A Secure Searchable Encryption Framework for Privacy-Critical Cloud Storage Services

Thang Hoang, Attila A. Yavuz, *Member, IEEE* and Jorge Guajardo

Abstract—Searchable encryption has received a significant attention from the research community with various constructions being proposed, each achieving asymptotically optimal complexity for specific metrics (e.g., search, update). Despite their elegance, the recent attacks and deployment efforts have shown that the optimal asymptotic complexity might not always imply practical performance, especially if the application demands a high privacy. In this article, we introduce a novel Dynamic Searchable Symmetric Encryption (DSSE) framework called Incidence Matrix (IM)-DSSE, which achieves a high level of privacy, efficient search/update, and low client storage with actual deployments on real cloud settings. We harness an incidence matrix along with two hash tables to create an encrypted index, on which both search and update operations can be performed effectively with minimal information leakage. This simple set of data structures surprisingly offers a high level of DSSE security while achieving practical performance. Specifically, IM-DSSE achieves forward-privacy, backward-privacy and size-obliviousness simultaneously. We also create several DSSE variants, each offering different trade-offs that are suitable for different cloud applications and infrastructures. We fully implemented our framework and evaluated its performance on a real cloud system (Amazon EC2). We have released IM-DSSE as an open-source library for wide development and adaptation.

Index Terms—Privacy-enhancing technologies, private cloud services; dynamic searchable symmetric encryption.

1 Introduction

The rise of cloud storage and computing services provides vast benefits to the society and IT industry. One of the most important cloud services is data Storage-as-a-Service (SaaS), which can significantly reduce the cost of data management via continuous service, expertise and maintenance for resource-limited clients such as individuals or small/medium businesses. Despite its benefits, SaaS also brings significant security and privacy concerns to the user. That is, once a client outsource his/her own data to the cloud, sensitive information (e.g., email) might be exploited by a malicious party (e.g., malware). Although standard encryption schemes such as Advanced Encryption Standard (AES) can provide confidentiality, they also prevent the client from querying encrypted data from the cloud. This privacy versus data utilization dilemma may significantly degrade the benefits and usability of cloud systems. Therefore, it is vital to develop privacy-enhancing technologies that can address this problem while retaining the practicality of the underlying cloud service.

Searchable Symmetric Encryption (SSE) [1] enables a client to encrypt data in such a way that they can later perform keyword searches on it. These encrypted queries are performed via "search tokens" [2] over an encrypted index which represents the relationship between search token (keywords) and encrypted files. A prominent application of SSE is to enable privacy-preserving keyword search on the cloud (e.g., Amazon S3), where a data owner can outsource a collection of encrypted files and perform keyword searches on it without revealing the file and query contents [3]. Preliminary SSE schemes (e.g., [1], [4]) only provide search-only functionality on static data (i.e., no dynamism), which strictly limits their applicability due to the lack of update capacity. Later, several Dynamic Searchable Symmetric Encryption (DSSE) schemes (e.g., [3], [5]) were proposed that permit the user to add and delete files after the system is set up. To the best of our knowledge, there is *no* single DSSE scheme that outperforms *all* the other alternatives in terms of *all* the aforementioned metrics: privacy (e.g., information leakage), performance (e.g., search, update delay), storage efficiency and functionality. In the following, we first provide an overview on DSSE research and then, outline our research objectives and contributions toward addressing some of the limitations of the state-of-the-arts.

1.1 Related Work

SSE was first introduced by Song *et al*. [4]. Curtmola *et al*. [1] proposed a sublinear SSE scheme and introduced the security notion for SSE called *adaptive security against chosen-keyword attacks (CKA2)*. Refinements of [1] have been proposed which offer extended functionalities (e.g., [6], [7]). However, the static nature of those schemes limited their applicability to applications that require dynamic file collections. Kamara *et al*. were among the first to develop a DSSE scheme in [3] that could handle dynamic file collections via an encrypted index. However, it leaks significant information for updates and it is not parallelizable. Kamara *et al*.

- *Thang Hoang is with the School of EECS, Oregon State University, Corvallis, OR, 97331. E-mail: hoangmin@oregonstate.edu*
- *Attila A. Yavuz is with the Department of Computer Science and Engineering, University of South Florida, Tampa, FL, 33620. E-mail: attilaayavuz@usf.edu.*
- *Jorge Guajardo is with Robert Bosch RTC—LLC, Pittsburgh, PA, 12503. E-mail: Jorge.GuajardoMerchan@us.bosch.com.*
- *Part of this work was done while the first two authors were visiting the Bosch RTC—LLC, Pittsburgh, PA and the second author was employed at Oregon State University. E-mail: attila.yavuz@oregonstate.edu.*

[8] proposed a DSSE scheme, which leaked less information than that of [3] and it was parallelizable. Recently, a series of new DSSE schemes (e.g., [2], [5], [9], [10], [11], [12]) have been proposed which offer various trade-offs between security, functionality and efficiency properties such as small leakage (e.g., [2]), scalable searches with extended query types (e.g., [12], [13], [14], [15]), or high efficiency (e.g., [9]). Inspired by the work from [5], Kamara *et al*. in [12] proposed a new sublinear DSSE scheme which supports more complex queries such as disjunctive and boolean queries.

Forward-private DSSE schemes. Zhang *et al*. in [16] showed that new DSSE constructions should offer the forward-privacy property to mitigate the impact of practical attacks. After the preliminary IM-DSSE scheme was introduced in [17], several forward-private DSSE schemes achieving high efficiency in terms of asymptotic complexity and actual performance have been proposed. Specifically, Bost *et al*. in [11] proposed Sophos, which offers forward-privacy using an asymmetric primitive (i.e., trapdoor permutation). Rizomiliotis *et al*. in [18] leverages Oblivious Random Access (ORAM) techniques (e.g., [19]) to enable forward-privacy.

Recently, several forward-private DSSE schemes only relying on symmetric primitives have been proposed (e.g., [20], [21], [22], [23]), some of which offer parallelism (e.g., [20], [21], [23]), and improved I/O access with computation efficiency using a caching strategy (e.g., [20], [21], [22]). For example, Lai *et al*. in [20] modeled the relationship between keywords and files in DSSE as bipartite graphs. The authors also proposed a novel data structure called *cascaded triangles*, which offers parallelism and efficient update (add/delete). Kim *et al*. in [21] leveraged two hash tables to integrate forward index and inverted index together in the form of encrypted index, which offers efficient update with direct deletion. Several forward-private DSSE schemes, which offer extended query functionalities such as boolean query [12], similarity search [15] were also proposed. Bost *et al*. in [24] proposed some (single-keyword) DSSE schemes that achieve both forward-privacy and backward-privacy with optimal asymptotic complexity using asymmetric primitives. In Table 1, we outline the asymptotic complexity and the security of typical standard (single-keyword) DSSE schemes.

Access Pattern Leakage in DSSE. Due to the deterministic keyword-file relationship, most traditional DSSE schemes (*including our framework in this article*) leak search and access patterns defined in §4 which are vulnerable to statistical inference attacks. A number of attacks (e.g., [16], [25], [26], [27], [28]) have been demonstrated. Several DSSE schemes have been proposed to deal with such leakages (e.g., [29], [30]) but they are neither efficient nor provably secure. ORAM techniques (e.g., [19]) can hide search and access patterns in DSSE. Despite a lot of progress on these techniques, their costs are still extremely high to be applied to DSSE in practice [31].

1.2 Motivation and Research Objectives

Although a number of DSSE schemes have been introduced in the literature, most of them only provide a theoretical asymptotic analysis[1] and, in some cases, merely a prototype implementation. The lack of experimental performance evaluations on real platforms poses a significant difficulty in assessing the application and practicality of proposed DSSE schemes, as the impacts of security vulnerability, hidden computation costs, multi-round communication delay and storage blowup might be overlooked. For instance, most efficient DSSE schemes (e.g., [5], [10]) are vulnerable to file-injection attacks, which have been shown to be easily conducted even by a semi-honest adversary in practice, especially in the personal email scenario. Although several forward-secure DSSE schemes with an optimal asymptotic complexity have been proposed, they incur either very high delay due to public-key operations (e.g., [11]), or significant storage blow-up at both client and server side (e.g., [2]), and therefore, their ability to meet actual need of real systems in practice is still unclear.

There is a significant need for a DSSE scheme that can achieve a high level of security with a well-quantified information leakage, while maintaining a performance and functionality balance between the search and update operations. More importantly, it is critical that the performance of proposed DSSE should be experimentally evaluated in a realistic cloud environment with various parameter settings, rather than merely relying on asymptotic results. The investigation of alternative data structures and their optimized implementations on commodity hardware seem to be the key factors towards achieving these objectives.

1.3 Our Contributions

In this article, towards filling the gaps between theory and practice in DSSE research community, we introduce IM-DSSE, a fully-implemented Incidence Matrix-based DSSE framework which favors desirable properties for realistic privacy-critical cloud systems including high security against practical attacks and low end-to-end delay. In this framework, we provide the full-fledged implementation of our preliminary DSSE scheme proposed in [17], as well as extended schemes, which are specially designed to meet various application requirements and cloud data storage-as-a-service infrastructures in practice.

Improvements over Preliminary Version: This article is the extended version of [17] which includes the following improvements: (i) We propose extended DSSE schemes which are more compatible with the cloud SaaS infrastructure and offer backward-privacy at the cost of bandwidth overhead. (ii) As a significant improvement over the preliminary version, we provide a comprehensive DSSE framework, where our preliminary DSSE scheme in [17] and all its variants are fully implemented. We fully deployed our framework on Amazon EC2 cloud and provided a much more comprehensive performance analysis of each scheme with different hardware and network settings. (iii) Finally, we have released our framework for public use and improvement.

Desirable properties: IM-DSSE offers ideal features for privacy-critical cloud systems as follows.

[1]One noticeable outlier is [5], which provides a standalone implementation.

- *Highly secure against File-Injection Attacks:* IM-DSSE offers *forward privacy* (see [2] or §4 for definition) which is an imperative security feature to mitigate the impact of practical file-injection attacks [11], [16]. Only a limited number of DSSE schemes offer this property (i.e., [2], [11], [15], [18], [20], [21], [22], [23]), some of which incur high client storage with costly update (e.g., [2]) or high delay, due to oblivious access techniques (e.g., [18]) and public-key operations (e.g., [11]). Additionally, IM-DSSE offers *size-obliviousness* property, where it hides all size information involved with the encrypted index and update query including *(i)* update query size (i.e., number of unique keywords in the updated file); *(ii)* and the number of keyword-file pairs in the database. One of the IM-DSSE variants achieves *backward privacy* defined in [2]. We notice that Bost *et al*. in [24] have recently proposed a new DSSE scheme that can achieve all these security properties with padding. This scheme leverages asymmetric primitives (e.g., puncturable encryption [32]), which might incur high computation cost. Our scheme relies on symmetric primitives but with the cost of an extra communication overhead.
- *Updates with Improved Features*: (i) IM-DSSE allows to *directly* update keywords of an existing file without invoking the file delete-then-add operation sequence. The update in IM-DSSE also leaks minimal information, where it does not leak timing information (i.e., all updates take the same amount of time) and how many keywords are being added/deleted in the updated file. (ii) The encrypted index of our schemes does not grow with update operations and, therefore, it does not require re-encryption due to frequent updates. This is more efficient than some alternatives (e.g., [2]) in which the encrypted index can grow linearly with the number of deletions.
- *Fully Parallelizable*: IM-DSSE also supports parallelization (as in [5], [20], [21], [23]) for both update and search operations and, therefore, it takes full advantage of modern computing architecture to minimize the delay of cryptographic operations. Experiments on Amazon cloud indicates that the search latency of our framework is highly practical and mostly dominated by the client-server communication (see §5).
- *Detailed experimental evaluation and open-source framework*: We deployed IM-DSSE in a realistic cloud environment (i.e., Amazon EC2) to assess the practicality of our framework. We experimented with different database sizes and investigated the impacts of network condition and storage unit on the overall performance. We also evaluated the performance of IM-DSSE on a resource-limited mobile client. We give a comprehensive cost breakdown analysis to highlight the main factors contributing to the overall delay in all these settings. We have released the implementation of our framework to public to provide opportunities for broad adaptation and testing (see §5).

2 IM-DSSE Framework

2.1 DSSE for Cloud Storage Applications

Recent data breach incidents (e.g., Equifax, Apple iCloud, Ashley Madison) have shown that it is extremely important to achieve data confidentiality on the cloud. While a number of DSSE schemes have been introduced, we observed that they are not ready to be deployed on current cloud storage architecture due to their vulnerability against practical attacks (as discussed in §1). Moreover, most prior schemes require special computation services, which might significantly increase the monetary cost of deployment and are incompatible with storage-only cloud services such as Dropbox, Google Drive. In this paper, we focus on addressing these problems by proposing a new DSSE framework which can achieve a high security and be compatible with current cloud infrastructure. Potential applications of our framework include, but not limited to, privacy-preserving email and file storage services, where the client can be able to store, search and update their sensitive data (e.g., email, photos, transactions) on the cloud without the service provider knowing what have been stored and retrieved. Figure 1 illustrates the overview of our IM-DSSE framework for file-storage applications.

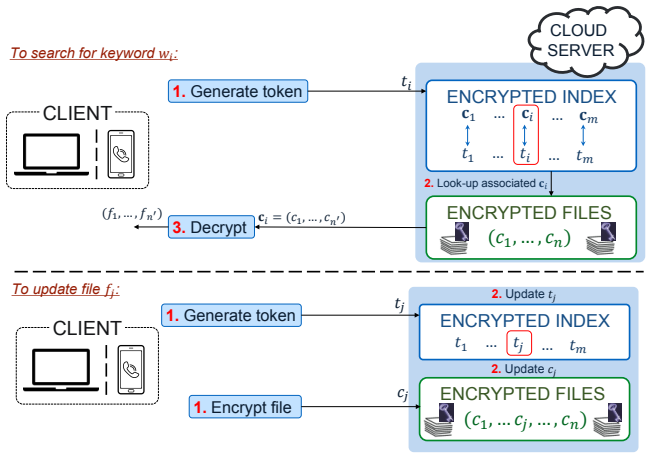


Fig. 1: IM-DSSE framework for file-storage services.

2.2 System and Threat Models

Our system model comprises one server and one client, which can be mobile resource-constrained (e.g., cell phone) as illustrated in Figure 1. Our model can be extended into multiple clients that share the same keys.

In our threat model, the client is trusted and the server is honest-but-curious, meaning that it follows the protocol faithfully but attempts to extract sensitive information during the client's search/update operations. The server can know the encrypted files, the encrypted index and record the transcripts of the protocol. Our objective is to allow the client to perform search and update operations in a secure manner, in which files can be securely retrieved/updated while leaking least information to the server. Specifically, once the server is compromised, the client should only leak the query content and no file contents or specific keywords are ever compromised. Since search and update tokens are deterministic, our IM-DSSE framework leaks *search*, and *file-access patterns* as in all other DSSE schemes. We present the formal security model in §4.

2.3 Notation and Data Structure

Notation. Operators $||$ and $|x|$ denote the concatenation and the bit length of variable x, respectively. $\oplus$ denotes

the Exclusive-OR (XOR) operation. $x \xleftarrow{\$} \mathcal{S}$ denotes variable x is randomly and uniformly selected from set $\mathcal{S}$. $(x_1, \ldots, x_n) \xleftarrow{\$} \mathcal{S}$ denotes $(x_1 \xleftarrow{\$} \mathcal{S}, \ldots, x_n \xleftarrow{\$} \mathcal{S})$. We denote $\{0,1\}^*$ as a set of binary strings of any finite length. $\lfloor x \rfloor$ and $\lceil x \rceil$ denote the floor and the ceiling of x, respectively. Given a matrix $\mathbf{I}$, $\mathbf{I}[i,j]$ denotes the cell indexing at row i and column j. $\mathbf{I}[*,j]$ and $\mathbf{I}[i,*]$ denote accessing column j and row i of matrix $\mathbf{I}$, respectively. $\mathbf{I}[*, a \ldots b]$ denotes accessing columns from a to b of matrix $\mathbf{I}$. $\mathbf{u}[i]$ denotes accessing the i'th component of vector $\mathbf{u}$.

We denote an encryption scheme with Indistinguishability against Chosen Plaintext Attack (IND-CPA) as a triplet $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$: $k \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$, where κ is a security parameter and k is a key; $c \leftarrow \mathcal{E}.\mathsf{Enc}_k(M, u)$ takes as input a secret key k, a counter u and a message M and returns a ciphertext c; $M \leftarrow \mathcal{E}.\mathsf{Dec}_k(c, u)$ takes as input a key k, a counter u and ciphertext c, and returns M if k and u were the key and the counter under which c was produced. The function G is a keyed Pseudo Random Function (PRF), denoted by $\tau \leftarrow G_k(x)$, which takes as input a secret key $k \xleftarrow{\$} \{0,1\}^\kappa$ and a string x, and returns a token/key r. We denote $H : \{0,1\}^{|x|} \rightarrow \{0,1\}$ as a Random Oracle (RO), which takes an input x and returns a bit.

IM-DSSE Data Structures. Our encrypted index is an incidence matrix $\mathbf{I}$, in which $\mathbf{I}[i,j].v \in \{0,1\}$ stores the (encrypted) relationship between keyword indexing at row i and file indexing at column j, and $\mathbf{I}[i,j].\mathsf{st} \in \{0,1\}$ stores a bit indicating the state of $\mathbf{I}[i,j].v$. Particularly, $\mathbf{I}[i,j].\mathsf{st}$ is set to 1 or 0 if $\mathbf{I}[i,j].v$ is last accessed by update or search, respectively. For simplicity, we write $\mathbf{I}[i,j]$ to denote $\mathbf{I}[i,j].v$, and be explicit about the state bit as $\mathbf{I}[i,j].\mathsf{st}$.

The encrypted index $\mathbf{I}$ is augmented by two static hash tables T_w and T_f that associate a keyword and file to a unique row and a column, respectively. Specifically, T_f is a file static hash table whose key-value pair is $(s_{id_j}, \langle y_j, u_j \rangle)$, where $s_{id_j} \leftarrow G_{k_2}(id_j)$ for file with identifier id_j, column index $y_j \in \{1, \ldots, n\}$ is equivalent to the index of s_{id_j} in T_f and u_j is a counter value. We denote access operations by $y_j \leftarrow T_f(s_{id_j})$ and $u_j \leftarrow T_f[y_j].\mathsf{ct}$. T_w is a keyword static hash table whose key-value pair is $\{s_{w_i}, \langle x_i, u_i \rangle\}$, where token $s_{w_i} \leftarrow G_{k_2}(w_i)$ for keyword w_i, row index $x_i \in \{1, \ldots, m\}$ is equivalent to the index of s_{w_i} in T_w and u_i is a counter value. We denote access operations by $x_i \leftarrow T_w(s_{w_i})$ and $u_i \leftarrow T_w[x_i].\mathsf{ct}$. All counter values are incremental and initially set to 1. So, the client state information is in the form of T_w and T_f, that offers (on average) $\mathcal{O}(1)$ access time.

2.4 IM-DSSE$_{\mathsf{main}}$ Algorithms

We present the detailed algorithmic construction for the main scheme (denoted IM-DSSE$_{\mathsf{main}}$) in IM-DSSE framework in Scheme 1, which consists of nine algorithms with high-level ideas as follows.

• *Setup*: The client first executes IM-DSSE$_{\mathsf{main}}$.Gen Algorithm to generate secret keys ($\mathcal{K}$). Based on the generated keys $\mathcal{K}$, the client executes IM-DSSE$_{\mathsf{main}}$.Enc Algorithm to create encrypted data structures to be outsourced to the cloud. In IM-DSSE$_{\mathsf{main}}$.Enc Algorithm, it first extracts m' unique keywords $(w_1, \ldots, w_{m'})$ from n' files $\mathcal{F} = \{f_{id_1}, \ldots, f_{id_{n'}}\}$ with unique IDs $(id_1, \ldots, id_{n'})$ (step 3). It then constructs an (unencrypted) incidence matrix δ (steps 4–9), by setting each cell value $\delta[i,j]$ to $\{0,1\}$, where i, j are the row and column indexes of keyword and file derived from their hash table indexes, respectively (steps 5, 13). Next, it encrypts each cell $\delta[i,j]$ with a unique $\langle$key-counter$\rangle$ pair, where the key (r_i) is

Scheme 1 IM-DSSE$_{\mathsf{main}}$ Scheme

$\mathcal{K} \leftarrow$ IM-DSSE$_{\mathsf{main}}$.Gen(1^κ): Given security parameter κ, generate secret key $\mathcal{K}$

1: $k_1 \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$ and $(k_2, k_3) \xleftarrow{\$} \{0,1\}^\kappa$
2: **return** $\mathcal{K}$, where $\mathcal{K} \leftarrow \{k_1, k_2, k_3\}$

$f \leftarrow$ IM-DSSE$_{\mathsf{main}}$.Dec$_{\mathcal{K}}(c)$: Decrypt encrypted file c with key κ

1: $f \leftarrow \mathcal{E}.\mathsf{Dec}_{k_1}(c', y||u)$ where $u \leftarrow T_f[y].\mathsf{ct}$, $(c', y) \leftarrow c$
2: **return** f

$(\gamma, \mathcal{C}) \leftarrow$ IM-DSSE$_{\mathsf{main}}$.Enc$_{\mathcal{K}}(\delta, \mathcal{F})$: Given index δ and plaintext files $\mathcal{F}$, generate corresponding encrypted index γ and encrypted files $\mathcal{C}$

1: $T_w[i].\mathsf{ct} \leftarrow 1$, $T_f[j].\mathsf{ct} \leftarrow 1$, for $0 \le i \le m, 0 \le j \le n$
2: $\mathbf{I}[*,*].\mathsf{st} \leftarrow 0$ and $\delta[*,*] \leftarrow 0$
3: Extract $(w_1, \ldots, w_{m'})$ from $\mathcal{F} = \{f_{id_1}, \ldots, f_{id_{n'}}\}$
4: **for** $i = 1, \ldots, m'$ **do**
5: $\quad s_{w_i} \leftarrow G_{k_2}(w_i)$, $x_i \leftarrow T_w(s_{w_i})$
6: $\quad$ **for** $j = 1, \ldots, n'$ **do**
7: $\quad\quad$ **if** w_i appears in f_{id_j} **then**
8: $\quad\quad\quad s_{id_j} \leftarrow G_{k_2}(id_j)$ and $y_j \leftarrow T_f(s_{id_j})$
9: $\quad\quad\quad \delta[x_i, y_j] \leftarrow 1$
10: **for** $i = 1, \ldots, m$ **do**
11: $\quad r_i \leftarrow G_{k_3}(i||\overline{u}_i)$, where $\overline{u}_i \leftarrow T_w[i].\mathsf{ct}$
12: $\quad$ **for** $j = 1, \ldots, n$ **do**
13: $\quad\quad \mathbf{I}[i,j] \leftarrow \delta[i,j] \oplus H(r_i||j||u_j)$, where $u_j \leftarrow T_f[j].\mathsf{ct}$
14: **for** $j = 1, \ldots, n'$ **do**
15: $\quad c_j \leftarrow (c'_j, y_j)$, where $c'_j \leftarrow \mathcal{E}.\mathsf{Enc}_{k_1}(f_{id_j}, y_j||u_{y_j})$
16: **return** $(\gamma, \mathcal{C})$, where $\gamma \leftarrow (\mathbf{I}, T_f)$ and $\mathcal{C} \leftarrow \{c_1, \ldots, c_{n'}\}$

$\tau_w \leftarrow$ IM-DSSE$_{\mathsf{main}}$.SearchToken$(\mathcal{K}, w)$: Generate search token τ_w from keyword w and key $\mathcal{K}$

1: $s_w \leftarrow G_{k_2}(w)$, $i \leftarrow T_w(s_w)$
2: $\overline{u} \leftarrow T_w[i].\mathsf{ct}$, $r_i \leftarrow G_{k_3}(i||\overline{u})$
3: **if** $\overline{u} = 1$ **then**
4: $\quad \tau_w \leftarrow (i, r_i)$
5: **else**
6: $\quad \overline{r}_i \leftarrow G_{k_3}(i||\overline{u} - 1)$ and $\tau_w \leftarrow (i, r_i, \overline{r}_i)$
7: $T_w[i].\mathsf{ct} \leftarrow \overline{u} + 1$
8: **return** τ_w

$(\mathcal{I}_w, \mathcal{C}_w) \leftarrow$ IM-DSSE$_{\mathsf{main}}$.Search(τ_w, γ): Given search token τ_w and encrypted index γ, return sets of file identifiers $\mathcal{I}_w$ and encrypted files $\mathcal{C}_w \subseteq \mathcal{C}$ matching with τ_w

1: **for** $j = 1, \ldots, n$ **do**
2: $\quad u_j \leftarrow T_f[j].\mathsf{ct}$
3: $\quad$ **if** $(\tau_w = (i, r_i)$ or $\mathbf{I}[i,j].\mathsf{st} = 1)$ **then**
4: $\quad\quad \mathbf{I}'[i,j] \leftarrow \mathbf{I}[i,j] \oplus H(r_i||j||u_j)$
5: $\quad\quad \mathbf{I}[i,j].\mathsf{st} \leftarrow 0$
6: $\quad$ **else**
7: $\quad\quad \mathbf{I}'[i,j] \leftarrow \mathbf{I}[i,j] \oplus H(\overline{r}_i||j||u_j)$
8: $\quad\quad \mathbf{I}[i,j] \leftarrow \mathbf{I}'[i,j] \oplus H(r_i||j||u_j)$
9: $l \leftarrow 0$
10: **for** each $j \in \{1, \ldots, n\}$ satisfying $\mathbf{I}'[i,j] = 1$ **do**
11: $\quad l \leftarrow l + 1$ and $y_l \leftarrow j$
12: $\mathcal{I}_w \leftarrow \{y_1, \ldots, y_l\}$
13: $\gamma \leftarrow (\mathbf{I}, T_f)$, $\mathcal{C}_w \leftarrow \{(c_{y_1}, y_1), \ldots, (c_{y_l}, y_l)\}$
14: **return** $(\mathcal{I}_w, \mathcal{C}_w)$

Scheme 1 IM-DSSE$_{\text{main}}$ Scheme (continued)

$(\tau_f, c) \leftarrow$ IM-DSSE$_{\text{main}}$.AddToken$(\mathcal{K}, f_{id})$: Given key $\mathcal{K}$ and file f_{id}, generate addition token τ_f and ciphertext c of f_{id}

1: $s_{id} \leftarrow G_{k_2}(id)$, $j \leftarrow T_f(s_{id})$, $T_f[j].\mathsf{ct} \leftarrow T_f[j].\mathsf{ct} + 1$ and $u_j \leftarrow T_f[j].\mathsf{ct}$
2: **for** $i = 1, \ldots, m$ **do**
3: $\quad r_i \leftarrow G_{k_3}(i||\overline{u}_i)$, where $\overline{u}_i \leftarrow T_w[i].\mathsf{ct}$
4: Extract $(w_1, \ldots, w_t)$ from f_{id} and set $\overline{\mathbf{I}}[*, j] \leftarrow 0$
5: **for** $i = 1, \ldots, t$ **do**
6: $\quad s_{w_i} \leftarrow G_{k_2}(w_i)$, $x_i \leftarrow T_w(s_{w_i})$, $\overline{\mathbf{I}}[x_i, j] \leftarrow 1$
7: **for** $i = 1, \ldots, m$ **do**
8: $\quad \mathbf{I}'[i, j] \leftarrow \overline{\mathbf{I}}[i, j] \oplus H(r_i||j||u_j)$
9: $c \leftarrow (c', j)$, where $c' \leftarrow \mathcal{E}.\mathsf{Enc}_{k_1}(f_{id}, j||u_j)$
10: **return** (τ_f, c) where $\tau_f \leftarrow (\mathbf{I}', j)$

$(\gamma', \mathcal{C}') \leftarrow$ IM-DSSE$_{\text{main}}$.Add$(\gamma, \mathcal{C}, c, \tau_f)$: Add addition token τ_f and ciphertext c to encrypted index γ and ciphertext set $\mathcal{C}$, resp.

1: Set $\mathbf{I}[i, j] \leftarrow \mathbf{I}'[i, j]$ and $\mathbf{I}[i, j].\mathsf{st} \leftarrow 1$, for $1 \leq i \leq m$
2: $T_f[j].\mathsf{ct} \leftarrow T_f[j].\mathsf{ct} + 1$
3: **return** $(\gamma', \mathcal{C}')$, where $\gamma' \leftarrow (\mathbf{I}, T_f)$ and $\mathcal{C}' \leftarrow \mathcal{C} \cup \{(c, j)\}$

$\tau'_f \leftarrow$ IM-DSSE$_{\text{main}}$.DeleteToken$(\mathcal{K}, f)$: Given key $\mathcal{K}$ and deleted file f_{id}, generate deletion token τ'_f

1: Execute steps 1–3 of IM-DSSE$_{\text{main}}$.AddToken Algorithm to produce $(j, u_j, \langle r_1, \ldots, r_m \rangle)$ and increase $T_f[j].\mathsf{ct}$ to 1
2: **for** $i = 1, \ldots, m$ **do**
3: $\quad \mathbf{I}'[i, j] \leftarrow H(r_i||j||u_j)$
4: **return** τ'_f, where $\tau'_f \leftarrow (\mathbf{I}', j)$

$(\gamma', \mathcal{C}') \leftarrow$ IM-DSSE$_{\text{main}}$.Delete$(\gamma, \mathcal{C}, \tau'_f)$: Update deletion token τ'_f to encrypted index γ' and delete a file from ciphertext set $\mathcal{C}'$

1: Set $\mathbf{I}[i, j] \leftarrow \mathbf{I}'[i, j]$ and $\mathbf{I}[i, j].\mathsf{st} \leftarrow 1$, for $1 \leq i \leq m$
2: $T_f[j].\mathsf{ct} \leftarrow T_f[j].\mathsf{ct} + 1$
3: **return** $(\gamma', \mathcal{C}')$, where $\gamma' \leftarrow (\mathbf{I}, T_f)$, $\mathcal{C}' \leftarrow \mathcal{C} \setminus \{(c, j)\}$

uniquely derived for each row (i) from $\mathcal{K}$ (step 11), and the counter (u_j) is distinct for each column (j) (step 13). Finally, it encrypts each file in $\mathcal{F}$ resulting in encrypted files $\mathcal{C}$ (steps 14–15). Once $\mathcal{C}$ and the encrypted matrix $(\mathbf{I})$ are constructed, the client sends them to the cloud server along with the file hash table (T_f).

• *Search*: To search for keyword w, the client executes IM-DSSE$_{\text{main}}$.SearchToken Algorithm to generate a search token τ_w to be sent to the server. The token contains the row index (i) of w and row keys $(r_i, \bar{r}_i)$ derived from $\mathcal{K}$, i, and the row counter (u) (steps 3–6). $\bar{r}_i$ and r_i are the old and new keys that are used to decrypt the row being searched for the first and latter times, respectively. Upon receiving τ_w, the server executes IM-DSSE$_{\text{main}}$.Search Algorithm to decrypt the row and retrieve the search result. Specifically, if the cell $\mathbf{I}[i, j]$ is previously updated (indicated via the state bit $\mathbf{I}[i, j].\mathsf{st}$), or being searched for the first time (step 3), it uses the new key (r_i) to decrypt the cell (step 4) and set the state to be 0 (step 5). Otherwise, it uses the old key $(\bar{r}_i)$ to decrypt the cell (step 7) and re-encrypts it with the new key (r_i) (step 8). Finally, the server determines column indexes j such that $\mathbf{I}[i, j] = 1$, and returns the corresponding j-labeled ciphertexts to the client (steps 9–13). The client executes IM-DSSE$_{\text{main}}$.Dec Algorithm on each ciphertext to decrypt the files and obtain the search result.

• *Add a file*: The client executes IM-DSSE$_{\text{main}}$.AddToken Algorithm to generate an addition token (τ_f). It gets the column index (j) of the file to be added from the file hash table (T_f) (step 1), and then, derives fresh row keys from the keyword hash table (T_w) to be used for encrypting the column (steps 2–3). It then extracts unique keywords $(w_1, \ldots w_t)$ from the file, and constructs an (unencrypted) column $\overline{\mathbf{I}}[*, j]$ with values being set to $\{0, 1\}$ (steps 4–6). Finally, it encrypts $\overline{\mathbf{I}}[*, j]$ with row keys (steps 7–8) and the file with $\mathcal{K}$ (step 9). The client sends τ_f containing the ciphertext and the encrypted column to the server. Upon receiving τ_f, the server executes IM-DSSE$_{\text{main}}$.Add Algorithm to update the column j and its state in $\mathbf{I}$ (step 1). It increases the column counter (step 2), and adds the ciphertext to the set of encrypted files (step 3).

• *Delete a file*: It is similar to the file addition protocol, where the client executes IM-DSSE$_{\text{main}}$.DeleteToken Algorithm to generate the deletion token, and the server executes IM-DSSE$_{\text{main}}$.Delete Algorithm to update the column and delete the file from the set of encrypted files.

Keyword update for existing files. Some existing schemes (e.g., [9]) allow file addition/deletion, but do not permit updating keywords in an existing file *directly*. This can be easily achieved in our scheme as follows. Assume the client wants to update file f_{id} by adding (or removing) some keywords, they will prepare a new column $\mathbf{I}'[i, j] \leftarrow b_i$ for $1 \leq i \leq m$, where $b_i = 1$ if w_i is added and $b_i = 0$ otherwise and $j \leftarrow T_f(s_{id})$ with $s_{id} \leftarrow G_{k_2}(id)$ as in IM-DSSE$_{\text{main}}$.AddToken algorithm (steps 4-6). The rest of the algorithm remains the same.

High-level Performance Analysis. For keyword search, IM-DSSE$_{\text{main}}$ incurs n invocations of hash function H and n XOR operations. Although IM-DSSE$_{\text{main}}$ has linear search complexity which is asymptotically less efficient than other DSSE schemes (e.g., [5], [11]), we show in our experiments that, this impact is *insignificant* in practice for personal cloud usage with moderate database size where all optimizations are taken into account. Since IM-DSSE$_{\text{main}}$ is fully parallelizable, the search and update computation times can be reduced to n/p and m/p, respectively, where p is the number of processors. Therefore, cryptographic operations in IM-DSSE$_{\text{main}}$ only contribute a small portion to the overall end-to-end search delay which is dominated by the network communication latency between client and server. Notice that all sub-linear DSSE schemes [2], [5] are less secure and sometimes incur more costly updates than IM-DSSE$_{\text{main}}$. For file update, IM-DSSE$_{\text{main}}$ incurs m invocations of H and m XOR operations along with m bits of transmission.

IM-DSSE$_{\text{main}}$ costs $\Big(2m \cdot n + n \cdot \big(\kappa + |u|\big)\Big)$ bits of storage at the server for encrypted index $\mathbf{I}$ and file hash table T_f. At the client side, IM-DSSE$_{\text{main}}$ requires $(n + m)\big(\kappa + |u|\big) + 3\kappa$ bits for two hash tables T_w, T_f and secret key $\mathcal{K}$.

3 IM-DSSE Extended Schemes

We present extended schemes derived from IM-DSSE$_{\text{main}}$ presented above that IM-DSSE framework also supports.

3.1 IM-DSSE$_{\text{I}}$: Minimized search latency

In IM-DSSE$_{\text{main}}$, we encrypt each cell of $\mathbf{I}$ with a unique key-counter pair, which requires n invocations of H during keyword search. This might not be ideal for some applications that require extremely prompt search delay.

Scheme 2 IM-DSSE$_{\text{I}}$ Scheme

$(\tau_f, c) \leftarrow$ IM-DSSE$_{\text{I}}$.AddToken$(\mathcal{K}, f_{id})$: Given key $\mathcal{K}$ and file f_{id}, generate addition token τ_f and ciphertext c of f_{id}

1: $s_{id} \leftarrow G_{k_2}(id)$, $j \leftarrow T_f(s_{id})$, $l \leftarrow \lfloor \frac{j-1}{b} \rfloor$, $u_l \leftarrow \mathbf{u}[l]$
2: $a \leftarrow (l \cdot b) + 1$, $a' \leftarrow b \cdot (l+1)$
3: Extract $(w_1, \ldots, w_t)$ from f_{id}
4: **for** $i = 1, \ldots, t$ **do**
5: $\quad s_{w_i} \leftarrow G_{k_2}(w_i)$, $x_i \leftarrow T_w(s_{w_i})$
6: Get from server $(\mathbf{I}[*, a \ldots a'])$ and $\mathbf{I}[*, l]$.st
7: **for** $i = 1, \ldots, m$ **do**
8: $\quad \overline{u}_i \leftarrow T_w[i]$.ct
9: $\quad$ **if** $(\overline{u}_i > 1$ and $\mathbf{I}[i, l]$.st $= 0)$ **then**
10: $\quad\quad \overline{u}_i \leftarrow \overline{u}_i - 1$
11: $\quad r_i \leftarrow G_{k_3}(i || \overline{u}_i)^{\dagger}$
12: $\quad \mathbf{I}'[i, a \ldots, a'] \leftarrow \mathcal{E}.\mathsf{Dec}_{r_i}(\mathbf{I}[i, a \ldots a'], l || u_l)$
13: $\mathbf{I}'[i, j] \leftarrow 0$ for $1 \le i \le m$ and $\mathbf{I}'[x_i, j] \leftarrow 1$ for $1 \le i \le t$
14: $\mathbf{u}[l] \leftarrow \mathbf{u}[l] + 1$, $u_l \leftarrow \mathbf{u}[l]$
15: **for** $i = 1, \ldots, m$ **do**
16: $\quad$ **if** $(\overline{u}_i > 1$ and $\mathbf{I}[i, l]$.st $= 0)$ **then**
17: $\quad\quad r_i \leftarrow G_{k_3}(i || \overline{u}_i + 1)$
18: $\quad \overline{\mathbf{I}}[i, a \ldots a'] \leftarrow \mathcal{E}.\mathsf{Enc}_{r_i}(\mathbf{I}'[i, a \ldots a'], l || u_l)$
19: $T_f[j]$.ct $\leftarrow T_f[j]$.ct $+ 1$ and $u'_j \leftarrow T_f[j]$.ct
20: $c \leftarrow (c', j)$ where $c' \leftarrow \mathcal{E}.\mathsf{Enc}_{k_1}(f_{id}, j || u'_j)$
21: **return** (τ_f, c) where $\tau_f \leftarrow (\overline{\mathbf{I}}, j)$

$(\gamma', \mathcal{C}') \leftarrow$ IM-DSSE$_{\text{I}}$.Add$(\gamma, \mathcal{C}, c, \tau_f)$: Add addition token τ_f and ciphertext $\mathcal{C}$ to encrypted index γ and ciphertext set $\mathcal{C}$, resp.

1: $l \leftarrow \lfloor \frac{j-1}{b} \rfloor$, $a \leftarrow (l \cdot b) + 1$, $a' \leftarrow b(l+1)$
2: $\mathbf{I}[i, j'] \leftarrow \overline{\mathbf{I}}[i, j']$, for $1 \le i \le m$ and $a \le j' \le a'$
3: $\mathbf{u}[l] \leftarrow \mathbf{u}[l] + 1$ and $\mathbf{I}[*, l]$.st $\leftarrow 1$
4: **return** $(\gamma', \mathcal{C}')$, where $\gamma' \leftarrow (\mathbf{I}, T_f)$ and $\mathcal{C}' \leftarrow \mathcal{C} \cup \{c\}$

† G should generate a suitable key for $\mathcal{E}$ (e.g., 128-bit key for AES-CTR)

Hence, we introduce an extended scheme called IM-DSSE$_{\text{I}}$, which aims at achieving a very low search latency with the cost of increasing update delay. Specifically, instead of encrypting the index bit-by-bit as in IM-DSSE$_{\text{main}}$ scheme, IM-DSSE$_{\text{I}}$ leverages b-bit block cipher encryption to encrypt b successive cells with the same key-counter pair. This is achieved by interpreting columns of $\mathbf{I}$ as $D = \lceil \frac{n}{b} \rceil$ blocks, each being IND-CPA encrypted using AES-CTR mode with block cipher size b. The counter will be stored via a block counter array (denoted as $\mathbf{u}$) instead of $T_f[\cdot].u$ as in the main scheme. The update state is maintained for each block rather than each cell of $\mathbf{I}[i, j]$. Hence, $\mathbf{I}$ is decomposed into two matrices with different sizes: $\mathbf{I}.v \in \{0,1\}^{m \times n}$ and $\mathbf{I}.\mathsf{st} \in \{0,1\}^{m \times D}$.

IM-DSSE$_{\text{I}}$ requires some algorithmic modifications from the main scheme. Scheme 2 presents IM-DSSE$_{\text{I}}$.AddToken Algorithm and IM-DSSE$_{\text{I}}$.Add Algorithm for file addition procedure in IM-DSSE$_{\text{I}}$ (modifications for file deletion follow the same principle). Specifically, we substitute encryption and decryption using random oracle $H(r_i||j||u_j)$ with block cipher encryption $\mathcal{E}.\mathsf{Enc}_{r_i}(\cdot, l||u'_l)$ and $\mathcal{E}.\mathsf{Dec}_{r_i}(\cdot, l||u'_l)$, respectively, where u_l is a block counter (see steps 12, 20). Since $\mathbf{I}$ is encrypted by blocks, to update a column during the file update, the client needs to retrieve a whole block and its state from the server first (step 6). The client then decrypts the block (steps 7–12), updates a column within it (step 13), re-encrypts the entire block (steps 15–18), and finally sends the encrypted block to the server for replacement. So, the reduction of search cost increases the cost of communication overhead for the update as a trade-off.

Since the modifications for IM-DSSE$_{\text{I}}$.Gen, IM-DSSE$_{\text{I}}$.Enc, IM-DSSE$_{\text{I}}$.SearchToken and IM-DSSE$_{\text{I}}$.Search algorithms are straightforward, where only the underlying encryption is changed from random oracle to block cipher (e.g., AES-CTR) as exemplified in IM-DSSE$_{\text{I}}$.AddToken Algorithm, we will not present those algorithms in detail due to space limitation.

High-level Performance Analysis. For keyword search, IM-DSSE$_{\text{I}}$ requires n/b invocations of $\mathcal{E}$, which is theoretically b times faster than the main scheme. Given the CTR mode, the search time can be reduced to $n/(b \cdot p)$, where p is the number of processors. For file update, IM-DSSE$_{\text{I}}$ requires transmission of $(2b+1) \cdot m$ bits along with decryption and encryption operations at the client side, compared with m non-interactive transmission and encryption-only in the main scheme. Thus, the keyword search speed in IM-DSSE$_{\text{I}}$ is increased by a factor of b (e.g., $b = 128$) with the cost of transmitting $(2b+1) \cdot m$ bits in the file update.

IM-DSSE$_{\text{I}}$ reduces the server storage to $\left(\frac{n \cdot |u| + m \cdot n \cdot (b+1)}{b}\right)$ bits. The client storage remains the same as in IM-DSSE$_{\text{main}}$.

3.2 IM-DSSE$_{\text{II}}$: Achieving cloud SaaS infrastructure with backward privacy

All DSSE schemes introduced so far require the server to perform some computation (i.e., encryption/decryption) during keyword search, which might not be fully compatible with typical storage-only clouds (e.g., Dropbox, Google Drive, Amazon S3). Hence, we propose an extended scheme derived from IM-DSSE$_{\text{main}}$ called IM-DSSE$_{\text{II}}$, where all computations are performed at the client side while the server does nothing rather than serving as a storage unit. This simple trick makes IM-DSSE$_{\text{II}}$ not only compatible with with storage-only clouds, but also more importantly, achieve the backward-privacy property. This is because the server now cannot decrypt any part of encrypted index to keep track of historical update operations. Moreover, IM-DSSE$_{\text{II}}$ also reduces the storage at both client and server sides by eliminating the state matrix and keyword counters that are needed in IM-DSSE$_{\text{main}}$ and IM-DSSE$_{\text{I}}$ to perform correct decryption and achieve forward-privacy during search and update, respectively.

We present the keyword search procedure of IM-DSSE$_{\text{II}}$ in Scheme 3, which combines SearchToken and Search algorithms in DSSE. To search for keyword w, the client sends to the server the w's row index (i) and receives the

Scheme 3 IM-DSSE$_{\text{II}}$ Scheme

$(\mathcal{I}_w, \mathcal{C}_w) \leftarrow \mathsf{Search}(\mathcal{K}, w)$: Given keyword w and key $\mathcal{K}$, return sets of file identifiers $\mathcal{I}_w$ and encrypted files $\mathcal{C}_w \subseteq \mathcal{C}$ matching with w

1: $s_{w_i} \leftarrow G_{k_2}(w)$, $i \leftarrow T_w(s_{w_i})$, $r_i \leftarrow G_{k_3}(i)$, $l \leftarrow 0$
2: Fetch the i-th row data $\mathbf{I}[i, *]$ from server
3: **for** $j = 1, \ldots, n$ **do**
4: $\quad u_j \leftarrow T_f[j]$.ct
5: $\quad \mathbf{I}'[i, j] \leftarrow \mathbf{I}[i, j] \oplus H(r_i||j||u_j)$
6: **for** each $j \in \{1, \ldots, n\}$ satisfying $\mathbf{I}'[i, j] = 1$ **do**
7: $\quad l \leftarrow l + 1$ and $y_l \leftarrow j$
8: $\mathcal{I}_w \leftarrow \{y_1, \ldots, y_l\}$
9: Send $\mathcal{I}_w$ to server and receive $\mathcal{C}_w = \{(c_{y_1}, y_1), \ldots, (c_{y_l}, y_l)\}$
10: $f_i \leftarrow \mathsf{Dec}_{\mathcal{K}}(c_{y_i})$ for $1 \le i \le l$
11: **return** $(\mathcal{I}_w, \mathcal{F}_w)$, where $\mathcal{F}_w \leftarrow \{f_1, \ldots, f_l\}$

corresponding row $\mathbf{I}[i,*]$ (step 2). The client decrypts $\mathbf{I}[i,*]$, extracts column indexes j such that $\mathbf{I}[i,j] = 1$. Since the client computes everything, it is not required to derive new row keys for forward-privacy and therefore, state matrix $\mathbf{I}[*,*]$.st as well as file hash table T_f at the server and keyword counters T_w.ct at the client are not needed in IM-DSSE$_{\text{II}}$ (see steps 3–5 for example). The client then fetches and decrypts encrypted files indexed at j to obtain the search result (step 9).

IM-DSSE$_{\text{II}}$.Gen is identical to IM-DSSE$_{\text{main}}$.Gen Algorithm. IM-DSSE$_{\text{II}}$.Enc, IM-DSSE$_{\text{II}}$.Add, IM-DSSE$_{\text{II}}$.AddToken, IM-DSSE$_{\text{II}}$.Delete, IM-DSSE$_{\text{II}}$.DeleteToken can be easily derived from their version in the main scheme (IM-DSSE$_{\text{main}}$) by *(1)* substituting row key generation $r_i \leftarrow G_{k_3}(i, \overline{u}_i)$ with $r_i \leftarrow G_{k_3}(i)$, *(2)* omitting all keyword counters $\overline{u}_i$, block states $\mathbf{I}[*,*]$.st, *(3)* and removing T_f from the server storage. Due to space limitation and repetition, we will not present them in detail.

High-level Performance Analysis. The computation cost of IM-DSSE$_{\text{II}}$ is identical to IM-DSSE$_{\text{main}}$ (i.e., n and m invocations of H for search and update resp.). IM-DSSE$_{\text{II}}$ requires two-round communication with n bits being transmitted during keyword search.

IM-DSSE$_{\text{II}}$ reduces the client and server storage costs to $n(\kappa + |u|) + m \cdot \kappa + 3\kappa$ and $m \cdot n$ bits, respectively.

3.3 IM-DSSE$_{\text{I+II}}$: Low search latency, backward-privacy and compatibility with cloud SaaS infrastructure

Our IM-DSSE framework also supports IM-DSSE$_{\text{I+II}}$, an extended DSSE scheme which is the combination of IM-DSSE$_{\text{I}}$ and IM-DSSE$_{\text{II}}$ schemes. In IM-DSSE$_{\text{I+II}}$, the incidence matrix $\mathbf{I}$ is encrypted with b-bit block cipher encryption, and the decryption is performed by the client during search. Since IM-DSSE$_{\text{I+II}}$ inherits all properties of IM-DSSE$_{\text{I}}$ and IM-DSSE$_{\text{II}}$ schemes, IM-DSSE$_{\text{I+II}}$ is highly desirable for cloud SaaS infrastructure that requires a very low search latency and backward-privacy with the costs of more delayed update and an extra communication round during search.

4 Security Analysis

In this section, we analyze the security and update privacy of all the DSSE schemes provided in our IM-DSSE framework. Most known efficient SSE schemes (e.g., [2], [5], [9]) reveal the *search* and *file-access patterns* defined as follows.

- Given search query w at time t, *the search pattern* $\mathcal{P}(\delta, \text{Query}, t)$ 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 at time t, *the file-access pattern* $\Delta(\delta, \mathcal{F}, w, t)$ is identifiers $\mathcal{I}_w$ of files $\mathcal{F}$ containing w.

We consider leakage functions in the line of [8] that captures dynamic file addition/deletion in its security model, but we leak much less information compared to [8].

Definition 1 (Leakage Function)**.** *We define leakage functions* $(\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3)$ *as follows:*

1) $(m, n, \mathcal{I}, \langle |f_{id_1}|, \ldots, |f_{id_n}| \rangle) \leftarrow \mathcal{L}_1(\delta, \mathcal{F})$*: Given the index* δ *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) $(\mathcal{P}(\delta, Query, t), \Delta(\delta, \mathcal{F}, w, t)) \leftarrow \mathcal{L}_2(\delta, \mathcal{F}, w, t)$*: Given the index* δ*, the set of files* $\mathcal{F}$ *and a keyword* w *for the search operation at time* t*, it outputs the search pattern* $\mathcal{P}$ *and file-access pattern* Δ*.*
3) $|f_{id}| \leftarrow \mathcal{L}_3(\delta, \mathcal{F}, id, t, \text{op})$*: Given the index* δ*, the set of files* $\mathcal{F}$*, a file identifier* id*, and the update type* $\text{op} \in \{\langle \text{Add}, |f_{id}| \rangle, \text{Delete}\}$ *at time* t*, it outputs the size of updated file* f_{id} *(which also implies the size of its corresponding ciphertext* $|c_{id}|$*).*

Definition 2 (IND-CKA2 Security [1], [3])**.** *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* $\mathcal{K} \leftarrow \text{Gen}(1^\kappa)$*.* $\mathcal{A}$ *produces* $(\delta, \mathcal{F})$ *and receives* $(\gamma, \mathcal{C}) \leftarrow \text{Enc}_{\mathcal{K}}(\delta, \mathcal{F})$ *from the challenger.* $\mathcal{A}$ *makes polynomially bounded number of adaptive queries* $\text{Query} \in (w, f_{id}, f_{id'})$ *to the challenger. If* $\text{Query} = w$ *is a keyword search query then* $\mathcal{A}$ *receives a search token* $\tau_w \leftarrow \text{SearchToken}(\mathcal{K}, w)$ *from the challenger. If* $\text{Query} = f_{id}$ *is a file addition query then* $\mathcal{A}$ *receives an addition token* $(\tau_f, c) \leftarrow \text{AddToken}(\mathcal{K}, f_{id})$ *from the challenger. If* $\text{Query} = f_{id'}$ *is a file deletion query then* $\mathcal{A}$ *receives a deletion token* $\tau'_f \leftarrow \text{DeleteToken}(\mathcal{K}, f_{id'})$ *from the challenger. Eventually,* $\mathcal{A}$ *returns a bit* b *that is the output of the experiment.*

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

A DSSE *is said to be* $(\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3)$*-secure against adaptive chosen-keyword attacks (CKA2-security) if for all* PPT *adversaries* $\mathcal{A}$*, there exists a* PPT *simulator* $\mathcal{S}$ *such that*

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

Remark 1. *In Definition 2, we adopt the* dynamic CKA2-security *notion in [8] that captures the file addition and deletion by simulating corresponding tokens* τ_f *and* τ'_f*, resp.*

The security of IM-DSSE can be stated as follows.

Theorem 1. *If* $\mathcal{E}$.Enc *is IND-CPA secure,* G *is PRF and* H *is a RO then IM-DSSE is* $(\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3)$*-secure in ROM by Definition 2 (CKA-2 security with update capacity).*

Proof. We prove the IND-CKA2 for IM-DSSE$_{\text{main}}$ proposed in §2. The proof for extended schemes in §3 can be easily derived from this proof (see Remark 2 below for argument) and therefore, we will not repeat it.

To begin with, we construct a simulator $\mathcal{S}$ that interacts with an adversary $\mathcal{A}$ in an execution of an $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(\kappa)$ experiment as described in Definition 2. In this experiment, $\mathcal{S}$ maintains lists $\mathcal{LR}$, $\mathcal{LK}$ and $\mathcal{LH}$ to keep track of query results, states and history information, respectively. Initially, all lists are set to empty. $\mathcal{LR}$ is a list of key-value pairs and is used to keep track of $RO(\cdot)$ queries. We denote

$\mathsf{value} \leftarrow \mathcal{LR}(\mathsf{key})$ and $\perp \leftarrow \mathcal{LR}(\mathsf{key})$ if key does not exist in $\mathcal{LR}$. $\mathcal{LK}$ is to keep track of random values generated during the simulation and it is similar to $\mathcal{LR}$. $\mathcal{LH}$ is to keep track of search and update queries, $\mathcal{S}$'s replies to those queries and their leakage output from $(\mathcal{L}_1, \mathcal{L}_2)$. $\mathcal{S}$ executes the simulation as follows.

I. Handle $RO(\cdot)$ Queries: $b \leftarrow RO(x)$ takes an input x and returns a bit b as output. Given x, if $\perp = \mathcal{LR}(x)$ set $b \xleftarrow{\$} \{0,1\}$, insert (x, b) into $\mathcal{LR}$ and return b as the output. Else, return $b \leftarrow \mathcal{LR}(x)$ as the output.

II. Simulate $(\gamma, \mathcal{C})$: Given $(m, n, \langle id_1, \ldots, id_{n'}\rangle, \langle |c_1|, \ldots, |c_{n'}|\rangle) \leftarrow \mathcal{L}_1(\delta, \mathcal{F})$, $\mathcal{S}$ simulates $(\gamma, \mathcal{C})$ as follows.

1) $(s_{id_j}, k) \xleftarrow{\$} \{0,1\}^\kappa$, $y_j \leftarrow T_f(s_{id_j})$, insert (id_j, s_{id_j}, y_j) into $\mathcal{LH}$ and $c_{y_j} \leftarrow \mathcal{E}.\mathsf{Enc}_k(\{0\}^{|c_{id_j}|})$ for $1 \le j \le n'$.
2) For $j = 1, \ldots, n$ and $i = 1, \ldots, m$
   a) $T_w[i].\mathsf{ct} \leftarrow 1$ and $T_f[j].\mathsf{ct} \leftarrow 1$.
   b) $z_{i,j} \xleftarrow{\$} \{0,1\}^\kappa$, $\mathbf{I}[i,j] \leftarrow RO(z_{i,j})$ and $\mathbf{I}[i,j].\mathsf{st} \leftarrow 0$.
3) Output $(\gamma, \mathcal{C})$, where $\gamma \leftarrow (\mathbf{I}, T_f)$ and $\mathcal{C} \leftarrow \{\langle c_i, y_i\rangle\}_{i=1}^{n'}$

Correctness and Indistinguishability of the Simulation: $\mathcal{C}$ has the correct size and distribution, since $\mathcal{L}_1$ leaks $\langle |c_{id_1}|, \ldots, |c_{id_{n'}}|\rangle$ and $\mathcal{E}.\mathsf{Enc}(\cdot)$ is a IND-CPA secure scheme, respectively. $\mathbf{I}$ and T_f have the correct size since $\mathcal{L}_1$ leaks (m, n). Each $\mathbf{I}[i,j]$ for $1 \le j \le n$ and $1 \le i \le m$ has random uniform distribution, since $RO(\cdot)$ is invoked with random value $z_{i,j}$. T_f has the correct distribution, since each s_{id_j} has random uniform distribution, for $1 \le j \le n'$. Hence, $\mathcal{A}$ does not abort due to $\mathcal{A}$'s simulation of $(\gamma, \mathcal{C})$. The probability that $\mathcal{A}$ queries $RO(\cdot)$ on any $z_{i,j}$ before $\mathcal{S}$ provides $\mathbf{I}$ to $\mathcal{A}$ is negligible (i.e., $\frac{1}{2^\kappa}$). Hence, $\mathcal{S}$ also does not abort.

III. Simulate τ_w: Simulator $\mathcal{S}$ receives a search query for an arbitrary keyword w on time t. $\mathcal{S}$ is given $\big(\mathcal{P}(\delta, \mathsf{Query}, t), \Delta(\delta, \mathcal{F}, w, t)\big) \leftarrow \mathcal{L}_2(\delta, \mathcal{F}, w, t)$. $\mathcal{S}$ adds these to $\mathcal{LH}$. $\mathcal{S}$ then simulates τ_w and updates lists $(\mathcal{LR}, \mathcal{LK})$ as follows.

1) If w is in $\mathcal{LH}$, then fetch s_w. Else, $s_w \xleftarrow{\$} \{0,1\}^\kappa$, $i \leftarrow T_w(s_{w_i})$, $\overline{u}_i \leftarrow T_w[i].\mathsf{ct}$, insert $(w, \mathcal{L}_1(\delta, \mathcal{F}), s_w)$ into $\mathcal{LH}$.
2) If $\perp = \mathcal{LK}(i||\overline{u}_i)$, then $r_i \xleftarrow{\$} \{0,1\}^\kappa$ and insert $(r_i, i, \overline{u}_i)$ into $\mathcal{LK}$. Else, $r_i \leftarrow \mathcal{LK}(i||\overline{u}_i)$.
3) If $\overline{u}_i > 1$, then $\overline{r}_i \leftarrow \mathcal{LK}(i||\overline{u}_i - 1)$, $\tau_w \leftarrow (i, r_i, \overline{r}_i)$. Else, $\tau_w \leftarrow (i, r_i)$.
4) $T_w[i].\mathsf{ct} \leftarrow \overline{u}_i + 1$.
5) Given $\mathcal{L}_2(\delta, \mathcal{F}, w, t)$, $\mathcal{S}$ knows identifiers $\mathcal{I}_w = \{y_1, \ldots, y_l\}$. Set $\mathbf{I}'[i,y] \leftarrow 1$ for each $y \in \mathcal{I}_w$ and the rest of the elements as $\mathbf{I}'[i,j] \leftarrow 0$ for each $j \in \{\{1, \ldots, n\} \setminus \mathcal{I}_w\}$.
6) If $((\tau_w = (i, r_i) \vee \mathbf{I}[i,j].\mathsf{st}) = 1)$, then $\mathbf{V}[i,j] \leftarrow \mathbf{I}[i,j]' \oplus \mathbf{I}[i,j]$ and insert tuple $(r_i||j||u_j, \mathbf{V}[i,j])$ into $\mathcal{LR}$, where $u_j \leftarrow T_f[j].\mathsf{ct}$ for $1 \le j \le n$.
7) $\mathbf{I}[i,j].\mathsf{st} \leftarrow 0$ for $1 \le j \le n$.
8) $\mathbf{I}[i,j] \leftarrow \mathbf{I}'[i,j] \oplus RO(r_i||j||u_j)$, where $u_j \leftarrow T_f[j].\mathsf{ct}$ for $1 \le j \le n$.
9) Output τ_w and insert (w, τ_w) into $\mathcal{LH}$.

Correctness and Indistinguishability of the Simulation: Given any $\Delta(\delta, \mathcal{F}, w, t)$, $\mathcal{S}$ simulates the output of $RO(\cdot)$ such that τ_w always produces the correct search result for $\mathcal{I}_w \leftarrow \mathsf{Search}\tau_w, \gamma$. $\mathcal{S}$ needs to simulate the output of $RO(\cdot)$ for two conditions (as in *III-Step 6*): (i) The first search of w (i.e., $\tau_w \stackrel{?}{=} (i, r_i)$), since $\mathcal{S}$ did not know δ during the simulation of $(\gamma, \mathcal{C})$. (ii) If any file f_{id} containing w has been updated after the last search on w (i.e., $\mathbf{I}[i,j].\mathsf{st} \stackrel{?}{=} 1$), since $\mathcal{S}$ does not know the update content. $\mathcal{S}$ sets the output of $RO(\cdot)$ for those cases by inserting tuple $(r_i||j||u_j, \mathbf{V}[i,j])$ into $\mathcal{LR}$ (as in *III-Step 6*). In other cases, $\mathcal{S}$ just invokes $RO(\cdot)$ with $(r_i||j||u_j)$, which consistently returns the previously inserted bit from $\mathcal{LR}$ (as in *III-Step 8*).

During the first search on w, each $RO(\cdot)$ outputs $\mathbf{V}[i,j] = RO(r_i||j|u_j)$ that has the correct distribution, since $\mathbf{I}[i,*]$ of γ has random uniform distribution (see *II-Correctness and Indistinguishability* argument). Let $\mathcal{J} = \{j_1, \ldots, j_l\}$ be the set of indexes of files containing w, which are updated after the last search on w. If w is searched again after being updated, then each $RO(\cdot)$'s output $\mathbf{V}[i,j] = RO(r_i||j|u_j)$ has the correct distribution, since $\tau_f \leftarrow (\mathbf{I}', j)$ for indexes $j \in \mathcal{J}$ has random uniform distribution (see *IV-Correctness and Indistinguishability* argument). Given that $\mathcal{S}$'s τ_w always produces correct $\mathcal{I}_w$ for given $\Delta(\delta, \mathcal{F}, w, t)$, and relevant values and $RO(\cdot)$ outputs have the correct distribution, $\mathcal{A}$ does not abort during the simulation due to $\mathcal{S}$'s search token. The probability that $\mathcal{A}$ queries $RO(\cdot)$ on any $(r_i||j|u_j)$ before querying $\mathcal{S}$ on τ_w is negligible (i.e., $\frac{1}{2^\kappa}$) and, therefore, $\mathcal{S}$ does not abort due to $\mathcal{A}$'s search query.

IV. Simulate (τ_f, τ_f'): $\mathcal{S}$ receives an update request $\mathsf{op} \in \{\langle \mathsf{Add}, |c|\rangle, \mathsf{Delete}\}$ for an arbitrary file having id at time t. Given $|c_{id}| \leftarrow \mathcal{L}_3(\delta, \mathcal{F}, id, t, \mathsf{op})$, $\mathcal{S}$ simulates update tokens (τ_f, τ_f') as follows.

1) If id is in $\mathcal{LH}$, then fetch (id, s_{id}, j). Else set $s_{id} \xleftarrow{\$} \{0,1\}^\kappa$, $j \leftarrow T_f(s_{id})$ and insert (id, s_{id}, j) into $\mathcal{LH}$.
2) $T_f[j].\mathsf{ct} \leftarrow T_f[j].\mathsf{ct} + 1$, $u_j \leftarrow T_f[j].\mathsf{ct}$.
3) If $\perp = \mathcal{LK}(i||\overline{u}_i)$, then $r_i \xleftarrow{\$} \{0,1\}^\kappa$ and insert $(r_i, i, \overline{u}_i)$ into $\mathcal{LK}$, where $\overline{u}_i \leftarrow T_w[i].\mathsf{ct}$ for $1 \le i \le m$.
4) $\mathbf{I}'[i,j] \leftarrow RO(z_i)$, where $z_i \xleftarrow{\$} \{0,1\}^{2\kappa}$ for $1 \le i \le m$.
5) Set $\mathbf{I}[i,j] \leftarrow \mathbf{I}'[i,j]$ and $\mathbf{I}[i,j].\mathsf{st} \leftarrow 1$ for $1 \le i \le m$.
6) If $\mathsf{op} = \langle \mathsf{Add}, |c|\rangle$, then simulate $c_j \leftarrow \mathcal{E}.\mathsf{Enc}_k(\{0\}^{|c|})$ add c_j into $\mathcal{C}$, set $\tau_f \leftarrow (\mathbf{I}', j)$ and output τ_f. Else, set $\tau_f' \leftarrow (\mathbf{I}', j)$, remove c_j from $\mathcal{C}$ and output τ_f'.

Correctness and Indistinguishability of the Simulation: Given access pattern (τ_f, τ_f') for a file f_{id}, $\mathcal{A}$ checks the correctness of update by searching all keywords $\mathcal{W} = \{w_{i_1}, \ldots, w_{i_l}\}$ in f_{id}. Since $\mathcal{S}$ is given access pattern $\Delta(\delta, \mathcal{F}, w, t)$ for a search query (which captures the last update before the search), the search operation always produces a correct result after an update (see *III-Correctness and Indistinguishability* argument). Hence, $\mathcal{S}$'s update tokens are correct and consistent.

It remains to show that (τ_f, τ_f') have the correct probability distribution. In the real algorithm, the counter u_j is increased for each update as simulated in *IV-Step 2*. If f_{id} is updated after the keyword w at row i is searched, a new r_i is generated for w as simulated in *IV-Step 3* (r_i remains the same for consecutive updates but u_j increases). Hence, the real algorithm invokes $H(.)$ with a different $(r_i||j||u_j)$ for $1 \le i \le m$. $\mathcal{S}$ simulates this step by invoking $RO(\cdot)$ with z_i and $\mathbf{I}'[i,j] \leftarrow RO(z_i)$, for $1 \le i \le m$. (τ_f, τ_f') have random uniform distribution since $\mathbf{I}'$ has random uniform distribution and update operations are correct and consistent as shown above. c_j also has the correct distribution since $\mathcal{E}.\mathsf{Enc}(\cdot)$ is an IND-CPA encryption. Hence, $\mathcal{A}$ does not abort during the simulation due to $\mathcal{S}$'s update tokens. The probability that $\mathcal{A}$ queries $RO(\cdot)$ on any z_i prior querying $\mathcal{S}$

on (τ_f, τ'_f) is negligible (i.e., $\frac{1}{2^{2\cdot\kappa}}$) and, therefore, $\mathcal{S}$ does not abort due to $\mathcal{A}$'s update query.

V. Final Indistinguishability Argument: (s_{w_i}, s_{id_j}, r_i) for $1 \leq i \leq m$ and $1 \leq j \leq n$ are indistinguishable from real tokens and keys since they are generated by PRFs that are indistinguishable from random functions. $\mathcal{E}.\mathsf{Enc}(\cdot)$ is a IND-CPA scheme, the answers returned by $\mathcal{S}$ to $\mathcal{A}$ for $RO(\cdot)$ queries are consistent and appropriately distributed, and all query replies of $\mathcal{S}$ to $\mathcal{A}$ during the simulation are correct and indistinguishable as discussed in *I-IV Correctness and Indistinguishability* arguments. Hence, for all PPT adversaries, the outputs of $\mathbf{Real}_{\mathcal{A}}(\kappa)$ and $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(\kappa)$ experiment are:

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

where $\mathsf{neg}(\cdot)$ is a negligible function. □

We argue the security of extended schemes as follows.

Remark 2. *IM-DSSE$_{\mathsf{I}}$, IM-DSSE$_{\mathsf{II}}$ and IM-DSSE$_{\mathsf{I+II}}$ are secure by Definition 2, and their proofs can be easily derived from the security analysis of IM-DSSE$_{main}$ presented above.*

Specifically, IM-DSSE$_{\mathsf{I}}$ only differs from IM-DSSE$_{main}$ in terms of b-bit encryption, compared with 1-bit encryption. This modification does not impact the IND-CKA2 security of IM-DSSE$_{\mathsf{I}}$ over IM-DSSE$_{main}$. Given that we use ROs and an IND-CPA encryption scheme (e.g., AES with CTR mode), the security of IM-DSSE$_{\mathsf{I}}$ is not affected in our model, and in particular, there is no additional leakage. The price that is paid for this performance improvement is that the scheme becomes interactive. Since the block data exchanged between client and server are encrypted with an IND-CPA encryption scheme, there is no additional leakage due to this operation.

IM-DSSE$_{\mathsf{II}}$ only differs from IM-DSSE$_{main}$ in terms of where the decryption during keyword search takes place. Performing decryption at the client (instead of at the server) does not impact the IND-CKA2 security of IM-DSSE$_{\mathsf{II}}$ over IM-DSSE$_{main}$. Given that we, respectively, use ROs and PRF for H and G as in IM-DSSE$_{main}$, the security of IM-DSSE$_{\mathsf{II}}$ remains the same.

IM-DSSE$_{\mathsf{I+II}}$ is merely the combination of IM-DSSE$_{\mathsf{I}}$ and IM-DSSE$_{\mathsf{II}}$, where block cipher encryption and client decryption during search are both implemented. As analyzed, each of these strategies does not impact the security and therefore, IM-DSSE$_{\mathsf{I+II}}$ still preserve the IND-CKA2 security.

The leakage definition and formal security model imply various levels of privacy for different DSSE schemes. We summarize some important privacy notions based on the various leakage characteristics discussed in [2] as follows.

- *Size pattern*: The number of actual keyword-file pairs.
- *Forward privacy*: A search on a keyword w does not leak the IDs of files being updated in the future and having w.
- *Backward privacy*: A search on a keyword w does not leak all historical update operations (e.g., addition /deletion) on the identifiers of files having this keyword.

Since keyword-file relationships are represented by an encrypted incidence matrix, IM-DSSE framework hides the *size pattern* (i.e., number of '1' in $\mathbf{I}$), so that it is size-oblivious.

Corollary 1. *IM-DSSE framework offers forward-privacy.*

Proof. In IM-DSSE framework, the update involves reconstructing a new column/block of encrypted index $\mathbf{I}$. The column/block is always encrypted with row keys that have never been revealed to the server (Step 2–4 in Simulate (τ_f,τ'_f))). This is achieved in IM-DSSE$_{\mathsf{main}}$ and IM-DSSE$_{\mathsf{I}}$ schemes by increasing the row counter after each keyword search operation (e.g., Step 4 in Simulate (τ_w)) so that fresh row keys will be used for subsequent update operations. In IM-DSSE$_{\mathsf{II}}$ scheme, since all cryptographic operations are performed at the client side where no keys are revealed to the server, it is unable for the server to infer any information in the update, given that the encryption scheme is IND-CPA secure. These properties enable our IM-DSSE framework to achieve forward privacy. □

Corollary 2. *IM-DSSE$_{\mathsf{II}}$ and IM-DSSE$_{\mathsf{I+II}}$ achieve backward-privacy.*

Proof. In most DSSE schemes, the client sends a key that allows the server to decrypt a small part of the encrypted index during keyword search. The server can use this key to backtrack historical update operations on this part and therefore, compromise the backward-privacy. In IM-DSSE$_{\mathsf{II}}$ and IM-DSSE$_{\mathsf{I+II}}$ schemes, instead of sending the key to the server, the client requests this part and decrypts it locally. This prevents the server from learning information about historical update operations on the encrypted index and therefore, allows both schemes to achieve backward-privacy. □

5 Performance Analysis and Evaluation

We evaluate the performance of our IM-DSSE framework in real-life networking and system settings. We provide a detailed cost breakdown analysis to fully assess the criteria that constitute the performance overhead of our constructions. Given that such analysis is generally missing in the literature, this is the main focus of our performance evaluation. Finally, we give a brief asymptotic comparison of our framework with several DSSE schemes in the literature.

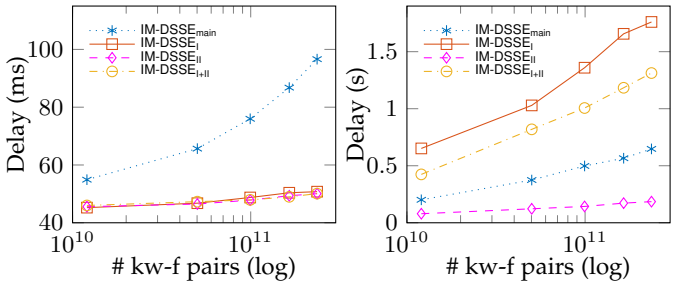
Implementation Details. We implemented our framework using C/C++. For cryptographic primitives, we used libtomcrypt library [33]. We modified low level routines to call AES hardware acceleration instructions (via Intel AES-NI library [34]) if they are supported by the underlying hardware platform. We used AES-128 Cipher-based Message Authentication Code (CMAC) for hash function. Our random oracles were all implemented via 128-bit AES CMAC. For hash tables, we employed Google's C++ sparse hash map library [35] with the hash function being implemented by the CMAC-based random oracles truncated to 80 bits. We implemented the IND-CPA encryption $\mathcal{E}$ using AES with CTR mode. For network communication, we used ZeroMQ library [36].

IM-DSSE framework contains the full implementation of all schemes presented in this article including IM-DSSE$_{\mathsf{main}}$, IM-DSSE$_{\mathsf{I}}$, IM-DSSE$_{\mathsf{II}}$ and IM-DSSE$_{\mathsf{I+II}}$, which can be freely accessed via our following Github repository [37].

https://github.com/thanghoang/IM-DSSE/

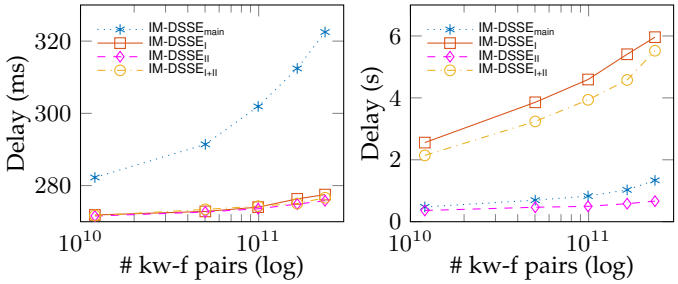
Our implementation supports the encrypted index stored on either memory or local disk. Therefore, our

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2897096, IEEE Transactions on Services Computing



(a) Keyword search (b) File update

Fig. 2: The latency of our schemes with fast network.



(a) Keyword search (b) File update

Fig. 3: The latency of our schemes with moderate network.

schemes can be directly deployed in either storage-as-a-service (e.g., Amazon S3) or infrastructure-as-a-service clouds (e.g., Amazon EC2). For this experimental evaluation, we selected block cipher size $b = 128$ for IM-DSSE$_{I}$ and IM-DSSE$_{I+II}$ schemes.

Dataset. We used subsets of the Enron email dataset [38], ranging from 50,000 to 250,000 files with 240,000–940,000 unique keywords to evaluate the performance of our schemes with different encrypted index sizes. These selected sizes surpass the experiments in [3] by three orders of magnitude and are comparable to the experiments in [2].

Hardware. We conducted the experiment with two settings:

(i) We used HP Z230 Desktop as the client and built the server using Amazon EC2 with `m4.4xlarge` instance type. The desktop was equipped with Intel Xeon CPU E3-1231v3 @ 3.40GHz, 16 GB RAM, 256 GB SSD and CentOS 7.2 was installed. The server was installed with Ubuntu 14.04 and equipped with 16vCPUs @2.4 GHz Intel Xeon E5-2676v3, 64 GB RAM and 500 GB SSD hard drive.

(ii) We selected LG G4 mobile phone to be the client machine running Android OS, v5.1.1 (Lollipop) and was equipped with Qualcomm Snapdragon 808 64-bit Hexa-core CPU @1.8 GHz, 3GB RAM and 32 GB internal storage. Notice that AES-NI library cannot be used to accelerate cryptographic operations on this mobile device due to its incompatible CPU, which affects the performance of our schemes in the mobile environment as will be shown in the following section.

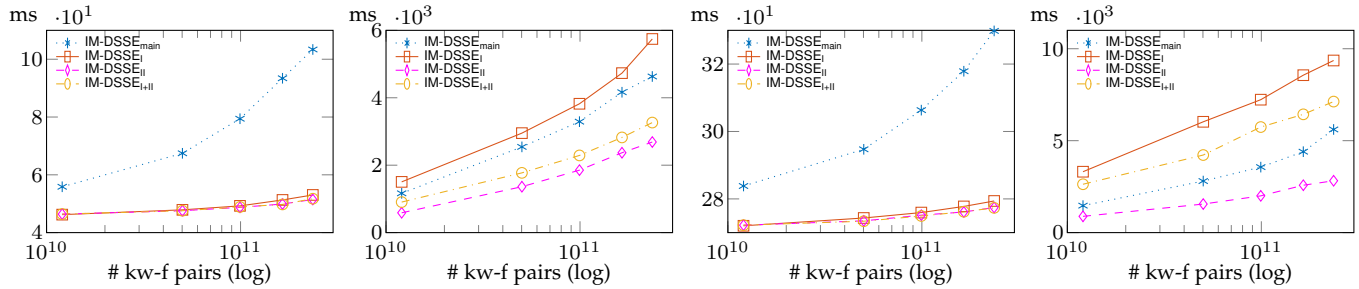
We disabled the slow-start TCP algorithm and maximized initial congestion window parameters in Linux (i.e., 65535 bytes) (see [39] for more insights) to reduce the network impact during the initial phase in case the scheme requires low amount of data to be transmitted.

Performance Results. Figure 2 presents the overall performance in terms of end-to-end cryptographic delay of all the schemes in IM-DSSE framework. In this experiment, we located client and server in the same geographical region, resulting in a network latency of 11.2 ms and a throughput of 264 Mbps. We refer to this configuration as a *fast network* setting. Notice that we only measured the delay due to accessing the encrypted index $\mathbf{I}$, and omitted the time to access encrypted files (i.e., set $\mathcal{C}$) as it is identical for all searchable encryption and non-searchable encryption schemes. For instance, in keyword search, we measured the delay of IM-DSSE$_{main}$ scheme and IM-DSSE$_{I}$ scheme by the time the client sends the request and the server finishes decrypting an entire row of the encrypted index and gets cells whose value is 1. The IM-DSSE$_{main}$ scheme and its extended versions took less than 100 ms to perform a keyword search, while it took less than 2 seconds to update a file. The cost per keyword search depends linearly on the maximum number of files in the database (i.e., $\mathcal{O}(n)$) and yet it is highly practical even for very large numbers of keyword-file pairs (i.e., more than 10^{11} pairs). Indeed, we confirm that the search operation in IM-DSSE is very fast and most of the overhead is due to network communication delay as it will be later analyzed in this section. Note that the costs for adding and deleting files (updates) over the encrypted index are highly similar since their procedure is almost identical.

The keyword search operation delay of IM-DSSE$_{main}$ is higher than that of extended schemes and the difference increases as the size of the encrypted index increases due to two reasons: First, the encrypted index $\mathbf{I}$ in IM-DSSE$_{main}$ scheme is bit-by-bit encrypted compared with 128-bit block encryption in IM-DSSE$_{I}$. Hence, the server needs to derive more AES keys than in IM-DSSE$_{I}$ to decrypt a whole row. Thus, the gap between IM-DSSE$_{main}$ and IM-DSSE$_{I}$ represents the server computation cost required for this key derivation and encryption. Second, the processes in IM-DSSE$_{main}$ scheme are performed subsequently, in which the server needs to receive some information sent from the client first before being able to derive keys to decrypt a row. Such processes in IM-DSSE$_{I}$ and IM-DSSE$_{II}$ can be parallelized, where the client generates the AES-CTR keys while receiving a row of data transmitted from the server. We can see that the delay is similar between IM-DSSE$_{I}$ and IM-DSSE$_{II}$ and IM-DSSE$_{I+II}$. This indicates that using 128-bit encryption significantly reduces the server computation cost to a point, where it becomes negligible as being later shown.

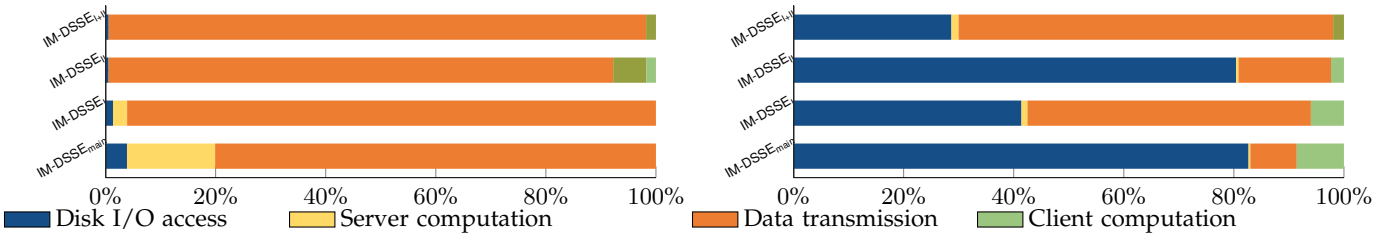
Considering the file update operation, our IM-DSSE$_{main}$ and IM-DSSE$_{II}$ schemes leverage 1-bit encryption and, therefore, it does not require to transfer a 128-bit block to the client first prior to updating the column as in IM-DSSE$_{I}$ and IM-DSSE$_{I+II}$ schemes. Hence, they are faster and less affected by the network latency than IM-DSSE$_{I}$ and IM-DSSE$_{I+II}$. So, the gap between such schemes reflects the data download delays, which will be significantly higher on slower networks as shown in the next experiment. Update in IM-DSSE$_{I+II}$ is considerably faster than in IM-DSSE$_{I}$ because it allows for parallelization, in which the client can precompute AES-CTR keys while receiving data from the server. In IM-DSSE$_{I}$, such keys cannot be pre-computed as they need some information from the server beforehand (i.e., state data $\mathbf{I}[*, j]$.st).

The impact of network quality. The previous experiments



(a) Keyword search (b) File update (c) Keyword search (d) File update

Fig. 4: The latency of IM-DSSE framework with SSD server-storage and (a,b) fast and (c,d) moderate networks.



(a) Keyword search (b) File update

Fig. 5: Detailed costs of IM-DSSE framework with moderate network and SSD server-storage.

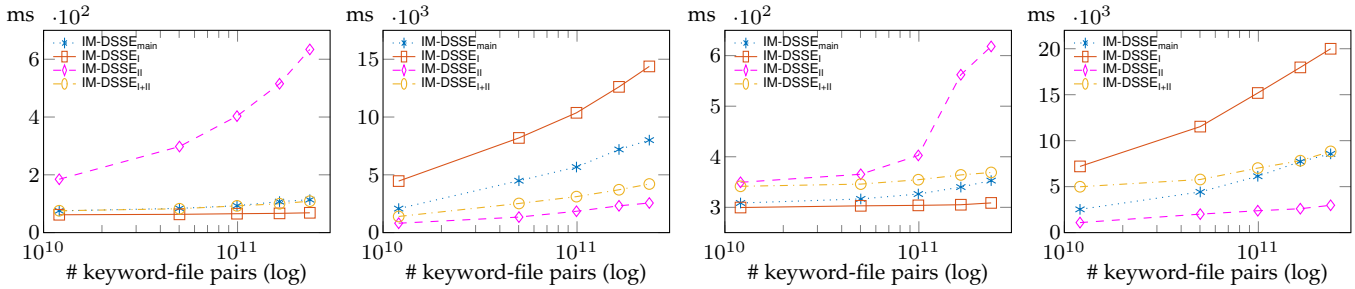
were conducted on a high-speed network, which might not be widely available in practice. Hence, we additionally investigated how our schemes perform when the network speed is degraded. We setup the server to be geographically located distant from the client machine, resulting in the network latency and throughput to be 67.5 ms and 46 Mbps, respectively. Figure 3 shows the end-to-end cryptographic delay of our schemes in this *moderate network* setting. Due to the high network latency, search operation of each scheme is slower than that of fast network by 230ms. The impact of the network latency is clearly shown in the update operation as reflected in Figure 3b. The delays of IM-DSSE$_{I}$, IM-DSSE$_{I+II}$ are significantly higher than those of IM-DSSE$_{main}$ and IM-DSSE$_{II}$. As explained previously, this gap actually reflects the download delay incurred by such schemes.

Storage location of encrypted index: RAM vs. disk. Another important performance factor for DSSE is the encrypted index storage access delay. Hence, we investigated the impact of the encrypted index storage location on the performance of our schemes. Clearly, the ideal case is to store all server-side data on RAM to minimize the delay introduced by storage media access as shown in previous experiments. However, deploying a cloud server with a very large amount of RAM capacity can be very costly. Thus, in addition to the RAM-stored results shown previously, we stored the encrypted index on the secondary storage unit (i.e., SSD drive), and then measured how overall delays of our scheme were impacted by this setting. Figure 4 presents the results with two aforementioned network quality environments (i.e., fast and moderate speeds). In IM-DSSE$_{main}$ and IM-DSSE$_{I}$ schemes, the disk I/O access is incurred by loading a part of the encrypted index including value $\mathbf{I}.v$ and state $\mathbf{I}$.st . It is clear that the disk I/O access time incurred an insignificant latency to the overall delay in terms of keyword search operation as shown in Figure 4a and Figure 4d, since our schemes achieve perfect locality as defined by Cash *et al*. [40]. However, in the file update operation, the delay in IM-DSSE framework was 1–4 seconds more, compared with RAM-based storage. That is because we stored all cells in each row of the encrypted matrix $\mathbf{I}$ in contiguous memory blocks. Therefore, keyword search invokes accessing *subsequent memory blocks* while update operation results in accessing *scattered blocks* which incurs much higher disk I/O access time. Due to the incidence matrix data structure and this storage strategy, our search operation was not affected as much by disk I/O access time as other non-local DSSE schemes (e.g., [5], [13], [41]), which require accessing random memory blocks for security.

Cost breakdown. We dissected the overall cost of our schemes previously presented in Figure 2, Figure 3 and Figure 4 to investigate which factors contribute a significant amount to the total delay of each scheme. For analysis, we selected the cost of our schemes when performing on the largest encrypted index size being experimented (i.e., 2.36×10^{11}) with moderate network speed, where the encrypted index is stored on an SSD drive. Figure 5 presents the major factors that contribute to the total delay of our schemes during keyword search and file update operations.

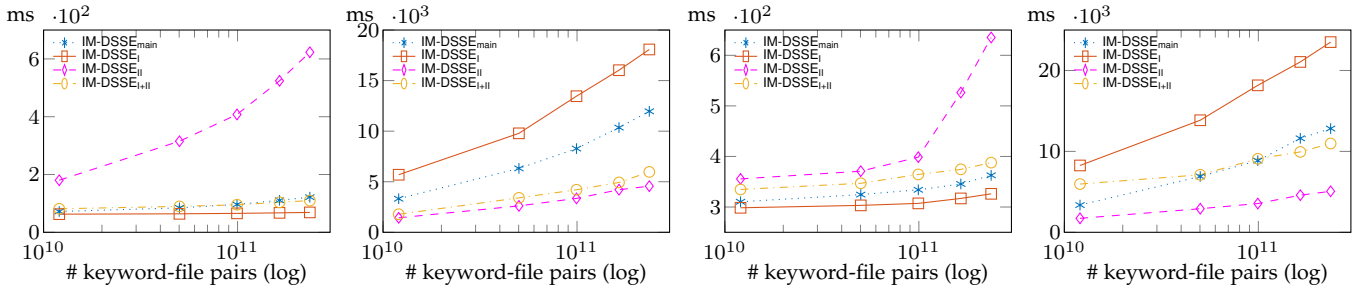
Considering the search operation, it is clear that data transmission occupied the largest amount of delay among all schemes. In our IM-DSSE$_{main}$ and IM-DSSE$_{I}$ schemes, most of the computations were performed by the server wherein cryptographic operations were accelerated by AES-NI so that they only took a small portion of the total, especially in IM-DSSE$_{I}$ scheme. Meanwhile, the client only performed simple computations such as search token generation so that its cost was negligible. In IM-DSSE$_{II}$ and IM-DSSE$_{I+II}$ schemes, encrypted data were decrypted at the client side, while the server did nothing but transmission. Therefore, the client computation cost took a small portion of the total delay and the server's cost was negligible. However, as indicated in §3, the client computation and data transmission in IM-DSSE$_{II}$ and IM-DSSE$_{I+II}$ are fully paral-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2897096, IEEE Transactions on Services Computing



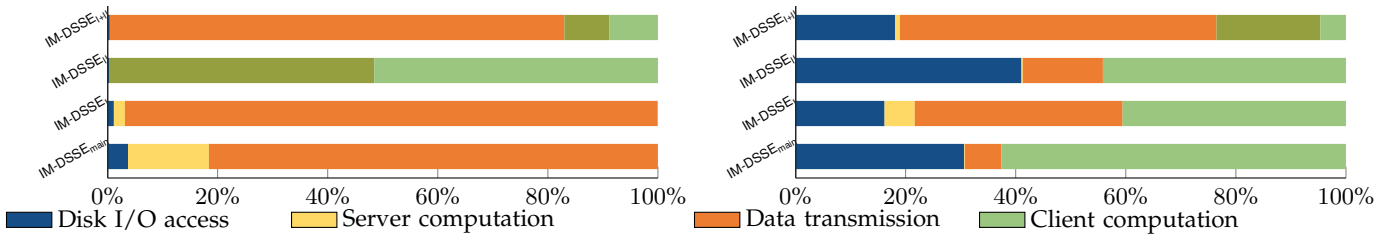
(a) Keyword search (b) File update (c) Keyword search (d) File update

Fig. 6: The latency of IM-DSSE framework on mobile and RAM server-storage with (a,b) fast and (c,d) moderate networks.



(a) Keyword search (b) File update (c) Keyword search (d) File update

Fig. 7: The latency of IM-DSSE framework on mobile and SSD server-storage with (a,b) fast and (c,d) moderate networks.



(a) Keyword search (b) File update

Fig. 8: Detailed costs of IM-DSSE framework on mobile with moderate network and SSD server-storage.

lelizable where their partially parallel costs are indicated by their overlapping area in Figure 5a. Hence, we can infer that client computation was actually dominated by data transmission and, therefore, the computation cost did not affect the total delay of the schemes. As explained above, we stored the encrypted matrix on disk with row-friendly strategy so that the disk I/O access time due to keyword search was insignificant, which contributed less than 3% to the total delay.

In contrast, it is clear that disk I/O access time occupied a considerable proportion of the overall delay of the update operation, especially in the IM-DSSE$_{main}$ and IM-DSSE$_{II}$ schemes due to non-contiguous memory access. Data transmission was the second major factor contributing to the total delay. As the server did not perform any expensive computations, its cost was negligible in all schemes. The client performed cryptographic operations which were accelerated by AES-NI library so that it only contributed less than 7% to the overall cost. Additionally, the client computation was mostly parallelized with the data transmission and the server's operations in IM-DSSE$_{II}$ and IM-DSSE$_{I+II}$ schemes so that it can be considered not to significantly impact the total delay.

Realization on mobile environments. We evaluated our schemes' performance when deployed on a mobile device with limited computational resources. Similar to the desktop experiments, we tested on fast and moderate network by geographically locating the server close and distant from the mobile, respectively. The phone was connected to a local WiFi which, in turn, allowed the establishment of the connection to the server via a wireless network resulting in the latency and throughput of fast network case to be 18.8 ms, 136 Mbps while those of moderate case were 76.3 ms and 44 Mbps, resp. Figure 6 and Figure 7 present the benchmark results with aforementioned network settings when the data in the server were stored on RAM and SSD, respectively. In the mobile environment, the IM-DSSE$_{II}$ scheme performed considerably slower than others in terms of keyword search. That is because, in this scheme, a number of cryptographic operations (i.e., $\mathcal{O}(n)$) were performed by the mobile device. Moreover, these operations were not accelerated by AES-NI library as in our Desktop machine because the mobile CPU did not have special crypto-accelerated instructions. Considering the keyword search performance of IM-DSSE$_{II}$ in the moderate network setting (i.e., Figure 6c and Figure 7c), we can see that its delay significantly increased when the size of encrypted index exceeded 10^{11} keyword-file pairs. This is because starting from this size of the encrypted index, the client computation began to dominate the data transmission cost. The

TABLE 1: Security and asymptotic complexity of some state-of-the-art single-keyword DSSE schemes.

Property/ Scheme	Forward Privacy	Backward Privacy	Client Storage	Index Size	Search Cost	Update Cost	Parallel?
KPR12 [3]	✗	✗	$\mathcal{O}(1)$	$\mathcal{O}(N'+m)$	$\mathcal{O}(r_w)$	$\mathcal{O}(m'')$	✗
KP13 [8]	✗	✗	$\mathcal{O}(1)$	$\mathcal{O}(m \cdot n)$	$\mathcal{O}(r_w \log n)$	$\mathcal{O}(m \log n)$	✓
CJK$^+$14 [5]	✗	✗	$\mathcal{O}(1)$	$\mathcal{O}(N')$	$\mathcal{O}(a_w + d_w)$	$\mathcal{O}(m'' + r'')$	✓
HK14 [10]	✗	✗	$\mathcal{O}(m')$	$\mathcal{O}(N')$	$\mathcal{O}(r_w)$	$\mathcal{O}(m'')$	✗
SPS14 [2]	✓	✗	$\mathcal{O}(N'^{\alpha})$	$\mathcal{O}(N')$	$\mathcal{O}\left(\min\left\{\begin{matrix} d_w + \log N' \\ r_w \log^3 N' \end{matrix}\right\}\right)$	$\mathcal{O}(m'' \log^2 N')$	✗
B16 [11]	✓	✗	$\mathcal{O}(m \log n)$	$\mathcal{O}(N')$	$\mathcal{O}(a_w + d_w)$	$\mathcal{O}(m'')$	✗
LC17 [20]	✓	✗	$\mathcal{O}(m \log n + n \log m)$	$\mathcal{O}(N')$	$\mathcal{O}(r_w)$	$\mathcal{O}(m'')$	✓
BMO17 [24]	✓	✓	$\mathcal{O}(m \log n)$	$\mathcal{O}(N')$	$\mathcal{O}(a_w)$	$\mathcal{O}(m'' \log a_w)$	✗
KKL$^+$17 [21]	✓	✗	$\mathcal{O}(m)$	$\mathcal{O}(N')$	$\mathcal{O}(a_w)$	$\mathcal{O}(m'')$	✓
SDY$^+$18 [22]	✓	✗	$\mathcal{O}(m)$	$\mathcal{O}(N')$	$\mathcal{O}(a_w + d_w)$	$\mathcal{O}(m'')$	✗
IM-DSSE	✓	✓†	$\mathcal{O}(m+n)$	$\mathcal{O}(m \cdot n)$	$\mathcal{O}(n)$	$\mathcal{O}(m)$	✓

• 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 the number of keyword-file pairs. m'' is the number unique keywords included in an updated file, $0 < \alpha < 1$, a_w (resp. d_w) is the number of historical *addition* (resp. *deletion*) operation on keyword w, r_w is the number of files matching keyword w and $r_w = a_w - d_w$.
• We omitted the security parameter κ for analyzed complexity cost.
• This table only presents standard single-keyword DSSE schemes. DSSE schemes with extended query functionalities (e.g., [6], [7], [12], [13], [14], [15]) are summarized in §1.1.
† In IM-DSSE framework, IM-DSSE$_{\text{II}}$ and IM-DSSE$_{\text{I+II}}$ schemes offer backward-privacy (see §4).

update delays of our schemes, especially the IM-DSSE$_{\text{main}}$ and IM-DSSE$_{\text{II}}$ schemes, were substantial in the mobile environment because the mobile platform had to perform intensive cryptographic operations.

Figure 8 shows the decomposition of the total end-to-end delay of our schemes in the out-of-state network setting when the server data were stored on an SSD drive. For the search operation, the detailed costs of IM-DSSE$_{\text{main}}$ and IM-DSSE$_{\text{I}}$ schemes are the same as in the desktop setting since computations were mostly performed by the server while the client only performed some lightweight computation to generate the token. In IM-DSSE$_{\text{II}}$, the client computation contributed almost 100% to the total delay due to $\mathcal{O}(n)$ number of AES-CTR decryptions, compared with $\mathcal{O}(n)/128$ in IM-DSSE$_{\text{I+II}}$ which was all dominated by the data transmission delay. The limitation of computational capability of the mobile device is reflected clearly in Figure 8b, wherein the client computation cost accounted for a considerable amount of the overall delay of most schemes except for the IM-DSSE$_{\text{I+II}}$ scheme.

Caching Discussion. In this article, we reported the delays of our framework without taking optimization into account. Meanwhile, the performance of our framework can be further optimized by applying several caching strategies to minimize the computation, I/O access and communication overhead at both client and server sides. Specifically, for each search operation, one can observe that our framework requires to decrypt and re-encrypt an entire row in the incidence matrix This increases significantly the computation overhead whereas it is not necessary to protect the row confidentiality once its content is already revealed. Therefore, given a keyword to be searched at the first time, the server can decrypt the row, and cache all positions that indicate the files containing the keyword in a compact index such as encrypted dictionary. When the keyword is repeatedly searched, the server can simply look up this index to obtain the search result. This caching strategy reduces the server computation overhead with the cost of maintaining an extra data structure. Both client and server can also cache the content of keywords being searched/update frequently on local persistent memory to reduce the network communication and I/O access thereby, reducing the overall delay. We note that the efficiency of this caching is application-specific since it depends on the characteristics of the outsourced database and the user query. By open-sourcing the implementation of our framework, we leave its optimization to practitioners when deployed on specific use-cases in practice.

6 Conclusions

In this article, we presented IM-DSSE, a new DSSE framework which offers very high privacy, efficient updates, low search latency simultaneously. Our constructions rely on a simple yet efficient incidence matrix data structure in combination with two hash tables that allow efficient and secure search and update operations. Our framework offers various DSSE constructions, which are specifically designed to meet the needs of cloud infrastructure and personal usage in different applications and environments. All of our schemes in IM-DSSE framework are proven to be secure and achieve the highest privacy among their counterparts. We conducted a detailed experimental analysis to evaluate the performance of our schemes on real Amazon EC2 cloud systems. Our results showed the high practicality of our framework, even when deployed on mobile devices with large datasets. We have released the full-fledged implementation of our framework for public use and analysis.

Acknowledgment

This research is kindly supported by Robert Bosch with an unrestricted gift and by NSF CAREER Award CNS-1652389.

References

[1] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *Proc. 13th ACM Conf. Comput. Commun. security*, ser. CCS '06. ACM, 2006, pp. 79–88.
[2] E. Stefanov, C. Papamanthou, and E. Shi, "Practical dynamic searchable encryption with small leakage," in *21st Annu. Network and Distributed System Security Symp. — NDSS 2014*. The Internet Soc., February 23-26, 2014.

[3] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proc. 2012 ACM Conf. Comput. Commun. security*. New York, NY, USA: ACM, 2012, pp. 965–976.

[4] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. 2000 IEEE Symp. Security and Privacy*, 2000, pp. 44–55.

[5] 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 Annu. Network Distributed System Security Symp. — NDSS 2014*. The Internet Soc., February 23-26, 2014.

[6] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," *IEEE Trans. Parallel Distributed Syst.*, vol. 25, no. 1, pp. 222–233, 2014.

[7] 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 Trans. Parallel Distributed Syst.*, vol. 25, no. 11, pp. 3025–3035, 2014.

[8] S. Kamara and C. Papamanthou, "Parallel and dynamic searchable symmetric encryption," in *Financial Cryptography and Data Security (FC)*, ser. Lecture Notes in Comput. Sci. Springer Berlin Heidelberg, 2013, vol. 7859, pp. 258–274.

[9] M. Naveed, M. Prabhakaran, and C. A. Gunter, "Dynamic searchable encryption via blind storage," in *35th IEEE Symp. Security Privacy*, May 2014, pp. 48–62.

[10] F. Hahn and F. Kerschbaum, "Searchable encryption with secure and efficient updates," in *Proc. 2014 ACM SIGSAC Conf. Comput. and Commun. Security*. ACM, 2014, pp. 310–320.

[11] R. Bost, "Sophos – forward secure searchable encryption," in *Proc. 2016 ACM Conf. Comput. Commun. Security*. ACM, 2016.

[12] S. Kamara and T. Moataz, "Boolean searchable symmetric encryption with worst-case sub-linear complexity," *EUROCRYPT 2017*, 2017.

[13] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for boolean queries," in *Advances in Cryptology, CRYPTO 2013*, ser. Lecture Notes in Comput. Sci., vol. 8042, 2013, pp. 353–373.

[14] Z. Fu, X. Wu, C. Guan, X. Sun, and K. Ren, "Toward efficient multi-keyword fuzzy search over encrypted outsourced data with accuracy improvement," *IEEE Trans. Inform. Forensics Security*, vol. 11, no. 12, pp. 2706–2716, 2016.

[15] Q. Wang, M. He, M. Du, S. S. Chow, R. W. Lai, and Q. Zou, "Searchable encryption over feature-rich data," *IEEE Trans. Dependable Secure Computing*, 2016.

[16] 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*, Austin, TX, 2016, pp. 707–720.

[17] A. A. Yavuz and J. Guajardo, "Dynamic searchable symmetric encryption with minimal leakage and efficient updates on commodity hardware," in *Int. Conf. Selected Areas in Cryptography*. Springer, 2015, pp. 241–259.

[18] P. Rizomiliotis and S. Gritzalis, "Oram based forward privacy preserving dynamic searchable symmetric encryption schemes," in *Proc. 2015 ACM Workshop Cloud Computing Security Workshop*. ACM, 2015, pp. 65–76.

[19] 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 *Proc. 2013 ACM SIGSAC Conf. Comput. Commun. security*. ACM, 2013, pp. 299–310.

[20] R. W. Lai and S. S. Chow, "Forward-secure searchable encryption on labeled bipartite graphs," in *Int. Conf. Appl. Cryptography Network Security*. Springer, 2017, pp. 478–497.

[21] K. S. Kim, M. Kim, D. Lee, J. H. Park, and W.-H. Kim, "Forward secure dynamic searchable symmetric encryption with efficient updates," in *Proc. 2017 ACM SIGSAC Conf. Comput. Commun. Security*. ACM, 2017, pp. 1449–1463.

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

[23] M. Etemad, A. Küpçü, C. Papamanthou, and D. Evans, "Efficient dynamic searchable encryption with forward privacy," *Proc. Privacy Enhancing Technologies*, vol. 2018, no. 1, pp. 5–20, 2018.

[24] R. Bost, B. Minaud, and O. Ohrimenko, "Forward and backward private searchable encryption from constrained cryptographic primitives," in *Proc. 2017 ACM SIGSAC Conf. Comput. Commun. Security*. ACM, 2017, pp. 1465–1482.

[25] M. S. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation." in *NDSS*, vol. 20, 2012, p. 12.

[26] D. Pouliot and C. V. Wright, "The shadow nemesis: Inference attacks on efficiently deployable, efficiently searchable encryption," in *Proc. 2016 ACM Conf. Comput. Commun. Security*. ACM, 2016.

[27] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-abuse attacks against searchable encryption," in *Proc. 22nd ACM CCS*, 2015, pp. 668–679.

[28] C. Liu, L. Zhu, M. Wang, and Y.-a. Tan, "Search pattern leakage in searchable encryption: Attacks and new construction," *Inform. Sci.*, vol. 265, pp. 176–188, 2014.

[29] T. Hoang, A. Yavuz, and J. Guajardo, "Practical and secure dynamic searchable encryption via oblivious access on distributed data structure," in *Proc. 32nd Annu. Comput. Security Applications Conf. (ACSAC)*. ACM, 2016.

[30] C. Bösch, A. Peter, B. Leenders, H. W. Lim, Q. Tang, H. Wang, P. Hartel, and W. Jonker, "Distributed searchable symmetric encryption," in *Privacy, Security and Trust (PST), 2014 Twelfth Annu. Int. Conf. on*. IEEE, 2014, pp. 330–337.

[31] M. Naveed, "The fallacy of composition of oblivious ram and searchable encryption." *IACR Cryptology ePrint Archive*, vol. 2015, p. 668, 2015.

[32] M. D. Green and I. Miers, "Forward secure asynchronous messaging from puncturable encryption," in *Security Privacy (SP), 2015 IEEE Symp. on*. IEEE, 2015, pp. 305–320.

[33] T. S. Denis, "LibTomCrypt library," Available at http://libtom.org/?page=features&newsitems=5&whatfile=crypt, Released May 12th, 2007.

[34] S. Gueron, "White Paper: Intel Advanced Encryption Standard (AES) New Instructions Set," Available at https://software.intel.com/sites/default/files/article/165683/aes-wp-2012-09-22-v01.pdf. Software Library available at https://software.intel.com/sites/default/files/article/181731/intel-aesni-sample-library-v1.2.zip, Document Revision 3.01, September 2012.

[35] "sparsehash: An extemely memory efficient hash_map implementation," Available at https://code.google.com/p/sparsehash/, February 2012.

[36] "Zeromq distributed messaging," Available at http://zeromq.org.

[37] T. Hoang, "Im-dsse framework implementation," https://github.com/thanghoang/IM-DSSE, 2017.

[38] "The enron email dataaset." http://www.cs.cmu.edu/~enron/.

[39] N. Dukkipati, T. Refice, Y. Cheng, J. Chu, T. Herbert, A. Agarwal, A. Jain, and N. Sutin, "An argument for increasing tcp's initial congestion window." *Comput. Commun. Rev.*, vol. 40, no. 3, pp. 26–33, 2010.

[40] D. Cash and S. Tessaro, "The locality of searchable symmetric encryption," in *Advances in Cryptology - EUROCRYPT 2014*. Springer, 2014, pp. 351–368.

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

Thang Hoang is currently a PhD student in the School of Electrical Engineering and Computer Science, Oregon State University (September 2015). He received his ME degree in Computer Science from Chonnam National University, Gwangju, South Korea in February, 2014, and BS degree in Computer Science from University of Natural Sciences, Saigon, Vietnam in September, 2010. His research interest currently focuses on privacy-enhancing technologies (e.g., searchable encryption, ORAM) and authentication mechanisms for mobile devices.

Attila Altay Yavuz is an Assistant Professor in the Department of Computer Science and Engineering, University of South Florida (August 2018). He was an Assistant Professor in the School of Electrical Engineering and Computer Science, Oregon State University (09/2014-07/2018). He was a member of the security and privacy research group at the Robert Bosch Research and Technology Center North America (2011-2014). He received his PhD degree in Computer Science from North Carolina State University in August 2011. He received his MS degree in Computer Science from Bogazici University (2006) in Istanbul, Turkey. He is broadly interested in design, analysis and application of cryptographic tools and protocols to enhance the security of computer networks and systems. Attila Altay Yavuz is a recipient of NSF CAREER Award (2017). His research on privacy enhancing technologies (searchable encryption) and intra-vehicular network security are in the process of technology transfer with potential world-wide deployments. He has authored more than 40 research articles in top conferences and journals along with several patents. He is a member of IEEE and ACM.

Jorge Guajardo received the B.Sc. degree in physics and electrical engineering and the M.S. degree in electrical engineering from the Worcester Polytechnic Institute and the Ph.D. degree in electrical engineering and information sciences from the Ruhr-University Bochum. Prior to joining Bosch, he was with Philips Research, The Netherlands, where he performed fundamental work in SRAM physical unclonable functions, leading to the creation of the company Intrinsic-ID. Prior to joining Philips Research, he was with GTE Government Systems, RSA Laboratories, cv cryptovision GmbH, and Infineon Technologies AG. He is currently a Principal Scientist and the Manager of the Security and Privacy Group, Robert Bosch Research and Technology Center, Pittsburgh, PA, USA. During his scientific career, he has co-authored over 40 scientific publications in refereed conferences and journals, 14 issued patents, and several patent applications. His research interests include applied cryptography, embedded security, and noisy crypto. Dr. Guajardo has served in the program committee of the workshop on Cryptographic Hardware and Embedded Systems and the HOST Symposium. He was the Program Co-Chair for the TrustED 2014 and TrustED 2015 workshops.