Enc, Dec) as an IND-CPA-secure symmetric encryption where $k \leftarrow \mathcal{E}.\mathsf{Setup}(1^\kappa)$ generates a symmetric key k given a security parameter κ; $c \leftarrow \mathcal{E}.\mathsf{Enc}_k(M)$ returns the ciphertext c of the message M encrypted with key k; $M \leftarrow \mathcal{E}.\mathsf{Dec}_k(c)$ returns the plaintext M of the ciphertext c which is previously encrypted by k. A Pseudo Random Function (PRF) is a polynomial-time computable function, which is indistinguishable from a true random function by any PPT adversary. Table 1 summarizes some notable notations that will be frequently used in our proposed scheme in the next Section.

System Model. Our system model consists of one client and one server. The server is assumed to be honest-but-curious, meaning that it will not inject malicious inputs to compromise the protocol, but

Table 2: Notation in FS-DSSE scheme.

$\mathbf{I}$	Incidence Matrix-based encrypted index
w_i, f_j	Keyword and file with IDs i, j, resp.
r_i	Row Key
u_i	Row Counter (Initially set to 1)
v_i	Row State (0 if row key is revealed to server, 1 otherwise)
c	Encrypted file
n, m	Maximum number of keywords and files in DB, resp.
$D[i]$	Dictionary to store file indexes for each keyword
T_w	Keyword hash table.
$\mathbf{I}[i,j].st$	State bit indicating the last access type† on $\mathbf{I}[i,j]$

$^\dagger\mathbf{I}[i,j].st = 1$ if last access is update, and $\mathbf{I}[i,j].st = 0$ if it is search.

can extract the information from the protocol transcript as much as possible. We assume that the client communicates with the server via a secure channel (e.g., TLS) in the synchronous model.

We present the definition of DSSE as follows:

Definition 1. *A DSSE scheme is a tuple of one algorithm and two protocols DSSE =* (Setup, Search, Update)

1. $(\mathbf{I}, C, \sigma, \mathcal{K}) \leftarrow \mathsf{Setup}(1^\kappa, \mathcal{F})$*: It takes as input a security parameter* κ*, a list of (plaintext) documents and returns an encrypted index* $\mathbf{I}$*, a list of encrypted files* C*, a state* σ*, and a key* $\mathcal{K}$ *.*

2. $(\mathcal{R}, \sigma') \leftarrow \mathsf{Search}_{\mathcal{K}}(w, \mathbf{I}, \sigma)$*: The client inputs the key* $\mathcal{K}$*, a keyword* w *to be searched, the server inputs the encrypted index* $\mathbf{I}$ *and a state* σ*. The protocol outputs the search result* $\mathcal{R}$ *to the client, and outputs a new state* σ *to the server.*

3. $(\mathbf{I}', C', \sigma') \leftarrow \mathsf{Update}_{\mathcal{K}}(f, \mathbf{I}, \sigma, C)$*: The client inputs the key* $\mathcal{K}$*, a file to be updated* f*, the servers inputs the encrypted index* $\mathbf{I}$*, the state* σ *and the list of encrypted files* C*. The protocol outputs to the server a newly updated encrypted index* $\mathbf{I}'$*, the updated state* σ'*, and the updated list of encrypted files* C' *where* f *is added or deleted.*

In the following section, we present our proposed FS-DSSE scheme according to Definition 1.

4 The Proposed Scheme

Intuition. Our main observation is that the search query in standard DSSE will reveal a part of the encrypted index to the server while retrieving the corresponding encrypted files. Therefore, once a keyword is searched again, it is not necessary to repeat the computation on the encrypted index to extract corresponding files that were previously revealed. Instead, one can leverage a more compact and simple data structure (e.g., dictionary) to store file IDs revealed in the first search so that if the same query is repeated, the server will simply get the results stored in this data structure. This strategy will amortize the computation cost incurred in the first search operation and therefore, will make DSSE schemes more efficient. Note that the price to pay for gaining this search efficiency is (in worst case) doubling the server storage overhead.

The second objective is to find a DSSE scheme that efficiently adopts the aforementioned strategy. We observe that, the DSSE scheme in [21] offers a high level of security including forward-privacy with the cost of linear search complexity. This computation cost can be significantly reduced by using our proposed caching strategy mentioned above. Therefore, we take the DSSE scheme in [21] as the base

case to construct FS-DSSE with the caching strategy. We start by giving some brief overview about the data structures used in our scheme, some of which are borrowed from [21]. We refer an interested reader to [21] for detailed description.

4.1 FS-DSSE Data structures

FS-DSSE leverages three different types of data structure as follows:

4.1.1 Incidence matrix

Similar to [21], we construct the encrypted index using an incidence matrix $\mathbf{I}$, which represents the keyword-file relationships via its cell values. Specifically, $\mathbf{I}[i, j] = 1$ if the keyword indexing at row i appears in the file indexing at column j, and $\mathbf{I}[i, j] = 0$ if otherwise. Search and update operations will access a row and a column of $\mathbf{I}$, respectively. We encrypt $\mathbf{I}$ bit-by-bit with IND-CPA encryption. Each cell of $\mathbf{I}$ has a bit state as $\mathbf{I}[i, j].st$ to keep track of the last access operation (search/update) on it. We refer reader to [21] for detailed description.

4.1.2 Hash table

We use a hash table T_w to determine the row indexes assigned to the keywords in $\mathbf{I}$. For simplicity, we assume files are indexed from 1 to n and therefore, it is not required to create a hash table for them. In T_w, we additionally store a counter for each keyword w as $c_i \leftarrow T_w[i].c$, where i is the index of w in T_w, which is incremented after each file update operation to achieve the forward-privacy. We refer reader to [21] for detailed description. We also store a state bit for each keyword w in T_w as $v_i \leftarrow T_w[i].v$, which indicates if the keyword has just been searched ($v_i = 0$) or updated ($v_i = 1$). In FS-DSSE scheme, we store T_w at the server, where all counters inside T_w are encrypted, to achieve $\mathcal{O}(1)$ client storage, in which some of its components will be retrieved first during the search and update operations (see Section 4.2).

4.1.3 Dictionary

FS-DSSE leverages a caching strategy at the server to reduce the computation cost of repeated search operations. We employ a dictionary data structure D to store the search result of the queries when the keyword is first searched. D can be considered as an array of size m, where $D[i]$ stores the list of file IDs which is revealed when searching the keyword indexing at row i in $\mathbf{I}$. D is encrypted with IND-CPA encryption and is updated if there are file operations performed on $\mathbf{I}$ in between the search queries. We present the update policy in the following section to keep D consistent.

4.2 Detailed FS-DSSE Procedures

In this section, we present the detailed procedures in FS-DSSE according to Definition 1 as follows:

Setup: First, the client creates the encrypted files $\mathcal{F}$ and encrypted index and sends them to the server by calling FS-DSSE.Setup procedure. This procedure *(i)* generates three keys $\mathcal{K}$ used to encrypt the files, the incidence matrix, and the counters in the hash table, *(ii)* extracts keywords from the input files, each being assigned to a row index via the hash table T_w, and *(iii)* sets the corresponding value for each $\mathbf{I}[i, j]$. Note that a counter for each keyword is also stored in T_w, which will be used to derive a key to achieve the forward-privacy during update. Finally, the client encrypts the incidence matrix $\mathbf{I}$,

$(\mathbf{I}, \mathcal{C}, \sigma, \mathcal{K}) \leftarrow$ FS-DSSE.Setup$(1^\kappa, \mathcal{F})$: Create encrypted index

$k_1 \leftarrow \mathcal{E}.\text{Gen}(1^\kappa)$ and $(k_2, k_3) \xleftarrow{\$} \{0,1\}^\kappa$
Set $\mathcal{K} \leftarrow \{k_1, k_2, k_3\}$
Extract keywords $(w_1, \ldots, w_{m'})$ from $\mathcal{F} = \{f_{id_1}, \ldots, f_{id_{n'}}\}$
Set counter $u_i = 1$, state $v_i = 1$ for each w_i in hash table T_w
Set $\mathbf{I}[i, j] \leftarrow 1$ where i is the keyword index in T_w and $0 \le j \le n$ if f_j contains keyword i
Generate row keys $r_i \leftarrow \text{PRF}(k_2 \| i \| u_i)$ for $1 \le i \le m$
Encrypt each row of $\mathbf{I}$ with key r_i for $1 \le i \le m$
Encrypt counters in hash table T_w with k_3
Encrypt all files $\mathcal{F}$ with k_1 as $\mathcal{C} = \{c_j : c_j \leftarrow \mathcal{E}.\text{Enc}_{k_1}(f_j)\}$
return $(\mathbf{I}, \mathcal{C}, \sigma, \mathcal{K})$, where $\sigma \leftarrow T_w$ (Send $(\mathbf{I}, \mathcal{C}, \sigma)$ to server)

Figure 1: FS-DSSE Setup algorithm.

counters in hash table T_w and all files $\mathcal{F}$, and sends them to the server, while keeping the key $\mathcal{K}$ secret.

Search: To search a keyword w, the client first requests the encrypted counter of w stored in T_w. Then, they send a search token containing the row index i and the row key r_i derived from the counter, to the server. If the keyword is being searched for the first time, then the server decrypts the whole row $\mathbf{I}[i, *]$ with r_i, adds all column indexes j, where $\mathbf{I}[i, j] = 1$, to the dictionary $D[i]$, and encrypts $D[i]$[1]. The server returns the corresponding encrypted files matching with such indexes to the client.

If a previously-searched keyword is searched, the server retrieves indexes of corresponding encrypted files by simply decrypting $D[i]$. It is important to note that $D[i]$ might need to be updated,

$(\mathcal{R}, \sigma') \leftarrow$ FS-DSSE.Search$_{\mathcal{K}}(w, \mathbf{I}, \sigma)$
Client:
Let i be index of w in T_w
Download the counter u_i of keyword w_i in T_w
Decrypt u_i with k_3 and generate key $r_i \leftarrow \text{PRF}(k_2 \| i \| u_i)$
Send (i, v_i', r_i) to server
Server: On receive (i, v_i', r_i):
if i is first-time searched **then**
    Let $\mathbf{I}'[i, *]$ be the decryption of $\mathbf{I}[i, *]$ using key r_i
    $D[i] \leftarrow \{j : \mathbf{I}'[i, j] = 1\}$
    Encrypt $D[i]$
else
    Decrypt $D[i]$
    Let $J = \{j : \mathbf{I}[i, j].st = 1\}$
    for each j in J **do**
        Let $\mathbf{I}'[i, j]$ be the decryption of $\mathbf{I}[i, j]$ using key r_i
        Add j to $D[i]$ if $\mathbf{I}'[i, j] = 1$, or delete j if otherwise
Send $\mathcal{R} = \{c_j : j \in D[i]\}$ to client
Re-encrypt $D[i]$, set $\mathbf{I}[i, *].st = 0$ and $T_w[i].v \leftarrow 0$
return $(\mathcal{R}, \sigma')$, where $\sigma' \leftarrow T_w$ with i-th entry being updated
Client: On receive $\mathcal{R}$
$f_j \leftarrow \mathcal{E}.\text{Dec}_{k_1}(c_j)$ for each $c_j \in \mathcal{R}$

Figure 2: FS-DSSE Search protocol.

[1]Server encrypts D with a self-generated key just to preserve the data privacy against outside attackers.

given that there are some file update operations on $\mathbf{I}$ that happened after the latest search on w_i. This is achieved by checking the state bit $\mathbf{I}[*,*].st$. Specifically, if $\mathbf{I}[i,j].st = 1$, then the server decrypts $\mathbf{I}[i,j]$ and adds the entry j to $D[i]$ if $\mathbf{I}[i,j] = 1$ (or deletes j if $\mathbf{I}[i,j] = 0$).

The use of data structure D enables FS-DSSE to have an amortized sublinear search complexity. Specifically, the computation cost of the first query is $\mathcal{O}(n)$ while that of repeated queries is $\mathcal{O}(r)$, where r is the result size of the first query. The amortized cost is $\mathcal{O}(r + d_w)$, where d_w is the number of updates, after n search repetitions.

Update: Given an updated file f_j, the client extracts the updated keywords and creates an unencrypted column $\mathbf{I}[*,j]$, which represents the relationship between f_j with all the keywords in DB. The client then generates m row keys r_i according to corresponding counters stored in T_w at the server. To achieve the forward-privacy, the client must encrypt $\mathbf{I}[*,j]$ with fresh keys, which are unknown to the server. This can be done by generating the row key with the incremented counter, given that the key generated with the current counter has been previously revealed to the server during the previous searches. Finally, the client sends the encrypted column and the encrypted files to the server, where the encrypted index and encrypted database are updated accordingly.

$(\mathbf{I}', C', \sigma') \leftarrow \mathsf{FS\text{-}DSSE.Update}_{\mathcal{K}}(f_j, \mathbf{I}, \sigma, C)$

Client:

1: Set $\bar{\mathbf{I}}[*,j] \leftarrow 0$ and $c \leftarrow NULL$
2: Download all counters u_i and states v_i $(1 \le i \le m)$ in T_w
3: Download state column $\mathbf{I}[*,j]$.st
4: Let $V = \{i : v_i = 0\}$
5: Extract keywords $(w_1, \ldots, w_t)$ from f_j
6: Decrypt counters u_i with k_3, and set $u_i \leftarrow u_i + 1$ for each $i \in V$
7: Generate keys $r_i \leftarrow \mathsf{PRF}(k_2 || i || u_i)$
8: Set $\bar{\mathbf{I}}[x_i, j] \leftarrow 1$ for $1 \le i \le t$, where x_i is index of w_i in T_w
9: Encrypt f_j as $c_j \leftarrow \mathcal{E}.\mathsf{Enc}_{k_1}(f_j)$
10: Encrypt each row $\bar{\mathbf{I}}[i,j]$ with row keys r_i, for $1 \le i \le m$
11: Let $\vec{u} \leftarrow (u_1, \ldots, u_n)$, where each u_i is encrypted with k_3
12: Send $(\bar{\mathbf{I}}[*,j], c_j, j, \vec{u})$ to server

Server: On receive $(\bar{\mathbf{I}}[*,j], c_j, j, \vec{u})$

13: $T_w[i].u \leftarrow u_i$ and set $T_w[i].v \leftarrow 1$ for $1 \le i \le m$
14: Set $\mathbf{I}[*,j] \leftarrow \bar{\mathbf{I}}[*,j]$ and $\mathbf{I}[*,j]$.st $\leftarrow 1$
15: **return** $(\mathbf{I}', C', \sigma')$, where $\mathbf{I}' = \mathbf{I}$ with column j updated, $C' = C$ updated with c_j, $\sigma' = T_w$ with counters and states updated

Figure 3: FS-DSSE Update protocol.

5 Security Analysis

The security of DSSE schemes is defined with the dynamic IND-CKA2 notion presented in [13], which is refined from static IND-CKA2 in [6] that captures information leakages from search and update tokens. This notion pertains to leakage functions $\mathcal{L}$, which quantifies precisely what information is leaked from the ciphertext and the tokens. For the sake of completeness, we provide the detailed leakages functions and IND-CKA2 definition in Appendix. Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be a leakage function which captures information

leakage in FS-DSSE including the maximum number of keywords and files, file IDs, the size of each file and access patterns (see Appendix for detailed description). FS-DSSE achieves the following security.

Theorem 1. *FS-DSSE is* $(\mathcal{L}_1, \mathcal{L}_2)$*-IND-CKA2 secure.*

Proof (sketch). We refer readers to [21]. □

Theorem 2. *FS-DSSE is forward-private.*

Proof (sketch). The forward-privacy implies that the content of the updated files should not be linked with any previous search operations. FS-DSSE harnesses the update strategy in [21], in which the update operation uses all fresh row keys which were never revealed to the server. It is achieved by increasing the keyword counter maintained in the hash table T_w. □

6 Performance Evaluation

We first describe our implementation details, experimental setup and evaluation metrics with state-of-the-art schemes. We then present the performance of the proposed scheme along with in-depth comparison.

6.1 Experimental Setup and Evaluation Metrics

Software library. Our implementation uses the following libraries: tomcrypt for cryptographic primitives; Intel AES-NI for AES-CTR encryption acceleration; google-sparsehash for hash table; zeroMQ for network communication.

Hardware setting. We used a Desktop with Intel Xeon E3-1231v3 @ 3.40 GHz and 16GB RAM at the client side. We leveraged our on-campus computing platform equipped with 32 CPUs @ 2.70GHz, and 512GB RAM as the server.

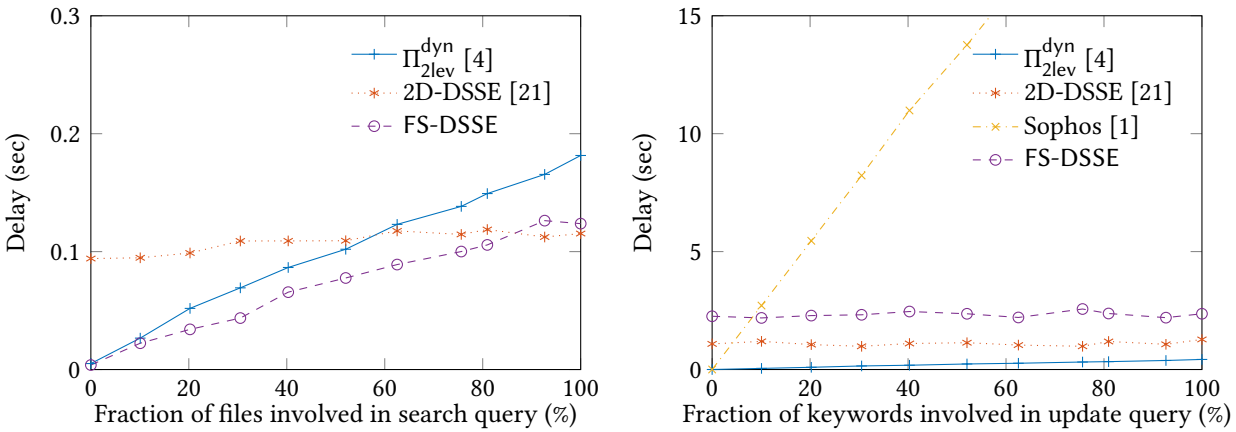
Database. We used the full Enron email dataset, including 517401 files and 1728833 distinct keywords according to the standard tokenization method. The total number of keyword-file pairs is around 10^8.

Comparison. We compare FS-DSSE with some state-of-the art DSSE schemes including the scheme in [21] (called as 2D-DSSE), Sophos [1] and $\Pi^{\mathsf{dyn}}_{\mathsf{2lev}}$ [4]. For Sophos and 2D-DSSE, we used their public open-source since their implementation setting is same with ours (C/C++), but we only simulated $\Pi^{\mathsf{dyn}}_{\mathsf{2lev}}$ due to its lack of open-source C/C++ implementation.

Evaluation Metrics. We evaluated all schemes according to search and update delay. To compare the search time, we searched from least-common keywords (e.g., only appears in 1,2 files) to most-common keywords (e.g., appears in 100% files) with 10% intervals. We applied the same strategy to compare the update times. We present the cost breakdown of search and update operations for our scheme.

6.2 Performance Evaluation and Comparison

We present the end-to-end delays for search and update operations of FS-DSSE and its counterparts in Fig. 4a and 4b, respectively. One may notice that FS-DSSE achieves the fastest search time among the counterparts in most cases, where it is even 1.4× faster than the most efficient yet forward-insecure scheme ($\Pi^{\mathsf{dyn}}_{\mathsf{2lev}}$).



(a) Keyword search

(b) File update

Figure 4: The latency of FS-DSSE and its counterparts. In (a), we excluded Sophos scheme [1] since its plot is beyond the y-axis limit.

Since search complexity of 2D-DSSE is *linear with the maximum number of files in database,* its search time was constant in any size of search query results. Due to the sublinear property, FS-DSSE is faster than 2D-DSSE, for the most of the keywords, where it is up to 35× faster when searching least-common keywords. 2D-DSSE is only 8ms faster than FS-DSSE when searching the most-common keywords. The search time of Sophos could not fit into this graph due to its heavy public key operations. Specifically, we measured that end-to-end delay to be around 20 seconds even when searching for least-common keywords.

FS-DSSE has a constant update time similar to 2D-DSSE for all files with different number of keywords associated to, since they are both linear with the number of keywords in the database. The latency difference between them is that in FS-DSSE, we store T_w at the server instead of the client as in 2D-DSSE to achieve $\mathcal{O}(1)$ client storage, which incurs an extra round of communication overhead. On the other hand, update time of Π^{dyn}_{2lev} is the fastest. It is due to the fact that Π^{dyn}_{2lev} scheme has a smaller encrypted index size and therefore, random access is performed on a smaller memory region. Moreover, the random access cost dominates the total update cost in our scheme. The Update cost of Sophos increases linearly with the number of keywords associated with the updated file. The cost is lower than FS-DSSE when file is associated with 8.18% of the total number of keywords and it is higher for the rest. Since the update cost of Sophos is dominated with the public key operations performed at the client-side, when the file is associated with 100% of keywords, it is 11.39× slower than FS-DSSE.

We also studied the detailed cost of search operation in FS-DSSE to observe the factors that had the most impact on the total delay. It is depicted in Fig. 5 that total delay is mostly dominated by the server computation with the increasing number of files associated with the keyword. Even though server performs symmetric key encryption/decryption in parallel using 32 cores, it still dominates the total time since our network speed is extremely fast and the size of the dictionary D, that stores the indexes, highly increases. Since generating search token does not incur any expensive operations, the client computation cost is negligible.

Update cost of our scheme is constant as the number of keywords associated with the updated file increases (see Fig. 4b). Our measurements showed that update cost of our scheme is dominated by the I/O access due to non-contiguous memory access. The second major dominating cost of update operation is the client computation, which requires the re-encryption of the encrypted index column

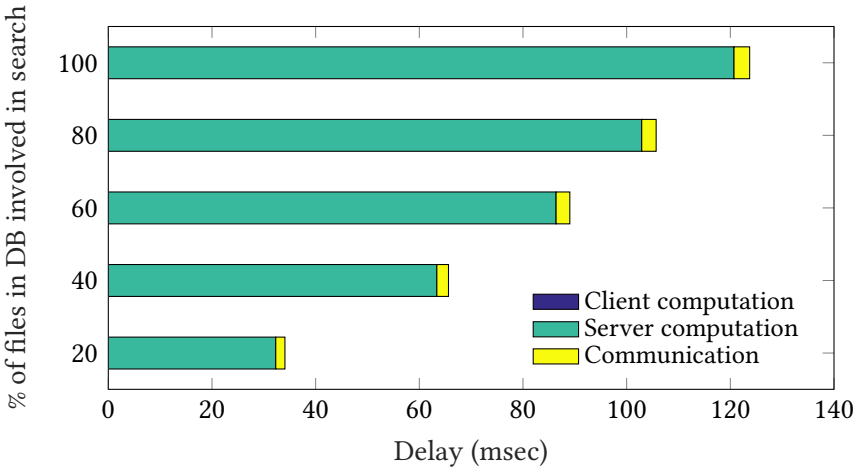


Figure 5: Cost breakdown of search query in FS-DSSE.

with new keys to achieve forward-privacy. Since the network speed is fast in this experiment, the communication cost is lower than other factors.

7 Conclusion

In this paper, we presented a new DSSE scheme that offers forward-privacy, sublinear search complexity and low end-to-end delay simultaneously. This is achieved by harnessing forward-private update strategies on 2-dimensional encrypted index along with a novel caching strategy to reduce the computation time incurred in repeated search queries. Our experimental results showed that the proposed scheme is highly secure and it outperforms state-of-the-art forward-private counterparts.

References

[1] R. Bost. Sophos: Forward secure searchable encryption. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1143–1154, New York, NY, USA, 2016. ACM.

[2] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou. Privacy-preserving multi-keyword ranked search over encrypted cloud data. *IEEE Transactions on Parallel and Distributed Systems*, 25(1), Jan 2014.

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

[4] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawcyk, M.-C. Rosu, and M. Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *21th Annual Network and Distributed System Security Symposium — NDSS 2014*. The Internet Society, February 23-26, 2014.

[5] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *Journal of the ACM (JACM)*, 45(6):965–981, 1998.

[6] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proceedings of the 13th ACM conference on Computer and communications security*, CCS '06, pages 79–88. ACM, 2006.

[7] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, 1996.

[8] F. Hahn and F. Kerschbaum. Searchable encryption with secure and efficient updates. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 310–320. ACM, 2014.

[9] T. Hoang, A. Yavuz, and J. Guajardo. Practical and secure dynamic searchable encryption via oblivious access on distributed data structure. In *Proceedings of the 32nd Annual Computer Security Applications Conference (ACSAC)*. ACM, 2016.

[10] M. S. Islam, M. Kuzu, and M. Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *Annual Network and Distributed System Security Symposium – NDSS*, volume 20, page 12, 2012.

[11] S. Kamara and C. Papamanthou. Parallel and dynamic searchable symmetric encryption. In *Financial Cryptography and Data Security*, pages 258–274. Springer, 2013.

[12] S. Kamara and C. Papamanthou. Parallel and dynamic searchable symmetric encryption. In *Financial Cryptography and Data Security (FC)*, volume 7859 of *Lecture Notes in Computer Science*, pages 258–274. Springer Berlin Heidelberg, 2013.

[13] S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 965–976, New York, NY, USA, 2012. ACM.

[14] M. Li, S. Yu, K. Ren, W. Lou, and Y. T. Hou. Toward privacy-assured and searchable cloud data storage services. *IEEE Network*, 27(4):56–62, July 2013.

[15] D. Pouliot and C. V. Wright. The shadow nemesis: Inference attacks on efficiently deployable, efficiently searchable encryption. In *Proceedings of the 2016 ACM Conference on Computer and Communications Security*. ACM, 2016.

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

[17] E. Stefanov, C. Papamanthou, and E. Shi. Practical dynamic searchable encryption with small leakage. In *21st Annual Network and Distributed System Security Symposium — NDSS 2014*. The Internet Society, February 23-26, 2014.

[18] W. Sun, B. Wang, N. Cao, M. Li, W. Lou, Y. T. Hou, and H. Li. Verifiable privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking. *IEEE Transactions on Parallel and Distributed Systems*, 25(11):3025–3035, Nov 2014.

[19] B. Wang, Y. Hou, M. Li, H. Wang, and H. Li. Maple: Scalable multi-dimensional range search over encrypted cloud data with tree-based index. In *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '14, pages 111–122, New York, NY, USA, 2014. ACM.

[20] B. Wang, M. Li, and L. Xiong. Fastgeo: Efficient geometric range queries on encrypted spatial data. *IEEE Transactions on Dependable and Secure Computing*, PP(99):1–1, 2017.

[21] A. A. Yavuz and J. Guajardo. Dynamic searchable symmetric encryption with minimal leakage and efficient updates on commodity hardware. *IACR Cryptology ePrint Archive*, 2015, 2015.

[22] Y. Zhang, J. Katz, and C. Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In *25th USENIX Security '16*, pages 707–720, Austin, TX, 2016.

Appendix

In [13], the *search and file-access patterns* in DSSE are defined as follows:

- Given search query w at time t, *the search pattern* $\mathcal{P}(\mathbf{I}, \text{Query}, t)$ is a binary vector of length t with a 1 at location i if the search time $i \leq t$ was for w, and 0 otherwise. The search pattern indicates whether the same keyword has been searched in the past or not.

- Given search query w_i at time t, *the file-access pattern* $\Delta(\mathbf{I}, \mathcal{F}, w, t)$ is identifiers $\mathcal{I}_w$ of files having w_i.

In [21], leakage functions are defined as follows:

Definition 2 (Leakage functions [21])**.** *Let* $(\mathcal{L}_1, \mathcal{L}_2)$ *be leakage functions such that:*

1. $(m, n, \mathcal{I}, \langle |f_{id_1}|, \ldots, |f_{id_n}| \rangle) \leftarrow \mathcal{L}_1(\mathbf{I}, \mathcal{F})$*: Given the index* $\mathbf{I}$ *and the set of files* $\mathcal{F}$ *(including their identifiers),* $\mathcal{L}_1$ *outputs the maximum number of keywords* m*, the maximum number of files* n*, the identifiers* $\mathcal{I} = \{id_1, \ldots, id_n\}$ *of* $\mathcal{F}$ *and the size of file* $|f_{id_j}|$ *for* $1 \leq j \leq n$ *(which also implies the size of its corresponding ciphertext* $|c_{id_j}|$*).*
2. $(\mathcal{P}(\mathbf{I}, Query, t), \Delta(\mathbf{I}, \mathcal{F}, w, t)) \leftarrow \mathcal{L}_2(\mathbf{I}, \mathcal{F}, w, t)$*: Given the index* δ*, the set of files* $\mathcal{F}$ *and a keyword* w *for a search operation at time* t*, it outputs the search pattern* $\mathcal{P}$ *and file-access pattern* Δ*.*

Definition 3 (IND-CKA2 Security [6, 13])**.** *Let* $\mathcal{A}$ *be a stateful adversary and* $\mathcal{S}$ *be a stateful simulator. Consider the following probabilistic experiments:*

$\mathbf{Real}_{\mathcal{A}}(\kappa)$*: The challenger executes* $\mathcal{K} \leftarrow \mathsf{Setup}(1^\kappa)$*.* $\mathcal{A}$ *produces* $(\mathbf{I}, \mathcal{F})$ *and receives* $(\gamma, C) \leftarrow \mathsf{Enc}_{\mathcal{K}}(\delta, \mathcal{F})$ *from the challenger.* $\mathcal{A}$ *makes a polynomial number of adaptive queries* $\mathsf{Query} \in (w, f_{id}, f_{id'})$ *to the challenger. If* $\mathsf{Query} = w$ *is a keyword search query then* $\mathcal{A}$ *receives a search token* $\tau_w \leftarrow \mathsf{SearchToken}(\mathcal{K}, w)$ *from the challenger. If* $\mathsf{Query} = f_{id}$ *is a file addition query then* $\mathcal{A}$ *receives an addition token* $(\tau_f, c) \leftarrow \mathsf{AddToken}(\mathcal{K}, f_{id})$ *from the challenger. If* $\mathsf{Query} = f_{id'}$ *is a file deletion query then* $\mathcal{A}$ *receives a deletion token* $\tau'_f \leftarrow \mathsf{DeleteToken}(\mathcal{K}, f_{id'})$ *from the challenger. Eventually,* $\mathcal{A}$ *returns a bit* b *that is output by the experiment.*

$\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(\kappa)$*:* $\mathcal{A}$ *produces* $(\delta, \mathcal{F})$*. Given* $\mathcal{L}_1(\mathbf{I}, \mathcal{F})$*,* $\mathcal{S}$ *generates and sends* (γ, C) *to* $\mathcal{A}$*.* $\mathcal{A}$ *makes a polynomial number of adaptive queries* $\mathsf{Query} \in (w, f_{id}, f_{id'})$ *to* $\mathcal{S}$*. For each query,* $\mathcal{S}$ *is given* $\mathcal{L}_2(\mathbf{I}, \mathcal{F}, w, t)$*. If* $\mathsf{Query} = w$ *then* $\mathcal{S}$ *returns a simulated search token* τ_w*. If* $\mathsf{Query} = f_{id}$ *or* $\mathsf{Query} = f_{id'}$*,* $\mathcal{S}$ *returns a simulated addition token* τ_f *or deletion token* τ'_f *,respectively. Eventually,* $\mathcal{A}$ *returns a bit* b *that is output by the experiment.*

A DSSE *is said to be* $(\mathcal{L}_1, \mathcal{L}_2)$*-secure against adaptive chosen-keyword attacks (CKA2-security) if for all* PPT *adversaries* $\mathcal{A}$*, there exists a* PPT *simulator* $\mathcal{S}$ *such that*

$$|\Pr[\mathbf{Real}_{\mathcal{A}}(\kappa) = 1] - \Pr[\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(\kappa) = 1]| \quad \leq \quad \mathsf{neg}(\kappa).$$