Practical and Secure Dynamic Searchable Encryption via Oblivious Access on Distributed Data Structure

Thang Hoang
EECS, Oregon State University
Corvallis, OR, 97331
hoangmin@eecs.oregonstate.edu

Attila Altay Yavuz
EECS, Oregon State University
Corvallis, OR, 97331
attila.yavuz@oregonstate.edu

Jorge Guajardo
Robert Bosch LLC — RTC
Pittsburgh, PA, 15222
Jorge.GuajardoMerchan@us.bosch.com

ABSTRACT

Dynamic Searchable Symmetric Encryption (DSSE) allows a client to perform keyword searches over encrypted files via an encrypted data structure. Despite its merits, DSSE leaks search and update patterns when the client accesses the encrypted data structure. These leakages may create severe privacy problems as already shown, for example, in recent statistical attacks on DSSE. While Oblivious Random Access Memory (ORAM) can hide such access patterns, it incurs significant communication overhead and, therefore, it is not yet fully practical for cloud computing systems. Hence, there is a critical need to develop private access schemes over the encrypted data structure that can seal the leakages of DSSE while achieving practical search/update operations.

In this paper, we propose a new oblivious access scheme over the encrypted data structure for searchable encryption purposes, that we call Distributed Oblivious Data structure DSSE (*DOD-DSSE*). The main idea is to create a distributed encrypted incidence matrix on two non-colluding servers such that no arbitrary queries on these servers can be linked to each other. This strategy prevents not only recent statistical attacks on the encrypted data structure but also other potential threats exploiting query linkability. Our security analysis proves that *DOD-DSSE* ensures the unlinkability of queries and, therefore, offers much *higher security than traditional DSSE*. At the same time, our performance evaluation demonstrates that *DOD-DSSE* is *two orders of magnitude faster than ORAM-based techniques (e.g., Path ORAM)*, since it only incurs a small-constant number of communication overhead. That is, we deployed *DOD-DSSE* on geographically distributed Amazon EC2 servers, and showed that, a search/update operation on a very large dataset only takes around one second with *DOD-DSSE*, while it takes 3 to 13 minutes with Path ORAM-based methods.

CCS Concepts

•**Security and privacy** → *Privacy-preserving protocols; Domain-specific security and privacy architectures;*

ACSAC '16, December 05-09, 2016, Los Angeles, CA, USA

DOI: http://dx.doi.org/10.1145/2991079.2991088

Keywords

Privacy enhancing technology; privacy in cloud computing; searchable encryption; ORAM; oblivious data structure

1. INTRODUCTION

Storage-as-a-Service (SaaS) is one of the most common cloud services which allows the clients to store data online so that they can access them from anywhere and reduce data management costs. Despite its merits, SaaS also brings serious privacy issues to users. Once data are outsourced, their privacy can be compromised by the cloud or external attackers (e.g., malware-infected cloud). While standard encryption can protect the content of outsourced data, it prevents the data owner from searching and retrieving information from the cloud and, therefore, invalidates the usability of cloud services. To address the dilemma of user privacy versus data utilization on the cloud, Dynamic Searchable Symmetric Encryption (DSSE) has been proposed, that enables the client to perform search or update operations on encrypted data [4, 13]. In current DSSE approaches, the client constructs an encrypted data structure (denoted as **I**) which associates each search/update query with its corre-

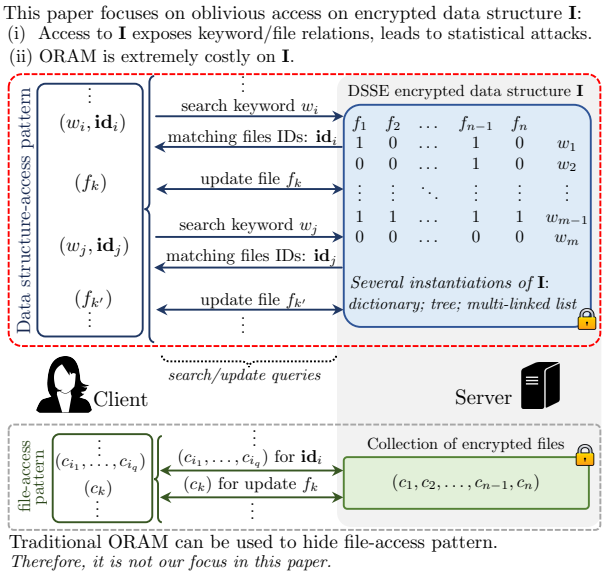


Figure 1: File-access pattern and data structure-access pattern in traditional DSSE.

Table 1: Security and performance of *DOD-DSSE*, traditional DSSE and ODS with specific DSSE data structure types[¶].

Scheme	Security					Performance	Setting
	Data structure-access pattern		Update leakage	Query result size	Statistical attacks	End-to-end crypto delay[†]	# Server
	1-time repetition of an unlinkable query	Full query linkability[§]					
Traditional DSSE [13],[17],[4],[26]	✗	✗	✗	✗	✗	< 0.2 s	1
ODICT	✓	✓	✗*	✗*	✓	192.2 s*	1
OMAT	✓	✓	✓	✓	✓	767.5 s	1
DOD-DSSE	✗[‡]	✓	✓	✓	✓	**1.1 s**	**2**

[¶] We simulated the cost of ODS [25] with Path ORAM protocol [24] on dictionary (ODICT) and incidence matrix (OMAT) data structures.
[†] The delays of schemes were measured in our experiment with an average network latency of 31 ms and throughput of 30 Mbps.
[‡] This leakage does not lead to any statistical attacks.
[*] Due to the sublinear operation time of dictionary data structure, ODS cannot fully hide the length of search/update result without fully padding which is very costly. To evaluate the performance of ODICT over the real network, we only simulated ODICT with average padding.
[§] Full query linkability allows the adversary to perform, for example, frequency analysis [15], resulting in statistical attacks.

sponding files encrypted by standard encryption (e.g., [13, 12, 10, 4, 26]). The client then can outsource encrypted files along with **I** and perform keyword searches or file updates without revealing the keyword/file content.

1.1 Problem Statement

Several DSSE schemes have been proposed (e.g., [13, 12, 10, 4, 21, 26]), which offer a wide spectrum of performance and privacy trade-offs (e.g., fast search/update, information leakage). It is well-known that all DSSE schemes have one common but critical drawback: They leak information during search and update operations due to accesses to the collection of encrypted files and to the encrypted data structure **I** [12, 21, 26] (Figure 1). Specifically,

- Accessing encrypted files leaks *file-access pattern*.
- Accessing the encrypted data structure **I** leaks *data structure-access pattern* including:
  - (i) *search pattern* which reveals if a search query has been repeated,
  - (ii) *update pattern* which reveals various statistical relationships among keywords and files.

These leakages may violate the privacy requirements of privacy-sensitive applications (e.g., healthcare, military) [2, 15] by exposing highly sensitive information [11, 15, 3, 27]. It is mandatory to hide both file-access and data structure-access patterns to achieve a secure DSSE as shown in [16]. One might consider using access pattern hiding techniques such as Oblivious Random Access Memory (ORAM) [18] to conduct oblivious access on both encrypted data structure **I** and encrypted files [16]. Unfortunately, despite recent progress (e.g., [23, 19, 24, 25]), applying ORAM on both **I** and encrypted files [16] is highly expensive in terms of storage and communication overhead [21, 4, 16, 1].

To reduce such costs, it is recommended to separate encrypted files and the encrypted data structure **I**, and then apply different access pattern hiding techniques to each [16]. For instance, Naveed et al. in [16] utilized Path ORAM and Oblivious Data Structure (ODS) [25] to access encrypted files and encrypted data structure, respectively, to reduce the overall cost. Due to the size variation between encrypted files, Path ORAM can be considered as the optimal solution for file-access pattern hiding. However, ODS, an instantiation of position-based ORAM such as Path ORAM designed for data structure, seems not to be ideal for large data structures as found in DSSE. It reduces the client storage but significantly increases the communication overhead and delay (Table 1) and, therefore, it is impractical for real-life applications.

Lack of practical oblivious access for encrypted data structure in the DSSE domain leads us to the following research problem:

> *"Can we propose a protocol to obliviously access the encrypted data structure* **I** *in DSSE which is much more efficient than ORAM, and yet seals leakages of traditional DSSE with only a minimal leakage?"*

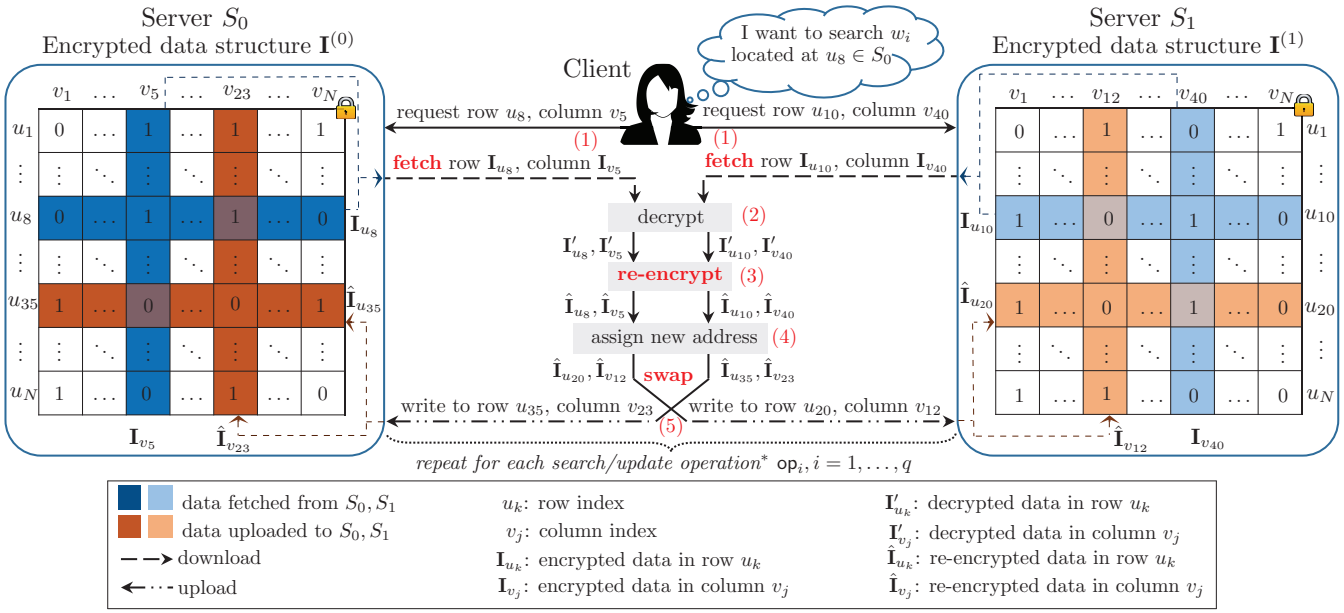
Sealing information leakages from data structure-access pattern while preserving high performance is necessary to make DSSE a feasible solution for cloud systems.

1.2 Our contributions

In this paper, we propose a new oblivious access scheme over the encrypted data structure in DSSE that we call Distributed Oblivious Data structure (*DOD-DSSE*). Our intuition is to leverage two non-colluding servers and exploit the properties of an incidence matrix to seal information leakages in DSSE, while incurring only a small-constant overhead. *DOD-DSSE* achieves the following desirable properties:

• *Significantly higher security than traditional DSSE:* *DOD-DSSE* seals leakages from search and update operations on the encrypted data structure and, therefore, it offers much higher security than traditional DSSE schemes. Specifically, *DOD-DSSE* breaks the linkability between access operations on the encrypted data structure **I**, hides search (i) and update (ii) patterns as shown in Section 1.1. This allows *DOD-DSSE* to prevent a server from learning whether or not the same query is continuously repeated as well as the relationship between keywords and files. Therefore, *DOD-DSSE* is secure against statistical attacks to which all traditional DSSE schemes are vulnerable due to leakages from data structure-access patterns. Table 1 summaries the security of *DOD-DSSE*, compared with traditional DSSEs.

• *Significantly higher efficiency than ORAM-based methods*: *DOD-DSSE* offers much lower bandwidth overhead and delay than applying ORAM-based techniques (e.g., ODS [25]) to the encrypted data structure **I**. Our experimental results



*This strategy provides the *unlinkability between queries* as the same keyword/file generates different queries to each server. Multiple repetitions of the same operation cannot be observed if S_0 and S_1 are non-colluding. Hence, it prevents leakage from search & update patterns and invalidates statistical attacks.

Figure 2: Illustration of a search/update operation on the DSSE encrypted data structure in *DOD-DSSE* scheme.

indicate that *DOD-DSSE* is much faster than using ODS with Path ORAM protocol on dictionary[1] and incidence matrix[2] data structures, respectively, in terms of end-to-end delay (Table 1). *DOD-DSSE* only takes around one second to perform an access operation on a very large data structure (see Section 5 for a detailed comparison).

• *Formal security analysis and full-fledged implementation*: We fully analyze the security and information leakages of *DOD-DSSE* (Section 4.3). We provide a detailed implementation of *DOD-DSSE* on two virtual Amazon EC2 servers and strictly evaluated the performance of *DOD-DSSE* on real network settings (Section 5). *We also released the implementation of DOD-DSSE for public use*[3].

These properties make *DOD-DSSE* an ideal alternative for privacy-critical cloud applications. We briefly describe the main idea of *DOD-DSSE* as follows:

Main idea. Existing DSSE schemes rely on a deterministic association between the (address) token of a query and its corresponding encrypted result in the DSSE data structure. In other words, each query x is represented by a deterministic address token-data tuple $(u_x, \mathbf{I}_{u_x})$ in the encrypted data structure $\mathbf{I}$. Despite permitting consistent and fast search/update operations, these deterministic relations leak data structure-access pattern as discussed in Section 1.1. The research challenge is to devise cryptographic methods that can create a random uniform address token-data tuple $(u_x, \mathbf{I}_{u_x})$ for each query x in an oblivious way with just a small number of communication rounds and processing time. *DOD-DSSE* achieves this by using a "*fetch–reencrypt–swap*" strategy between two servers as follows:

First, the client creates two encrypted data structures, each including address-data tuples $(u_x, \mathbf{I}_{u_x})$ of all possible search and update queries, and then sends them to two non-colluding servers (S_0, S_1), respectively. To perform a search or update operation, the client sends a search query and an update query to each server. One query is for the real operation while the other three are randomly selected (fake) queries (Figure 2, step (1)). Each server sends back to the client the corresponding address-data tuples that have been queried. After that, the client decrypts the received data to obtain the result (step (2)), and then re-encrypts them (step (3)). The client creates new address-data tuple for each performed query by assigning re-encrypted data to a random address (step (4)). Finally, the client swaps such address-data tuples and writes them back to the other server (step (5)). That means the new address-data tuple of the query being read from server S_0 will be written to server S_1 and vice versa. This strategy makes each server observe a randomized data structure-access pattern with only one-time repetition of a *unlinkable query* that has been performed on the other server, provided that the two servers do not collude. Section 4 presents detailed constructions of *DOD-DSSE*.

Limitations. (i) We assume that the two servers storing the encrypted data structures are non-colluding; (ii) *DOD-DSSE* leaks to each server S_b ($b = 0, 1$) a one-time repetition of a query that was previously performed on the other server. This query cannot be linked to any other queries performed on S_b and it *never repeats* on S_b again.

We note that the performance and security benefits of

[1]Dictionary is a $\langle \mathsf{key}, \mathsf{value} \rangle$ structure such that given a keyword key, its corresponding value is a list of file IDs in which key appears. This data structure offers sublinear search time, but leaks information due to its size depending on the file IDs associated with key.

[2]Incidence matrix is a data structure which represents the relationship between keywords (indexing rows) and files (indexing columns) via its cell value. For example, if matrix entry $\mathbf{I}[i, j]$ is set to 1, it means the keyword indexing the ith row appears in the file indexing the jth column. Similarly, the $\mathbf{I}[i, j]$ entry is set to 0 if the keyword indexing the ith row does not appear in the file indexing column j.

[3]Available at https://github.com/thanghoang/DOD-DSSE/

DOD-DSSE well-justifies these limitations. Furthermore, (i) two practical non-colluding servers can be found in real world as competitive cloud providers such as Amazon, Microsoft and Google are very unlikely to collude against their client. (ii) Indeed, we show that with minimal information leakage (i.e., one-time repetition of an unknown and unlinkable query on the other server), *DOD-DSSE* seals all search/update patterns, prevents statistical attacks which are main objectives of a secure DSSE. At the same time, it achieves extremely efficient performance compared to using ORAM-based techniques. Therefore, *DOD-DSSE* offers an ideal security-performance trade-off for DSSE.

2. PRELIMINARIES

Notation. Given a bit b, $\neg b$ means the complement of b. $||$ denotes a concatenation operation. $x \xleftarrow{\$} \mathcal{S}$ means variable x is randomly and uniformly selected from set $\mathcal{S}$. $\mathcal{S} \setminus \{x\}$ denotes $x \in \mathcal{S}$ is removed from $\mathcal{S}$, and $|\mathcal{S}|$ denotes cardinality of set $\mathcal{S}$. $\{x_i\}_{i=1}^{l}$ denotes $(x_1, \ldots, x_l)$. κ is a security parameter. $\mathcal{E} = (\mathsf{Enc}, \mathsf{Dec}, \mathsf{Gen})$ is an IND-CPA secure symmetric encryption scheme [14], which is comprised of three algorithms: key generation $k \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$; encryption with secret key k on message M as $c \leftarrow \mathcal{E}.\mathsf{Enc}_k(M)$; decryption as $M \leftarrow \mathcal{E}.\mathsf{Dec}_k(c)$. We denote $c \leftarrow \mathcal{E}.\mathsf{Enc}_k(M, a)$ and $M \leftarrow \mathcal{E}.\mathsf{Dec}_k(c, a)$ as IND-CPA encryption and decryption with a counter a, respectively. $H : \{0,1\}^* \rightarrow \{0,1\}^{|H|}$ is an ideal cryptographic hash function, where $|H|$ is the length of hash output. $\tau \leftarrow \mathsf{KDF}(x)$ is a key derivation function which takes as input an arbitrary string $x \in \{0,1\}^*$ and outputs a key τ. f and w denote a file and a keyword, respectively. m and n denote the maximum number of files and keywords in the dataset, respectively. $\mathbf{f} = (f_{id_1}, \ldots, f_{id_m})$ denotes the collection of files. If $\mathbf{I}$ is a matrix then $\mathbf{I}_u$ denotes the row indexed by u (i.e., $\mathbf{I}_u = \mathbf{I}[u, *]$). We abuse this notation to also indicate a whole column indexed by u (i.e., $\mathbf{I}_u = \mathbf{I}[*, u]$). This abuse of notation simplifies somewhat the presentation of our algorithms as well as our security analysis. $\mathbf{I}_u[j]$ means accessing the jth element of $\mathbf{I}_u$. $\mathsf{read}(u, \mathsf{data})$ (or $\mathsf{data} \leftarrow \mathsf{Read}(u)$) and $\mathsf{write}(u, \mathsf{data})$ are read and write operations on data at address u, respectively and $u \leftarrow \mathsf{pos}(\mathsf{data})$ returns the address u where data is located.

Security Definition. The security notion for DSSE is *Dynamic adaptive security against Chosen-Keyword Attacks (CKA2)* security [13, 12, 21, 26]), which captures information leakage via leakage functions characterizing the information leakage due to search and update operations (see [12, 21, 26] for the details). All existing DSSE schemes (e.g. [10, 5, 4, 21, 17, 26] with Dynamic CKA2 security leak *data structure-access pattern* which can be defined [23] as follows:

DEFINITION 1. Data structure-access pattern *is a data request sequence* $\overrightarrow{\sigma_b} = \{\mathsf{op}_i^{(b)}, u_i^{(b)}, \mathsf{data}_i^{(b)}\}_{i=1}^{q}$ *of length* q *over the encrypted data structure* $\mathbf{I}$ *on server* S_b *during search and update operations, where* $\mathsf{op}_i^{(b)} \in \{\mathsf{read}(u_i^{(b)}, \mathsf{data}_i^{(b)}), \mathsf{write}(u_i^{(b)}, \mathsf{data}_i^{(b)})\}$, $u_i^{(b)}$ *is the address identifier on* S_b *to be read or written and* $\mathsf{data}_i^{(b)}$ *is the data located at* $u_i^{(b)}$ *to be read or written on* S_b.

Data structure-access pattern leaks search patterns and update patterns which can be defined as follows:

- *Search pattern* indicates if the same keyword has been previously searched. Given a query on w at time t, the search pattern is a binary vector of length t with 1 at location i if the search $i \leq t$ was for w, 0 otherwise.
- *Update pattern* indicates information being leaked during an update operation with different levels, in that level 1 (as defined in [26]) leaks least information which is similar to search pattern. We refer readers to [26] for a detailed description.

The ORAM security definition [23] on server S_b is as follows:

DEFINITION 2. *Let* $\mathbf{AP}_b(\overrightarrow{\sigma_b})$ *denote an (possibly randomized) access pattern to* S_b *given the sequence of data requests* $\overrightarrow{\sigma_b}$ *as defined in Definition 1. The ORAM scheme is secure if for* any *two* $\overrightarrow{\sigma_b}$ *and* $\overrightarrow{\sigma_b}'$ *of the same length, their access patterns* $\mathbf{AP}_b(\overrightarrow{\sigma_b})$ *and* $\mathbf{AP}_b(\overrightarrow{\sigma_b}')$ *are computationally indistinguishable by anyone but the client.*

3. OUR MODELS

System Model. Our system model comprises a client and two servers $\mathbf{S} = (S_0, S_1)$, each storing an instance of an encrypted data structure created from the same file collection.

ASSUMPTION 1. *Servers communicate with the client via private channels. (i)* (S_0, S_1) *are honest-but-curious, meaning that they show interest in learning information but follow the protocol faithfully; they do not inject malicious inputs to the system.* (ii) S_0 *and* S_1 *do not collude.*

Security Model. In *DOD-DSSE*, data request sequences $(\overrightarrow{\sigma_0}, \overrightarrow{\sigma_1})$ of length q in Definition 1 are independently observed by servers (S_0, S_1), respectively. We assume that the encrypted data structure can store up to N distinct data items, each corresponding with either a search or an update query. Each item is represented by a unique address-data tuple (u, data) in the data structure. The security of *DOD-DSSE* scheme relies on the fact that any access operations $\mathsf{op}_i^{(b)} \in \overrightarrow{\sigma_b}$ observed by server S_b, for all $1 \leq i \leq q$ are guaranteed to be unlinkable. This achievement enables us to protect the data structure-access pattern in each server as defined in Definition 1. We define the unlinkability property of access operations on the encrypted data structure in DSSE as follows:

DEFINITION 3. *Let* $(u_i^{(b)}, \mathsf{data}_i^{(b)}), (u_j^{(b)}, \mathsf{data}_j^{(b)})$ *be address-data tuples requested by access operations* $(\mathsf{op}_i^{(b)}, \mathsf{op}_j^{(b)}) \in \overrightarrow{\sigma_b}$ *observed by server* S_b*, respectively.* $\mathsf{op}_i^{(b)}$ *is called unlinkable to* $\mathsf{op}_j^{(b)}$ *if the probability that* $\langle (u_i^{(b)}, \mathsf{data}_i^{(b)}), (u_j^{(b)}, \mathsf{data}_j^{(b)}) \rangle$ *represent the same item being accessed is* $\frac{1}{N}$*, where* N *is the number of distinct items stored in* S_b.

Note that this is the upper bound of linkability probability that one can infer from two arbitrary tuples. The unlinkability in Definition 3 implies the *DOD-DSSE* security definition, which is comparable to that of ORAM as follows:

DEFINITION 4. *DOD-DSSE on the server* S_b *leaks no information beyond ORAM (Definition 2) with the exception of one-time repetition of an unknown and unlinkable query on the other server* $S_{\neg b}$*, to which* S_b *does not have access.*

We will give a detailed security analysis in Section 4.3 after presenting the construction of *DOD-DSSE* in the following section.

4. THE PROPOSED SCHEME

We first describe the encrypted data structure used in *DOD-DSSE*, followed by several newly proposed algorithms.

4.1 DOD-DSSE Encrypted Data Structure

An encrypted data structure enables encrypted search and update operations for a keyword w or a file f_{id}. In this paper, we adopt a keyword-file *incidence matrix* to be the DSSE data structure $\mathbf{I}$ due to its security and performance advantages, compared with other typical types such as multi-linked list [13], dictionary [17], and tree [12]. For the sake of simplicity, we assume that keywords are assigned to row indices while file IDs are assigned to column indices. We assume our file collection $\mathbf{f}$ consists of $m' \leq N$ unique keywords and $n' \leq N$ file IDs, where N is the maximum number of unique keywords and files that our $\mathbf{I}$ can support. We construct the encrypted index $\mathbf{I}^{(0)}$ for server S_0 and $\mathbf{I}^{(1)}$ for S_1 as follows:

First, we assign each item x, where x is a keyword or file ID, to a unique random address in $\mathbf{I}^{(b)}$ as $u_x^{(b)} \xleftarrow{\$} \mathcal{L}_b$ for each $b \in \{0,1\}$, where $\mathcal{L}_b$ is the set of unassigned row or column indices in $\mathbf{I}^{(b)}$. For security reasons which will be analyzed in Section 4.3, $|\mathcal{L}| = 2N$. In other words, $\mathbf{I}$ *is a square matrix of size* $2N \times 2N$ to cover N keywords and files.

We represent the relationship between a keyword and a file by a cell value in $\mathbf{I}^{(b)}$. $\mathbf{I}^{(b)}[i,j] = 1$ means the keyword w assigned to row i appears in the file assigned to column j in server S_b and $\mathbf{I}^{(k)}[i,j] = 0$ otherwise. We can consider the data $\mathbf{I}_{u_x}^{(b)}$ of item x as a row or column data which is a binary string of length $2N$ representing the relationship between x and its object in server S_b. A search or update query of x will correspond with retrieving a whole row or column respectively.

Finally, we encrypt every cell in $\mathbf{I}^{(b)}$ using bit-by-bit IND-CPA encryption as $\mathbf{I}^{(b)}[i,j] \leftarrow \mathcal{E}.\mathsf{Enc}_{\tau_i^{(b)}}(\mathbf{I}^{(b)}[i,j], c_j^{(b)})$, where $c_j^{(b)}$ is a value derived from column index j and a counter $a_j^{(b)}$ in server S_b and $\tau_i^{(b)}$ is a row key derived from the row index i and a counter $a'^{(b)}_i$ as well as a secret key generated for server S_b. We store the information of $a_j^{(b)}$ and $a'^{(b)}_i$ in global counter arrays $\mathbf{C}_{\mathsf{r}}^{(b)}$ and $\mathbf{C}_{\mathsf{c}}^{(b)}$ of length $2N$ which can be retrieved, e.g., as $a_j^{(b)} \leftarrow \mathbf{C}_{\mathsf{c}}^{((b))}[j]$, for each $b \in \{0,1\}$. We describe constructions of $\mathbf{I}^{(0)}, \mathbf{I}^{(1)}$ in Algorithm 2. Figure 3 depicts the structure and content of $\mathbf{I}^{(0)}$ and $\mathbf{I}^{(1)}$.

We create data structures T_w, T_f stored on the client-side for keywords and files, respectively which are defined as:

$$T : (H(x), \langle u_x^{(0)}, u_x^{(1)}, b_x \rangle).$$

(T_w, T_f) are used to keep track of the assigned addresses $(u_x^{(0)}, u_x^{(1)})$ of each item x on servers (S_0, S_1), respectively, as well as the server ID (i.e., $b_x \in \{0,1\}$) where it was last accessed. We define functions for T as follows:

- $T.\mathsf{insert}(\mathsf{key}, \mathsf{value})$: insert hash of x (i.e., $H(x)$) as key and x's information $\langle u_x^{(0)}, u_x^{(1)}, b_x \rangle$ as value into T. It can accept null as key, in which the value (i.e, $\langle u_{\mathsf{null}}^{(0)}, u_{\mathsf{null}}^{(1)}, b_{\mathsf{null}} \rangle$) will be inserted into empty slots in T.
- $j_x \leftarrow T.\mathsf{get}(H(x))$: find the index j_x of x in T using its hash value $H(x)$.
- $j_x \leftarrow T.\mathsf{lookup}(u_x^{(b)}, b)$: find the index j_x of x in T using its address u_x on the server S_b.

Encrypted data structure $\mathbf{I}^{(0)}$

u_f / u_w	1 f_{id_1}	2	3 f_{id_2}	4	5 f_{id_3}	6 f_{id_4}	...	$2N$
1 w_{i_1}	1 (1)	1	1 (0)	0	0 (1)	0 (1)	...	1
2	0	1	1	1	0	0	...	1
3	1	1	0	0	1	0	...	1
4 w_{i_2}	1 (0)	0	0 (0)		1 (1)	0 (1)	...	0
5	1	0	1	0	1	0	...	1
⋮	⋮	0	⋮	1	⋮	⋮	⋱	0
$2N$ w_{i_3}	1 (1)	1	1 (1)	0	1 (0)	0 (0)	...	1

Encrypted data structure $\mathbf{I}^{(1)}$

u_f / u_w	1	...	21 f_{id_2}	22	23 f_{id_1}	24	...	$2N$ f_{id_4}
1	1	...	1	1	0	0	...	0
⋮	⋮	⋱	⋮	⋮	⋮	⋮	⋱	⋮
41 w_{i_3}	0	...	0 (1)	0	1 (1)	0	...	1 (0)
42 w_{i_2}	1	...	0 (0)	1	1 (0)	1	...	1 (1)
43 w_{i_1}	0	...	1 (0)	1	1 (1)	1	...	0 (1)
⋮	⋮	⋱	⋮	⋮	⋮	⋮	⋱	⋮
$2N$	1	...	1	0	0	0	...	1

$\binom{i}{w_k}$: keyword w_k assigned to row i.

$\binom{j}{f_{id_k}}$: file f_{id_k} assigned to column j.

: dummy row/column.

1 (0) → encrypted bit generated by bit-by-bit encryption [26]. actual (i.e., un-encrypted) bit that shows the relation between keyword and file.

Figure 3: Two encrypted instances of an incidence matrix data structure generated from the same file collection $\mathbf{f}$.

Information about x can be retrieved via its index j_x in T as $(H(x), \langle u_x^{(0)}, u_x^{(1)}, b_x \rangle) \leftarrow T[j_x]$.

We can see that $\mathbf{I}$ is a $2N \times 2N$ matrix storing the relationship between N unique keywords and N files. There are at least N empty rows and N empty columns in $\mathbf{I}$. We also include in T_w, T_f sets of such "dummy" addresses in $\mathbf{I}^{(0)}, \mathbf{I}^{(1)}$, denoted as $T_w.\mathcal{L}_0, T_w.\mathcal{L}_1$, for keywords and $T_f.\mathcal{L}_0, T_f.\mathcal{L}_1$ for files respectively. This is to achieve the consistency of *DOD-DSSE* data structure and security (see Section 4.3).

4.2 Proposed DOD-DSSE Algorithms

We present detailed implementations of *DOD-DSSE* in three main algorithms with four subroutines. We provide in Subroutine 3 the decryption procedure for a row/column data of $\mathbf{I}$. The encryption procedure *DOD-DSSE*.Enc() is not explicitly defined and it works similarly to Subroutine 3 by substituting in lines 4 and 8 $\mathcal{E}.\mathsf{Dec}()$ for $\mathcal{E}.\mathsf{Enc}()$.

The main operation of *DOD-DSSE* is presented in Algorithm 3. First, the client generates for each server one search and one update queries including two row indices (one is dummy) and two column indices (one is dummy) using Subroutine 2 (step 1). The client reads data in such addresses from the servers and decrypts only data in non-dummy addresses (steps 2 – 7). After that, the client can perform an actual search or update operation over the decrypted data (steps 8–11). Finally, the decrypted data are re-encrypted with new counters and written back to addresses in the server from where they were read as well as the dummy addresses in the other server (steps 17–21). Notice that such data need to be updated before re-encryption to preserve keyword-file relations (steps 15–16), and their new addresses in the other server are updated in hash tables (steps 12–16) so that they can be retrieved correctly in subsequent operations.

Algorithm 1 $K \leftarrow$ *DOD-DSSE*.Gen(1^κ)

```
    # Generate keys to encrypt two data structures
1:  k_0 ← E.Gen(1^κ), k_1 ← E.Gen(1^κ)
2:  return K ← (k_0, k_1)
```

Algorithm 2 $(K, T_w, T_f, \mathbf{I}^{(0)}, \mathbf{I}^{(1)}) \leftarrow$ *DOD-DSSE*.$\mathsf{Init}(\kappa, \mathbf{f})$

Initialization: Set T_w, T_f to be empty
$\mathbf{I'}^{(b)}[i,j] \leftarrow 0$ and $\mathbf{C}_{\mathsf{r}}^{(b)}[i] \leftarrow 1$ and $\mathbf{C}_{\mathsf{c}}^{(b)}[j] \leftarrow 1$, for all $1 \le i \le 2N$, $1 \le j \le 2N$ and for each $b \in \{0,1\}$
$T_x.\mathcal{L}_b \leftarrow \{1, 2, \ldots, 2N\}$ for each $x \in \{w, f\}$ and $b \in \{0,1\}$
$K \leftarrow$ *DOD-DSSE*.$\mathsf{Gen}(1^\kappa)$
Extract unique keywords $\mathbf{w} = (w_1, \ldots, w_{m'})$ from files $\mathbf{f} = (f_{id_1}, \ldots, f_{id_{n'}})$
`# Create unecnrypted data structure` $\mathbf{I'}^{(0)}$ `for server` S_0`,` $\mathbf{I'}^{(1)}$ `for server` S_1
for each $w_i \in \{w_1, \ldots, w_{m'}\}$ **do**
  $(u_i^{(0)}, u_i^{(1)}, T_w) \leftarrow$ *DOD-DSSE*.$\mathsf{Assign}(H(w_i), T_w)$
  for each $id_j \in \{id_1, \ldots, id_{n'}\}$ **do**
    $(v_j^{(0)}, v_j^{(1)}, T_f) \leftarrow$ *DOD-DSSE*.$\mathsf{Assign}(H(id_j), T_f)$
    if w_i appears in f_{id_j} **then**
      $\mathbf{I'}^{(0)}[u_i^{(0)}, v_j^{(0)}] \leftarrow 1$, $\mathbf{I'}^{(1)}[u_i^{(1)}, v_j^{(1)}] \leftarrow 1$
`# Reserve row and column indices for new keywords and files being added in the future`
Call *DOD-DSSE*.$\mathsf{Assign}(\mathsf{null}, T_x)$ multiple times until all values in all N slots in T_x are filled, for each $x \in \{w, f\}$
`# Encrypt every row of data structures` $\mathbf{I'}^{(0)}$ `and` $\mathbf{I'}^{(1)}$
for each server $S_b \in \{S_0, S_1\}$ **do**
  $\mathbf{I}^{(b)}[i, *] \leftarrow$ *DOD-DSSE*.$\mathsf{Enc}(\mathbf{I'}^{(b)}[i, *], i, b, K)$ for each row $i \in \{1, \ldots, 2N\}$
return $(K, T_w, T_f, \mathbf{I}^{(0)}, \mathbf{I}^{(1)})$ `# The client sends` $\mathbf{I}^{(0)}$ `to` S_0`,` $\mathbf{I}^{(1)}$ `to` S_1 `and keeps` (K, T_w, T_f) `secret`

Algorithm 3 $\mathbf{id} \leftarrow$ *DOD-DSSE*.$\mathsf{Access}(\mathsf{op}, x, \mathbf{S}, K, T_w, T_f)$

`# Generate search and update queries`
$(\{u_j^{(b)}, \widetilde{u}_j^{(b)}\}_{j \in \{w,f\}, b \in \{0,1\}}, \beta) \leftarrow$ *DOD-DSSE*.$\mathsf{CreateQueries}(H(x))$
for each $j \in \{w, f\}$ **do**
  `# Retrieve from each server 2 columns or 2 rows depending on index` j
  for each server $S_b \in \{S_0, S_1\}$ **do**
    $\mathbf{I}_{u_j}^{(b)} \leftarrow \mathsf{Read}(u_j^{(b)})$ from S_b
    $\mathbf{I}_{\widetilde{u}_j}^{(b)} \leftarrow \mathsf{Read}(\widetilde{u}_j^{(b)})$ from S_b
  `# Decrypt retrieved row and column`
  $\mathbf{I'}_{u_j}^{(b)} \leftarrow$ *DOD-DSSE*.$\mathsf{Dec}(\mathbf{I}_{u_j}^{(\beta)}, u_j^{(\beta)}, \beta, K)$
  $\mathbf{I'}_{\widetilde{u}_j}^{(\neg b)} \leftarrow$ *DOD-DSSE*.$\mathsf{Dec}(\mathbf{I}_{\widetilde{u}_j}^{(\neg\beta)}, \widetilde{u}_j^{(\neg\beta)}, \neg\beta, K)$
if $\mathsf{op} = \mathsf{search}$ **then**
  Extract column index from $\mathbf{I'}_{u_w}^{(\beta)}$ as $\mathbf{id} \leftarrow (y_1, \ldots, y_l)$, where $\mathbf{I'}_{u_w}^{(\beta)}[y_k] = 1$ for each $y_k \in \{1, \ldots, 2N\} \setminus T_f.\mathcal{L}_\beta$, $1 \le k \le l$
else
  Update list of keywords in $\mathbf{I'}_{u_f}^{(\beta)}$ and key in T_f, T_w corresponding with the file being updated
for each $j \in \{w, f\}$ **do**
  `# Update new address of non-dummy column and row data in the other server`
  $T_j \leftarrow$ *DOD-DSSE*.$\mathsf{UpdateT}(T_j, u_j^{(\beta)}, \beta, u_j^{(\neg\beta)})$
  $T_j \leftarrow$ *DOD-DSSE*.$\mathsf{UpdateT}(T_j, \widetilde{u}_j^{(\neg\beta)}, \neg\beta, \widetilde{u}_j^{(\beta)})$
  Update cell values in $\mathbf{I'}_{u_j}^{(\beta)}$ and $\mathbf{I'}_{\widetilde{u}_j}^{(\neg\beta)}$ to preserve keyword-file relations with changes at steps 13, 14
  Create ($\mathbf{I'}_{u_j}^{(\neg\beta)}$, $\mathbf{I'}_{u_j}^{(\neg\beta)}$) based on $\mathbf{I'}_{u_j}^{(\beta)}$, ($\mathbf{I'}_{\widetilde{u}_j}^{(\neg\beta)}$) respectively to preserve keyword-file relation consistency in both servers
`# Increase counter values in global counter arrays`
$\mathbf{C}_{\mathsf{r}}^{(b)}[i] \leftarrow \mathbf{C}_{\mathsf{r}}^{(b)}[i] + 1$ and $\mathbf{C}_{\mathsf{c}}^{(b)}[j] \leftarrow \mathbf{C}_{\mathsf{c}}^{(b)}[j] + 1$ for each $b \in \{0,1\}$, $i \in \{u_w^{(0)}, \widetilde{u}_w^{(0)}, u_w^{(1)}, \widetilde{u}_w^{(1)}\}$ and $j \in \{u_f^{(0)}, \widetilde{u}_f^{(0)}, u_f^{(1)}, \widetilde{u}_f^{(1)}\}$
for each $j \in \{w, f\}$ **do**
  for each server $S_b \in \{S_0, S_1\}$ **do**
    `# Re-encrypt retrieved data with newly updated counters`
    $\hat{\mathbf{I}}_{u_j}^{(b)} \leftarrow$ *DOD-DSSE*.$\mathsf{Enc}(\mathbf{I'}_{u_j}^{(b)}, u_j^{(b)}, b, K)$, $\hat{\mathbf{I}}_{\widetilde{u}_j}^{(b)} \leftarrow$ *DOD-DSSE*.$\mathsf{Enc}(\mathbf{I'}_{\widetilde{u}_j}^{(b)}, \widetilde{u}_j^{(b)}, b, K)$
    `# Write re-encrypted data back to corresponding server` S_b
    $\mathsf{Write}(u_j^{(b)}, \hat{\mathbf{I}}_{u_j}^{(b)})$, $\mathsf{Write}(\widetilde{u}_j^{(b)}, \hat{\mathbf{I}}_{\widetilde{u}_j}^{(b)})$ to S_b
return id

Subroutine 1 $(u_x^{(0)}, u_x^{(1)}, T) \leftarrow$ *DOD-DSSE*.Assign(x, T)

Assign hash x of an item to a random address in each server, and store assigned addresses in the hash table T:

```
    # Pick a random address from dummy set in each server
1:  for each b ∈ {0,1} do
2:      u_x^(b) ←$ T.L_b
3:      T.L_b ← T.L_b \ {u_x^(b)}
    # Randomly assign server ID for x
4:  b_x ←$ {0,1}
    # Store assigned info of x in hash table
5:  T.insert(x, ⟨u_x^(0), u_x^(1), b_x⟩)
6:  return (u_x^(0), u_x^(1), T)
```

Subroutine 2 $(\mathcal{U}, \beta) \leftarrow \mathsf{CreateQueries}(x)$

Generate search and update queries given an actual item x to be accessed, where x can be a keyword w or file f:

```
    # Get hash table entry for x and its info
1:  j_x ← T_x.get(x)
2:  β ← T_x[j_x].b_x
3:  u_x^(β) ← T_x[j_x].u_x^(β)
    # Select a random non-dummy row/column index u_x̄^(β).
    # If x is w then x̄ is f and vice-versa
4:  u_x̄^(β) ←$ {1,...,2N} \ T_x̄.L_β
5:  for each j ∈ {w, f} do
        # Select a random non-dummy index from server S_¬β
6:      u_j^(¬β) ←$ {1,...,2N} \ T_j.L_¬β
        # Randomly select dummy row & column indices in S_0, S_1
7:      for each b ∈ {0,1} do
8:          ũ_j^(b) ←$ T_j.L_b
9:  return (U, β), where U = {u_j^(b), ũ_j^(b)}_{j∈{w,f}, b∈{0,1}}
```

Subroutine 3 $\mathbf{I}'_u \leftarrow$ *DOD-DSSE*.Dec$(\mathbf{I}_u, u, b, K)$

Decrypt a row/column $\mathbf{I}_u$ using its address u, server ID b, and master key $K = (k_0, k_1)$:

```
1:  if u is a row index then
2:      τ_u^(b) ← KDF(k_b||u||C_r^(b)[u])
3:      for j = 1..., 2N do
4:          I'_u[j] ← E.Dec_{τ_u^(b)}(I_u[j], j||C_c^(b)[j])
5:  else        # if u is a column index
6:      for i = 1..., 2N do
7:          τ_i^(b) ← KDF(k_b||i||C_r^(b)[i])
8:          I'_u[i] ← E.Dec_{τ_i^(b)}(I_u[i], u||C_c^(b)[u])
9:  return I'_u
```

Subroutine 4 $T \leftarrow$ *DOD-DSSE*.UpdateT$(T, qIdx, b, nIdx)$

Update item's address on server $S_{\neg b}$ by $nIdx$ using its address $qidx$ on server S_b for hash table lookup:

```
    # Get hash table entry for qIdx in S_b
1:  j_x ← T.lookup(qIdx, b)
    # Update hash table with new entry nIdx and server b
2:  oIdx ← T[j].u_x^(¬b)
3:  T[j_x].u_x^(¬b) ← nIdx
4:  T[j_x].b_x ← ¬b
    # Remove nIdx from dummy set T.L_¬b and add oIdx to it
5:  T.L_¬b ← T.L_¬b ∪ {oIdx} \ {nIdx}
6:  return T
```

4.3 Security Analysis

Let $(\overrightarrow{\sigma_0}, \overrightarrow{\sigma_1})$ be a query sequence of length q sent to servers (S_0, S_1) respectively. By Definition 1, access patterns $\langle \mathbf{AP}_0(\overrightarrow{\sigma_0}), \mathbf{AP}_1(\overrightarrow{\sigma_1})\rangle$ observed by (S_0, S_1), respectively, are:

$$\mathbf{AP}_0 = \{\mathsf{access}(x_1^{(0)}), \ldots, \mathsf{access}(x_i^{(0)}), \ldots, \mathsf{access}(x_q^0)\}$$
$$\mathbf{AP}_1 = \{\mathsf{access}(x_1^{(1)}), \ldots, \mathsf{access}(x_i^{(1)}), \ldots, \mathsf{access}(x_q^{(1)})\}, \quad (1)$$

$\mathsf{access}(x_i^{(k)}) = (\{\mathsf{read}(u_{j,ti}^{(k)}, \mathbf{I}_{u_{j,ti}}^{(k)})\}, \{\mathsf{write}(u_{j,ti}^{(k)}, \hat{\mathbf{I}}_{u_{j,ti}}^{(k)})\})$, for $j \in \{w, f\}$, $1 \le t \le 2$, and $1 \le i \le q$ performing read-then-write operations on the server S_b, given a *DOD-DSSE*.Access operation op_i (Algorithm 3) at step i. Each address-data tuple $(u_{j,ti}^{(k)}, \mathbf{I}_{u_{j,ti}}^{(k)})$ comprises a random row or column address and an IND-CPA encryption output, respectively.

REMARK 1. *Due to the properties of the square incidence matrix data structure, rows and columns intersect each other and have the same length. For each actual operation, DOD-DSSE performs a search query and an update query to each server S_b. This prevents S_b from determining (i) if the actual intention of the client is to search or to update, and (ii) which data-address tuple corresponds with search or update query. These properties prevent S_b from separately forming search and update patterns as shown in Section 1.1.*

REMARK 2. *Data items associated with search and update queries are located in two independent address spaces (i.e., row index vs. column index) and, therefore, their access operations are independent from each other. For the sake of brevity, we only analyze the security of search queries. The same analysis can be applied to update queries. From now on, whenever we say data $\mathbf{I}_{u_x}$ of the query x at address u_x, we mean the row data corresponding with the search query along with its row index.*

According to the unlinkability definition (Definition 3) and *DOD-DSSE* access scheme in Algorithm 3, we define in Definition 5 the unlinkability property of a data item which is read from server $S_{\neg b}$, and its new representation is then written to S_b under S_b's view. We then show in Lemma 1 that any access patterns observed by servers (S_0, S_1) in our scheme are unlinkable to each other by Definition 5 under Assumption 1. Finally, we prove that *DOD-DSSE* achieves our main security notion (Definition 4) in Theorem 1.

DEFINITION 5. *Let $(u_x^{(b)}, \mathbf{I}_{u_x}^{(b)})$ represent an item x in a set $\mathcal{I}^{(b)}$ of N distinct data items on server S_b such that $u_x^{(b)} \neq u_{x'}^{(b)}$ and $\mathbf{I}_{u_x}^{(b)} \neq \mathbf{I}_{u_{x'}}^{(b)}$, for each $x', x \in \mathcal{I}^{(b)}$ and $x \neq x'$. $(\tilde{u}_{x''}^{(b)}, \hat{\mathbf{I}}_{\tilde{u}_{x''}}^{(b)}) \in \mathbf{AP}_b$ (as in (1)) is a new representation of an arbitrary data item $x'' \in \mathcal{D}$, which has just been accessed on server $S_{\neg b}$. In DOD-DSSE, $(\tilde{u}_{x''}^{(b)}, \hat{\mathbf{I}}_{\tilde{u}_{x''}}^{(b)})$ is unlinkable to $\mathcal{I}^{(b)}$ if and only if the probability that $(\tilde{u}_{x''}^{(b)}, \hat{\mathbf{I}}_{\tilde{u}_{x''}}^{(b)})$ represents the same item with any tuples $(u_x^{(b)}, \mathbf{I}_{u_x}^{(b)})$ for each $x \in \mathcal{I}^{(b)}$ is $\frac{1}{N}$.*

LEMMA 1. *Under Assumption 1, any access patterns observed by S_b and $S_{\neg b}$ as in (1) are unlinkable with each other by Definition 3.*

PROOF. For each *DOD-DSSE* operation x_i (Algorithm 3), server S_b observes that two address-data tuples are accessed per search query simultaneously. One of them is to

read while the other is to write data being read from $S_{\neg b}$. The data from all accessed addresses are IND-CPA re-encrypted with new counters before being written back (Algorithm 3, steps 17, 20) so that it is computationally indistinguishable for S_b to determine which address is being read or written. To begin with, we show that $\mathsf{access}(x_i^{(b)})$ is unlinkable to $\mathsf{access}(x_i^{(\neg b)})$ as follows:

We first analyze the address-data tuple denoted as $(u^{(b)}, \mathbf{I}_u^{(b)})$, which is read and observed by S_b. $\mathbf{I}_u^{(b)}$ is decrypted into $\mathbf{I'}_u^{(b)}$ and then is IND-CPA re-encrypted with a new counter before being written to $S_{\neg b}$ (steps 17, 20). $\mathbf{I}'$ is assigned by the client to a new random index selected from a set of dummy addresses in $S_{\neg b}$ as $u^{(\neg b)} \xleftarrow{\$} T_w.\mathcal{L}_{\neg b}$, which is independent from $u^{(b)}$. Under Assumption 1, S_b does not have a view on $S_{\neg b}$ and vice versa. So, S_b does not know if $\mathbf{I}'$ is assigned to which $u^{(\neg b)}$ in $S_{\neg b}$ and under which new encryption form $\hat{\mathbf{I}}$. Therefore, $(u^{(b)}, \mathbf{I}_u^{(b)})$ can represent the same item with any address-data tuples $(u^{(\neg b)}, \mathbf{I}_u^{(\neg b)})$ in $S_{\neg b}$ with the same probability of $\frac{1}{N}$, where N is the number of items in $S_{\neg b}$. By Definition 5, $(u^{(b)}, \mathbf{I}_u^{(b)})$ is unlinkable to any items in $S_{\neg b}$ from S_b's view. Considering the $S_{\neg b}$'s view, $S_{\neg b}$ also does not know which address-data tuple $(u^{(b)}, \mathbf{I}_u^{(b)})$ was read from S_b under Assumption 1. Meanwhile, $\hat{\mathbf{I}}$ is a IND-CPA encryption so that it looks random-uniform to all other data in $S_{\neg b}$. Moreover, the address associating with $\hat{\mathbf{I}}$ is selected randomly from the set of dummy addresses $\mathcal{L}_{\neg b}$ with $|\mathcal{L}_{\neg b}| = N$. It is oblivious for $S_{\neg b}$ to link $\hat{\mathbf{I}}$ to any item which will be queried subsequently. Notice that to achieve this obliviousness, it is mandatory to always keep $|\mathcal{L}_{\neg b}| = N$. Once the new IND-CPA encryption form $\hat{\mathbf{I}}$ of an item is written to new address $u^{(\neg b)}$ in $S_{\neg b}$, its old address in $S_{\neg b}$ will be set to dummy and included to $\mathcal{L}_{\neg b}$ by the client (Subroutine 4, steps 2, 5).

We next consider search address-data tuple denoted as $(\widetilde{u}^{(b)}, \hat{\mathbf{I}}_{\widetilde{u}}^{(b)})$ which is written to S_b under S_b's view. This tuple is the new representation of an arbitrary item which has just been queried from $S_{\neg b}$. As *DOD-DSSE* access operations on servers S_b and $S_{\neg b}$ are symmetric, meaning that S_b can act as $S_{\neg b}$ in the aforementioned analysis and vice versa. Therefore, the same analysis is applied to this case.

Finally, we show that if each pair $(\mathsf{access}(x_i^{(b)}), \mathsf{access}(x_i^{(\neg b)}))$ is pairwise unlinkable to each other, for all $1 \leq i \leq q$, then $\mathsf{access}(x_i^{(b)})$ is also unlinkable to others $\mathsf{access}(x_j^{(\neg b)})$ for all $1 \leq j \neq i \leq q$. Without loss of generality, we assume that $j < i$. As $(\mathsf{access}(x_j^{(b)}), \mathsf{access}(x_j^{(\neg b)}))$ is pairwise unlinkable, meaning that given $\mathsf{access}(x_j^{(b)})$ observed by S_b, the corresponding $\mathsf{access}(x_j^{(\neg b)})$ generated in $S_{\neg b}$ is oblivious from S_b's view. It is computationally infeasible for S_b to link $\mathsf{access}(x_i^{(b)})$ with any access patterns $\mathsf{access}(x_j^{(\neg b)})$ generated in $S_{\neg b}$ by just observing $\mathsf{access}(x_j^{(b)})$. The same principle applies to $S_{\neg b}$ as operations on two servers are symmetric. Hence, the Lemma 1 holds. □

COROLLARY 1. *For any access pattern observed by S_b (or $S_{\neg b}$), the same query will result in the same address being accessed on S_b (or $S_{\neg b}$), at most twice.*

PROOF. Assume that at step i the real query x is performed and its corresponding data item is read from address $u_x^{(b)}$ in server S_b. According to *DOD-DSSE* scheme, data of x will be written to an arbitrary address $u_x^{(\neg b)}$ in $S_{\neg b}$ (Algorithm 3, step 21). Given that the same query x is performed again at step $j > i$, its data will be read from $S_{\neg b}$ so that $S_{\neg b}$ can observe the address $u_x^{(\neg b)}$ accessed at step i is now accessed again. By this access pattern, $S_{\neg b}$ can infer the same query is performed at step i and j on it. However from S_b's view, it is oblivious for S_b to determine if the data being written to an arbitrary address $\widetilde{u}_x^{((b))}$ at step j is associated with the query x at step i due to Lemma 1. Now assume that at step k, where $k > j > i$, the query x is performed again. It will be read from S_b and then written to $S_{\neg b}$. Similar to that of S_b at step j, $S_{\neg b}$ observes an access operation which is unlinkable to access operations generated at step j and i by Lemma 1. It can be easily seen that the same query only generates the same address access at most two times. Therefore, the corollary holds. □

THEOREM 1. *Given a server S_b, DOD-DSSE achieves security Definition 4, meaning that DOD-DSSE leaks no information beyond ORAM security definition with the exception of one-time repetition of an unlinkable query on the other $S_{\neg b}$, to which S_b does not have access.*

PROOF. Given an access pattern $\mathbf{AP}_b$ of length q as in (1) observed by server S_b, denote $\mathcal{Y}_q$ as the set of all possible combinations y of N data items, where $|y| = q$. We have N^q possible strings as $|\mathcal{Y}_q| = N^q$. Let $u_{i,1}^{(b)}, u_{i,2}^{(b)}$ be read and write addresses for search query observed by S_b, respectively, given a data access request $\mathsf{access}(x_i^{(b)})$. From S_b's view, data item of the real query x_i (denoted as $\mathbf{I}'_{x_i}$) can be accessed from $u_{i,1}^{(b)}$, or $u_{i,2}^{(b)}$ (i.e., $\mathbf{I}'_{x_i}$ is actually accessed from $S_{\neg b}$ and then written to S_b). By Lemma 1, we have:

$$\Pr(\mathsf{access}(x_i^{(b)})) = \sum_{j=1}^{2} \frac{1}{2} \Big[\Pr(\mathsf{pos}(\mathbf{I}'_{x_i}) = u_{i,j}^{(b)}) \Pr(u_{i,j}^{(b)} \in T_w.\mathcal{L}_b)$$

$$+ \Pr(\mathsf{pos}(\mathbf{I}'_{x_i}) = u_{i,j}^{(b)}) \Pr(u_{i,j}^{(b)} \notin T_w.\mathcal{L}_b) \Big] = \sum_{j=1}^{2} \frac{1}{2} \left[\frac{1}{N} \frac{1}{2} + \frac{1}{N} \frac{1}{2} \right] = \frac{1}{N}.$$

Notice that given a real query x_i, Algorithm 3 generates two random row indices on each server S_b: one is from the set of dummy addresses and the other is from from the set of non-empty addresses. Such addresses are removed from their current set and included in the other set. This is to maintain the size of each set so that given another real query $x_j \neq x_i$, its generated addresses are randomly chosen from size-consistent sets, making it independent of each other. Hence, from S_b's observation, access patterns generated by $\mathsf{access}(x_i^{(b)})$ are computational indistinguishable from those generated by $\mathsf{access}(x_j^{(b)})$, given that $x_j \neq x_i$.

For $x_j = x_i$, with $q \geq j > i \geq 1$, we have two cases:

(i) If $\mathsf{pos}(\mathbf{I}'_{x_i}) = u_{i,1}^{(b)}$ then $\mathsf{pos}(\mathbf{I}'_{x_j}) = u_{j,2}^{(b)}$, meaning $\mathbf{I}'_{x_i}$ is read from S_b, while $\mathbf{I}'_{x_j}$ is read from $S_{\neg b}$. By Lemma 1, data in $u_{j,2}$ is unlinkable to any data items in S_b. Therefore, $\mathsf{access}(x_j^{(b)})$ generates an access pattern which is statistically independent from $\mathsf{access}^{(}x_i^{(b)})$ in server S_b.

(ii) If $\mathsf{pos}(\mathbf{I}'_{x_i}) = u_{i,2}^{(b)}$ then $\mathsf{pos}(\mathbf{I}'_{x_j}) = u_{j,1}^{(b)}$, meaning $\mathbf{I}_{x_i}$ is read from $S_{\neg b}$ and $\mathbf{I}'_{x_j}$ is read from S_b. S_b observes that the same address is accessed again (i.e., $u_{j,1}^{(b)} = u_{i,2}^{(b)}$). By Lemma 1, data from $u_{i,2}^{(b)}$ is written to $S_{\neg b}$ which is unlinkable to any data items in $S_{\neg b}$; the probability that S_b can determine if $\mathbf{I}'_{x_j}$ is re-written back to it

in subsequent access operations is $\frac{1}{N}$. That means given another query x_k such that $x_k = x_j = x_i$, with $k > j > i$, the access pattern generated by $\mathsf{access}(x_k^{(b)})$ is statistically independent from $\mathsf{access}(x_j^{(b)})$ as in case (i). Therefore, the information *DOD-DSSE* leaks in this case is that the same query can generate the same address access at most twice, as shown in Corollary 1.

To sum up, we have $\Pr(\mathbf{AP}_b) = \prod_{j=1}^{q} \Pr(\mathsf{access}(x_j^{(b)})) = \left(\frac{1}{N}\right)^{(q-r)}$, where r is the number of one-time repetitions of data access requests as in case (ii). There are $N^{(t-r)}$ possible strings $y \in \mathcal{Y}_q$ that can generate the same $\mathbf{AP}_b$ so that it is computationally indistinguishable for S_b to determine which string in $N^{(t-r)}$ candidates generates the $\mathbf{AP}_b$. In the worst case, where there are $r = q/2$ repetitions in the data request sequence of length q, then $\Pr(\mathbf{AP}_b) = \left(\frac{1}{N}\right)^{q/2}$. □

Security against statistical attacks. In traditional DSSE, each search or update query on a keyword or a file produces the same address being accessed for consistency purposes. This deterministic relation between queries and address tokens permits an adversary to perform statistical attacks, such as query frequency analysis, to uncover the relations among keyword/file being accessed [15, 3, 27, 11].

In *DOD-DSSE*, queries observed in each server are unlinkable by Definition 3 to each other, meaning that they can be independently generated by any possible keywords/files, from the server's view. *DOD-DSSE* achieves the security by Definition 4 in that, the only information that server S_b can infer from its observed access pattern is one-time repetition of an arbitrary query, which is previously performed on the other server $S_{\neg b}$. Note that servers do not have a view of each other's accesses or queries (i.e., non-colluding servers). This leakage is negligible for any practical setting, and does not permit to establish any statistical relationship, since one-time repetitions are unlinkable. These security guarantees imply that *DOD-DSSE* can not only prevent statistical analysis (e.g., [15]) but also any other potential threats that may exploit the linkability among arbitrary queries.

5. PERFORMANCE EVALUATION

We evaluated the performance of our scheme on real network settings with different network latencies. By latency, we mean the round-trip time taken by a packet to go from the host (i.e., client) to the destination (i.e., server) and back. In addition, we made several comparisons. First, we compared our scheme's cryptographic end-to-end delay (i.e., the time to completely process a search or update operation) with a traditional DSSE scheme which does not hide the access pattern (e.g., [26]). We then compared to a *simulated* scheme which applies ORAM on a DSSE dictionary data structure and matrix data structure[4]. We notice that in order to be comparable with ORAM, which can achieve oblivious operations (i.e., whether the operation is search or update), our *DOD-DSSE* scheme is designed to always perform *both* search and update queries regardless of the type of actual operation which is required. Therefore, search and update operations require the same amount of time. This is in contrast with traditional DSSE schemes in which search and update operations incur different delays. This will be shown in the following experiments.

Hardware setting and configuration. We used a HP Z230 Desktop as the client and two virtual servers provided by Amazon EC2. The client machine was installed with CentOS 7.2 and equipped with Intel Xeon CPU E3-1231v3 @ 3.40GHz, 16 GB RAM and 256 GB SSD. We deployed servers running Ubuntu 14.04 with `m4.4xlarge` instance type which offers 16 vCPUs @ 2.4 GHz, Intel Xeon E5-2676v3, 64 GB RAM and 200 GB SSD for each server.

We adopted Google sparse hash table[5] to implement the data structures T_f, T_w stored at the client side. We implemented IND-CPA encryption and decryption schemes using AES-CTR mode as it supports parallelism and key pre-computation. We used AES-128 CMAC to implement the hash function H. For cryptographic primitives, we utilized libtomcrypt[6] with Intel AES-NI hardware accelerated library[7] to optimize the performance of cryptographic operations. We used ZeroMQ library[8] to implement network communication between client and server(s).

Dataset. We performed our experiments on the Enron email dataset[9]. We selected subsets of the Enron corpus to construct the DSSE data structure with various combinations of keyword-file pairs, ranging from 10^8 to 9×10^{10}. This is to evaluate the performance of *DOD-DSSE* and its counterparts with different dataset sizes starting from small to very large similar to [4].

Results and Comparison. We first measured the preprocessing time to build the encrypted data structures in *DOD-DSSE* with different sizes. With the largest data structure being experimented, which consists of 9×10^{11} keyword-file pairs (i.e., 300,000 files and 300,000 keywords), it takes the client roughly 20 hours to construct two encrypted incidence matrices. For 10^8 keyword-file pairs, the time is 30 seconds. Notice that this initialization phase is only run one time in the offline phase so that its cost is not an important factor. Our focus is to evaluate the performance of *DOD-DSSE* and its counterparts in the online phase, where we perform search and update operations on the constructed data structure(s).

Next, we showed the performance of *DOD-DSSE* scheme in the online phase. We created two Amazon EC2 servers in the same geographical region (i.e., in-state), resulting in an average network latency of 11 ms and throughput of 100 Mbps. It takes approximately 800 ms to perform a search (or update) operation on the distributed data structure consisting of 9×10^{11} keyword-file pairs, as demonstrated in Figure 4b.

We compared the actual cost of *DOD-DSSE* with that of our counterparts. We selected the scheme in [26] to be our main traditional DSSE counterpart as, to the best of our knowledge, it is the most secure DSSE scheme in the literature. We used Path ORAM [24] protocol for ODS as it offers optimal bandwidth overhead. We simulated the cost of using ODS on a dictionary (denoted as ODICT) and a square incidence matrix (denoted as OMAT) data structures

[4]We did not implement the full scheme, we estimated the performance by simulation in a real network setting and assuming a 4 KB block-size ORAM as presented in [25, 24].

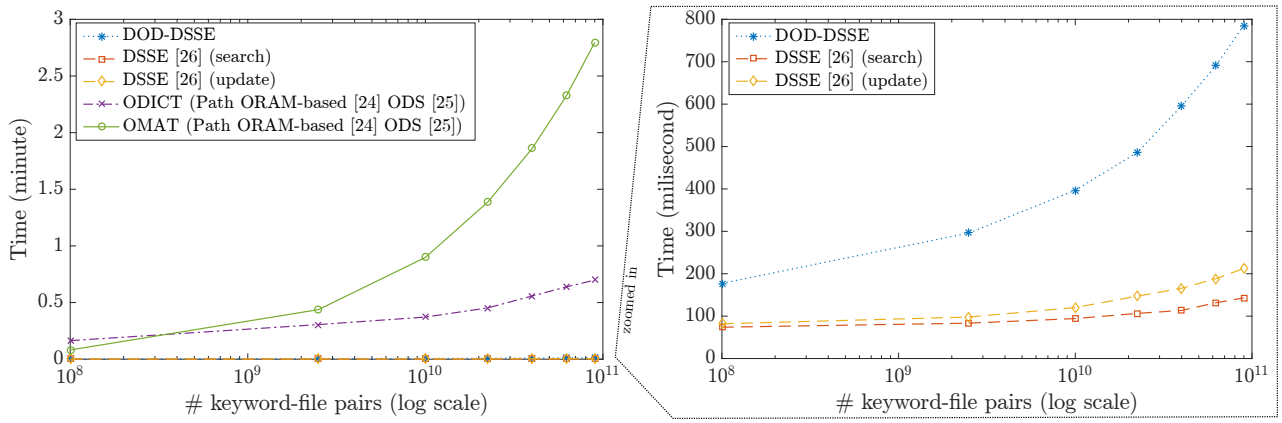
[5]https://github.com/sparsehash/sparsehash

[6]http://www.libtom.org/LibTomCrypt/

[7]https://software.intel.com/articles/download-the-intel-aesni-sample-library

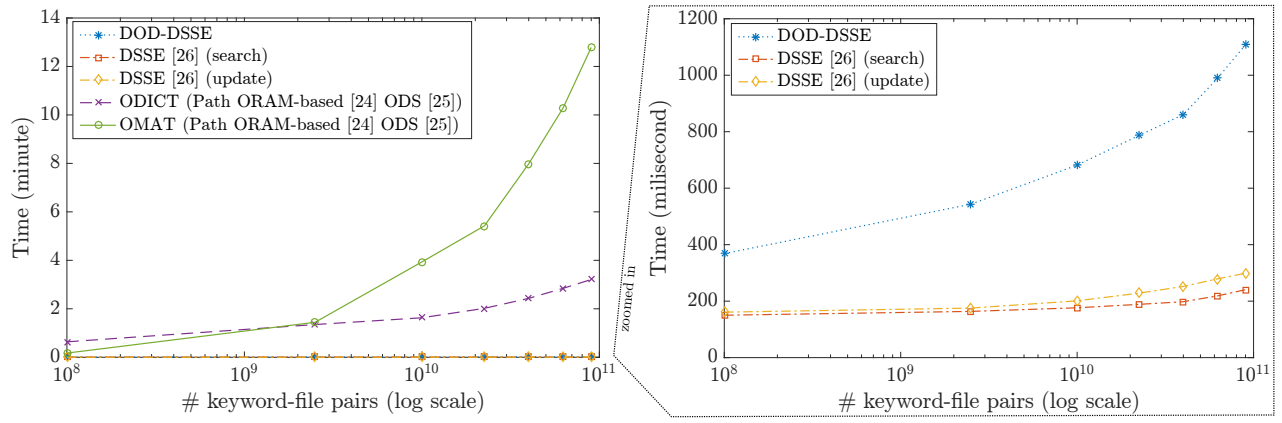
[8]http://zeromq.org/

[9]https://www.cs.cmu.edu/∼./enron/



(a) $DOD - DSSE$ vs. ODS and traditional DSSE (b) $DOD - DSSE$ vs. traditional DSSE

Figure 4: End-to-end cryptographic delay with in-state network latency, where (b) is zoomed in view of *DOD-DSSE* and traditional DSSE, which is hard to be observed in (a).



(a) $DOD - DSSE$ vs. ODS and traditional DSSE (b) $DOD - DSSE$ vs. traditional DSSE

Figure 5: End-to-end cryptographic delay with out-state network setting, where (b) is zoomed in view of *DOD-DSSE* and traditional DSSE, which is hard to be observed in (a).

because the former provides sublinear operating time while the latter achieves the best security. We applied an average padding strategy to mitigate the information leakage from ODICT due to the optimal search/update time property.

We analyzed the asymptotic communication complexity of the aforementioned schemes. Traditional DSSE achieves optimal bandwidth overhead of $O(r)$, where r is the number of data corresponding with the search/update query [26]. In ODS approaches, keyword-files pairs are packaged into 4KB blocks, and the total number of blocks is denoted as B. Given a search/update operation, the number of blocks being transmitted by OMAT and ODICT is $s \cdot c \cdot O(\log B)$ and $s' \cdot c \cdot O(\log B)$, respectively, where $c = 4$ is the bucket size in Path ORAM [24] and s, s' are the numbers of communication rounds to retrieve sufficient results for the query [25]. In *DOD-DSSE*, the bandwidth complexity is $4 \cdot O(N)$, where N is the maximum number of unique keywords/files that *DOD-DSSE* can support.

After that, we benchmarked the actual performance of *DOD-DSSE* and its counterparts in practice based on the previous asymptotic communication analysis. Figures 4a, 4b demonstrate the actual end-to-end cryptographic delay (i.e., encryption, transmission delays) of schemes using in-state Amazon EC2 server(s) with various data structure sizes. We can see that *DOD-DSSE* incurs a small-constant communication overhead due to extra queries (i.e., 4× times slower than traditional DSSE). However, it is *approximately 50× and 210× times faster than ODICT and OMAT* which specifically take 42 and 167 seconds to perform an operation on the large data structure, respectively. This indicated that even though the asymptotic complexity of ODS approaches looks very efficient, hidden constants such as c, s, s' actually contribute a lot to the communication overhead, as shown in Table 1 and Figure 4a.

Table 2: Total size of encrypted data structure(s) in GB.

# keyword-file pairs	*DOD-DSSE*†	DSSE [26]	ODICT	OMAT
10^8	0.1	0.02	1.37	0.05
2.5×10^9	2.4	0.6	37.38	1.16
10^{10}	10	2.4	149.54	4.66
2.25×10^{10}	22.4	5.6	336.46	10.48
4×10^{10}	40	10	598.15	18.63
6.25×10^{10}	62.4	15.6	934.60	29.10
9×10^{10}	90	22.4	1345.83	41.91

† *DOD-DSSE* stores two encrypted data structures in two non-colluding servers so that the storage cost for each server will be a half of presented numbers.

We inspected the cost of *DOD-DSSE* to investigate the impacts of network communication and cryptographic operations on the end-to-end delay. We observed that the majority of the delay is due to network transmission, in which the ratio between it and cryptographic operations is roughly 8:1. To investigate more the impact of network latency and throughput on *DOD-DSSE* and its counterparts, we setup two EC2 servers geographically located outside of our state, resulting in a network latency and throughput of 31 ms and 30 Mbps, respectively. As it can be seen in Figure 5b, this geographically distributed out-state environment makes *DOD-DSSE* and traditional DSSE perform approximately 200 ms and 100 ms slower than in-state setting, respectively. Due to the characteristics of ODS requiring a number of communication rounds to perform a search or update operation, slower network latency and throughput significantly impact the performance of ODICT and OMAT. We can see that *DOD-DSSE* is now 170x and 690x times faster than ODICT and OMAT, respectively by this setting. Comparing with the in-state configuration, ODICT and OMAT are both 4.5x times slower than their in-state version. This implies 12.78 minutes and 3.2 minutes to accomplish an operation, respectively (Figure 5a).

Finally, we analyzed the storage cost of *DOD-DSSE*. With the largest dataset being experimented in this study (i.e., 9×10^{11} keyword-file pairs), *DOD-DSSE* requires approximately 35 MB to store at the client side all necessary information for its operation such as symmetric keys, T_f, T_w and global counter arrays. This can be easily fulfilled even by resource-limited devices such as a smartphone or a tablet. Tables 2 shows the total size of the encrypted data structure(s) stored at the server side required by *DOD-DSSE* and its counterparts with different dataset sizes. *DOD-DSSE* requires 8x and 2x times as much storage space as that of traditional DSSE and OMAT, respectively, and yet, is much more compact than ODICT using dictionary[10]. Considering the advantages of *DOD-DSSE* in terms of efficiency, storage cost and achieved security aspects over traditional DSSE and ORAM-based methods, our scheme is likely to be an ideal security-performance trade-off DSSE scheme for privacy-critical cloud computing.

6. RELATED WORK

Searchable Encryption (SE). The first SE was proposed by Song et al. in [20] and was followed by several schemes which can search only on static file collections (e.g., [6, 8]). Kamara et al. were among the first to develop a DSSE scheme [13], followed by several DSSE schemes that offer various performance and security properties such as small leakage [21, 26], scalable searches with extended query types [5, 4] and high efficiency [17].

All DSSE schemes leak data structure-access pattern including search pattern and update pattern (see Section 1.1). Liu et al. in [15] showed that the search pattern reveals significant information about the queried keywords. Bosch et al. in [2] developed a search pattern hiding DSSE, which requires re-encrypting and transmitting entire encrypted data structure per search query and, therefore, seems to be impractical. Islam et al. in [11] showed that, with some prior knowledge on the keyword/file pairs, an adversary can learn significant information about the queries and keywords from the access pattern. The update pattern leaks information during updates depending on the type of data structure used and other security-performance trade-offs [26]: The most efficient DSSE [13] achieves the least security. The schemes in [12] and [4] achieved higher privacy. Recently, Yavuz and Guajardo [26] proposed a scheme that leaks less information.

PIR. Private Information Retrieval (PIR) is the task of retrieving a data item from a *public* (unencrypted) database server without revealing to the server the specific item that has been accessed (e.g., [7, 9]). In contrast, in our setting, the DSSE data structure is encrypted (rather than public) and it is private to each user. In both settings one also has distributed (multi-server) versions of the basic protocols.

ORAM. ORAM allows clients to perform arbitrary queries to an outsourced database without leaking any access pattern. Therefore, it can be used to obliviously access encrypted data structure in DSSE. Preliminary ORAM schemes [18] were very costly, but recent progress in ORAM constructions (e.g., [23, 19, 24, 25, 22]) are promising. The most efficient and popular ORAM scheme is Path ORAM [24], which offers optimal bandwidth/processing cost . Despite these improvements, as indicated by recent studies [21, 4, 16, 1], ORAM is not practical for searchable encryption as it introduces large bandwidth overhead and high delay.

ODS. ODS is an instantiation of position-based ORAM (e.g., Path ORAM) specifically designed for oblivious access on data structure [25]. Similar to recursive ORAMs, ODS reduces the client storage cost but increases the number of communication rounds and bandwidth overhead. Hence, ODS is also not practical for large data structure in DSSE.

Multi-cloud Oblivious Storage. Stefanov et al. in [22] proposed an ORAM scheme using two non-colluding servers to reduce the client bandwidth cost. This approach differs from us in that two servers are required to perform computation and communication with each other. We only use servers as basic storage units so that they are only required to have functionalities to transfer and receive data being requested.

7. CONCLUSIONS

In this paper, we developed a new oblivious access scheme over the encrypted data structure(s) for searchable encryption purposes that we refer to as *DOD-DSSE*. *DOD-DSSE* achieves high security yet practical encrypted search/update operations simultaneously. That is, *DOD-DSSE* seals critical information leakage from the data structure access pattern by guaranteeing query unlinkability and, therefore, it offers a much higher security than traditional DSSE schemes. At the same time, *DOD-DSSE* performs two orders of magnitude faster than using ORAM-based techniques (e.g., ODS with Path ORAM), for search/update operations on the encrypted data structure. These properties make *DOD-DSSE* an ideal alternative for oblivious yet practical searchable encryption on privacy-critical cloud computing applications.

8. ACKNOWLEDGMENTS

We would like to thank Gabriel Hackebeil, Daniel Lin as well as anonymous reviewers for their insightful comments and suggestions to improve the quality of the paper.

[10]We would like to notice an advantage of dictionary over matrix structure. That is, it is not limited by the number of unique keywords and files, but only the maximum number of keyword-file pairs which might be useful for applications requiring diverse keyword-file relations.

9. REFERENCES

[1] V. Bindschaedler, M. Naveed, X. Pan, X. Wang, and Y. Huang. Practicing oblivious access on cloud storage: the gap, the fallacy, and the new way forward. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 837–849. ACM, 2015.

[2] C. Bosch, 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 Annual International Conference on*, pages 330–337. IEEE, 2014.

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

[4] D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. *IACR Cryptology ePrint Archive*, 2014:853, 2014.

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

[6] Y.-C. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *Applied Cryptography and Network Security*, pages 442–455. Springer, 2005.

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

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

[9] I. Goldberg. Improving the robustness of private information retrieval. In *2007 IEEE Symposium on Security and Privacy (SP'07)*, pages 131–148. IEEE, 2007.

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

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

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

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

[14] J. Katz and Y. Lindell. *Introduction to modern cryptography*. CRC Press, 2014.

[15] C. Liu, L. Zhu, M. Wang, and Y.-a. Tan. Search pattern leakage in searchable encryption: Attacks and new construction. *Information Sciences*, 265:176–188, 2014.

[16] M. Naveed. The fallacy of composition of oblivious ram and searchable encryption. Technical report, Cryptology ePrint Archive, Report 2015/668, 2015.

[17] M. Naveed, M. Prabhakaran, and C. A. Gunter. Dynamic searchable encryption via blind storage. In *Security and Privacy (S&P), 2014 IEEE Symposium on*, pages 639–654. IEEE, 2014.

[18] B. Pinkas and T. Reinman. Oblivious ram revisited. In *Advances in Cryptology–CRYPTO 2010*, pages 502–519. Springer, 2010.

[19] L. Ren, C. Fletcher, A. Kwon, E. Stefanov, E. Shi, M. Van Dijk, and S. Devadas. Constants count: practical improvements to oblivious ram. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 415–430, 2015.

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

[21] E. Stefanov, C. Papamanthou, and E. Shi. Practical dynamic searchable encryption with small leakage. In *Annual Network and Distributed System Security Symposium – NDSS*, volume 14, pages 23–26, 2014.

[22] E. Stefanov and E. Shi. Multi-cloud oblivious storage. In *Proceedings of the 2013 ACM SIGSAC conference on Computer and Communications Security*, pages 247–258. ACM, 2013.

[23] E. Stefanov, E. Shi, and D. Song. Towards practical oblivious ram. *arXiv preprint arXiv:1106.3652*, 2011.

[24] E. Stefanov, M. Van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas. Path oram: an extremely simple oblivious ram protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer and Communications security*, pages 299–310. ACM, 2013.

[25] X. S. Wang, K. Nayak, C. Liu, T. Chan, E. Shi, E. Stefanov, and Y. Huang. Oblivious data structures. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 215–226. ACM, 2014.

[26] A. A. Yavuz and J. Guajardo. Dynamic searchable symmetric encryption with minimal leakage and efficient updates on commodity hardware. In *Selected Areas in Cryptography – SAC 2015*, Lecture Notes in Computer Science. Springer International Publishing, August 2015.

[27] Y. Zhang, J. Katz, and C. Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption.