Oblivious Dynamic Searchable Encryption on Distributed Cloud Systems

Thang Hoang[1], Attila A. Yavuz[1], F. Betül Durak[2], and Jorge Guajardo[3]

[1] EECS, Oregon State University, Corvallis, Oregon, USA
{hoangmin,attila.yavuz}@oregonstate.edu
[2] École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland
betul.durak@epfl.ch
[3] Robert Bosch RTC—LLC, Pittsburgh, PA, USA
jorge.guajardomerchan@us.bosch.com

Abstract. Dynamic Searchable Symmetric Encryption (DSSE) allows search/update operations over encrypted data via an encrypted index. However, DSSE has been shown to be vulnerable to statistical inference attacks, which can extract a significant amount of information from access patterns on encrypted index and files. While generic Oblivious Random Access Machine (ORAM) can hide access patterns, it has been shown to be extremely costly to be directly used in DSSE setting.
By exploiting the distributed cloud infrastructure, we develop a series of Oblivious Distributed DSSE schemes called ODSE, which enable oblivious access on the encrypted index with a high security and improved efficiency over the use of generic ORAM. Specifically, ODSE schemes are 3×-57× faster than applying the state-of-the-art generic ORAMs on encrypted dictionary index in real network settings. One of the proposed ODSE schemes offers desirable security guarantees such as information-theoretic security with robustness against malicious servers. These properties are achieved by exploiting some of the unique characteristics of searchable encryption and encrypted index, which permits us to harness the computation and communication efficiency of multi-server PIR and Write-Only ORAM simultaneously. We fully implemented ODSE and have conducted extensive experiments to assess the performance of our proposed schemes in a real cloud environment.

Keywords: Searchable encryption; Write-only ORAM; Multi-server PIR; Privacy-preserving clouds

1 Introduction

Data outsourcing allows a client to store their data on the cloud to reduce data management and maintenance costs. Despite its merits, cloud services come with severe privacy issues. The client may encrypt their data with standard encryption to protect their privacy. However, these techniques also prevent the client from performing basic operations (e.g., search/update) over the outsourced encrypted data. This significantly degrades the benefits of cloud services. In the following, we first outline the current state-of-the-art techniques and their limitations and then, present our methods towards addressing these challenges.

1.1 State-of-the-art and Limitations

Information leakage in DSSE. The concept of searchable symmetric encryption (SSE) was first proposed by Song et al. [24]. This construction can only search on static encrypted data. Curtmola et al. [11] introduced single-keyword-searched SSE with formal security definition, followed by refinements with extended capabilities such as ranked query [27], multi-keyword search [26] or their combinations [7]. Dynamic Searchable Symmetric Encryption (DSSE) was introduced by Kamara et al. [17], which offers both search and update on encrypted files $\mathcal{F}$ via an encrypted index $\mathbf{I}$ representing keyword-file relationships. Many DSSE schemes have been proposed, each offering various performance, functionality and security trade-offs [4] (e.g., [17,9,6,29,31,20]).

It is known that all standard DSSE schemes leak significant information, which are vulnerable to statistical inference analysis [16,18,8,30]. There are two sources of information leakages in DSSE: (i) leakages through search and update on encrypted index $\mathbf{I}$, (ii) leakages due to access of encrypted files $\mathcal{F}$. Specifically, since the search and update tokens are deterministic, all DSSE schemes leak access patterns on both $\mathbf{I}$ and $\mathcal{F}$. Furthermore, most of them also leak the content of updated files during the update (i.e., forward-privacy) and historical updates (add/delete) on the keyword during the search on $\mathbf{I}$ (i.e., backward-privacy). By exploiting these leakages, recent studies have shown that, sensitive information about encrypted queries and files can be recovered [18,8]. Zhang et al. [30] has presented file-injection attacks that can determine which keywords have been searched, especially in forward-insecure DSSE schemes. Although some DSSE schemes with improved security (e.g., forward and backward privacy) have been proposed (e.g., [6]), they rely on extremely costly public key operations and still leak access patterns. Liu et al. [18] demonstrated an attack that can determine which keywords have been searched by observing the frequency of search queries (search patterns). Zhang et al. [30] has indicated that, *future research on DSSE should focus on sealing information leakages rather than accepting them by default.* Unless these leakages are prevented, a trustworthy deployment of DSSE for privacy-critical applications may remain extremely difficult.

Performance Hurdles of the Existing Approaches to Reduce Information Leakages in DSSE. Several attempts (e.g., [15,5]) are either highly costly or unable to completely seal all leakages in DSSE access patterns. Generic Oblivious Random Access Machine (ORAM) [25][4] can hide access patterns, and therefore, it can prevent most of the information leakages in DSSE. Garg et al. [12] proposed TWORAM scheme, which optimizes the round-trip communication under $\mathcal{O}(1)$ client storage when using ORAM to hide *file access patterns*[5] in DSSE. Despite its merits, prior studies (e.g., [9,21]) stated that generic ORAM (e.g., [25]) is still costly to be used in DSSE due to its logarithmic communication overhead. Although several ORAMs with $\mathcal{O}(1)$ bandwidth complexity have been introduced recently, they are still very costly due to the use of Homomorphic

[4]By generic ORAM, we mean oblivious techniques that can hide operation type (whether it is read or write), as opposed to PIR or Write-Only ORAM.

[5]It differs from the objective of this paper, where we focus on hiding access patterns on the encrypted index in DSSE (see §5 for clarification).

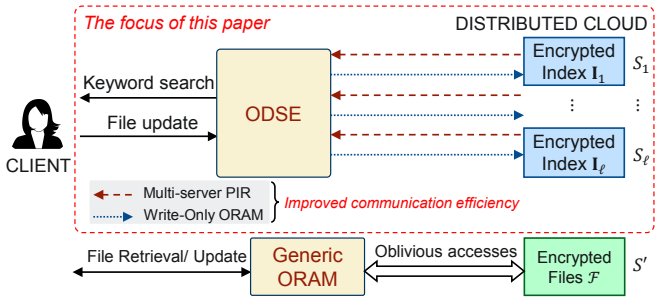


Fig. 1. Our research objective and high-level approach.

Encryption (HE). The performance of such schemes has been shown to be worse than $\mathcal{O}(\log N)$-bandwidth ORAMs [2].

1.2 Our Research Objective and Contributions

It is imperative to seal information leakages from accessing encrypted files $\mathcal{F}$ and encrypted index $\mathbf{I}$. Since the size of individual files in $\mathcal{F}$ might be arbitrarily large and each search/update query might involve a different number of files, to the best of our knowledge, generic ORAM seems to be the only option for oblivious access on $\mathcal{F}$. *The objective of this paper is to design oblivious access techniques on* $\mathbf{I}$*, which are more efficient than using generic ORAM, by exploiting special properties of searchable encryption and* $\mathbf{I}$ *as elaborated in Figure 1.* Particularly, we identify a suitable data structure for $\mathbf{I}$ that allows search and update to operate on separate dimensions. This property permits us to harness communication-efficient techniques such as Write-Only ORAM for update and, by exploiting distributed cloud infrastructure, multi-server PIR for search with low computation overhead. Note that the low communication and computation are important factors in practice since they directly translate into the low end-to-end delay and consequently, improve the quality of services of cloud systems. Notice that the price to pay for such low delay is the collusion vulnerability in the distributed setting, where we assume a limited number of servers that can collude with each other, which is the common adversarial model of multi-server PIR techniques (see §2 and §4).

We propose a series of Oblivious Distributed Encrypted Index $\mathbf{I}$ on the distributed cloud infrastructure with the application on DSSE, which we refer to as ODSE (Figure 1). We present two ODSE schemes called $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$ and $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{it}}$, each offering various desirable performance and security properties as follows.

- *Low end-to-end delay:* ODSE schemes achieve low end-to-end-delay, which are 3×-57×faster than the use of efficient generic ORAMs (e.g.,[25,22]) (with optimization [12]) on encrypted index under real network settings (see §5).
- *Full obliviousness with Information-theoretic security:* ODSE seals information leakages due to accesses on encrypted index $\mathbf{I}$ that lead into statistical attacks such as forward/backward privacy, query types (search/update), hidden size and access patterns. $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$ and $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{it}}$ offer computational and information-theoretic security for $\mathbf{I}$ and operations on it, respectively.

- *Robustness against malicious servers:* $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{it}}$ can tolerate a certain number of malicious servers in the system.
- *Full-fledged implementation and open-sourced framework:* We fully implemented all the proposed ODSE schemes, and evaluated their performance on real-cloud infrastructure. To the best of our knowledge, we are among the first to open-source an oblivious access framework for DSSE encrypted index that can be publicly used for comparison and wide adaptation (see §5).

It is clear that the standard DSSE constructions (e.g., [9]) are much faster, but also less secure than our proposed methods in the sense of leaking more information beyond the access patterns (e.g., forward-privacy, backward-privacy) over the encrypted index. Compared with standard DSSE where access patterns are leaked by default, ODSE schemes offer higher security by sealing all these leakages at the cost of higher latency. Nevertheless, they are more efficient than using generic ORAM techniques atop the DSSE encrypted index to seal such leakages in some certain cases regarding database and query sizes. We provide the detail analysis in §5.

2 Preliminaries and Building Blocks

Notation. We denote $\mathbb{F}_p$ as a finite field where p is a prime. Operators $||$ and $\oplus$ denote the concatenation and XOR, respectively. $(\cdot)_{\mathsf{bin}}$ denotes the binary representation. $\mathbf{u} \cdot \mathbf{v}$ denotes the inner product of two vectors $\mathbf{u}$ and $\mathbf{v}$. $x \xleftarrow{\$} \mathcal{S}$ denotes that x is randomly and uniformly selected from set $\mathcal{S}$. Given I as a row/column of a matrix, $I[i]$ denotes accessing i-th component of I. Given a matrix $\mathbf{I}$, $\mathbf{I}[*, j \dots j']$ denotes accessing columns j to j' of $\mathbf{I}$. Let $\mathcal{E} = (\mathsf{Enc}, \mathsf{Dec}, \mathsf{Gen})$ be an IND-CPA symmetric encryption: $\kappa \leftarrow \mathcal{E}.\mathsf{Gen}(1^\theta)$ generating key with security parameter θ; $C \leftarrow \mathcal{E}.\mathsf{Enc}_\kappa(M)$ encrypting plaintext M with key κ; $M \leftarrow \mathcal{E}.\mathsf{Dec}_\kappa(C)$ decrypting ciphertext C with key κ.

Shamir Secret Sharing (SSS). We present (t, ℓ)-threshold Shamir Secret Sharing (SSS) scheme [23] in Figure 2. Given a secret $\alpha \in \mathbb{F}_p$, the dealer generates a random t-degree polynomial f and evaluates $f(x_i)$ for party $\mathcal{P}_l \in \{\mathcal{P}_1, \dots, \mathcal{P}_\ell\}$, where $x_l \in \mathbb{F}_p \setminus \{0\}$ is the deterministic identifier of $\mathcal{P}_l$. We denote the share for $\mathcal{P}_l$ as $[\![\alpha]\!]_l$. The secret can be reconstructed by combining at least $t+1$ correct shares via Lagrange interpolation. Note that the secret can be recovered from a

$([\![\alpha]\!]_1, \dots, [\![\alpha]\!]_\ell) \leftarrow \mathrm{SSS}.\mathsf{CreateShare}(\alpha, t)$: Create ℓ shares of value α

1: $(a_1, \dots, a_t) \xleftarrow{\$} \mathbb{F}_p$
2: **return** $([\![\alpha]\!]_1, \dots, [\![\alpha]\!]_\ell)$, where $[\![\alpha]\!]_i \leftarrow \alpha + \sum_{u=1}^{t} a_u \cdot x_l^u$ for $1 \le l \le \ell$

$\alpha \leftarrow \mathrm{SSS}.\mathsf{Recover}(\{\mathcal{A}\}, t)$: Recover the value α from its shares

1: Randomly select $t+1$ shares $\{[\![\alpha]\!]_{x_i}\}_{i=1}^{t+1}$ among $\mathcal{A}$
2: $g(x) \leftarrow \mathsf{LagrangeInterpolation}\big(\{(x_i, [\![\alpha]\!]_{x_i})\}_{i=1}^{t+1}\big)$
3: **return** α, where $\alpha \leftarrow g(0)$

Fig. 2. Shamir Secret Sharing (SSS) scheme [23].

number of incorrect shares by error correction techniques (discussed in §4). We use this property to improve the robustness of our protocol in malicious settings.

SSS is t-private so that any combinations of t shares leak no information about the secret. SSS offers homomorphic properties including addition, scalar multiplication, and *partial* multiplication. We extend the notion of share of value to indicate the share of vector: $[\![\mathbf{v}]\!]_i = ([\![v_1]\!]_l, \ldots, [\![v_n]\!]_l)$ denotes the share of vector $\mathbf{v}$ for party $\mathcal{P}_\ell$, in which $[\![v_i]\!]$ is the share of component v_i in $\mathbf{v}$.

Private Information Retrieval (PIR). PIR enables private retrieval of a data item from a (unencrypted) public database server. We recall two efficient multi-server PIR protocols: *(i) XOR-based PIR* [10] (Figure 3) which uses XOR operations and requires each server S_l to store $\mathbf{b}_l$, a replica of database $\mathbf{b}$ containing m blocks $(b_1, \ldots, b_m)$ with the same size; *(ii) SSS-based PIR* [13] (Figure 4), which relies on homomorphic properties of SSS, where each server stores $\mathbf{b}_l$, a replica of the database $\mathbf{b}$ containing m blocks $(b_1, \ldots, b_m)$, where $b_i \in \mathbb{F}_p$.

$b \leftarrow \mathsf{PIR}^{\mathsf{xor}}(x, \langle \mathbf{b}_1, \ldots, \mathbf{b}_\ell \rangle)$: Retrieve a data item indexed x from public database
Client: $e \leftarrow$ 0's string of length m; Set $e[x] \leftarrow 1$
1: $\rho_l \leftarrow$ random binary string of length m for $1 \le l < \ell$; Set $\rho_\ell \leftarrow \rho_1 \oplus \ldots \rho_{\ell-1} \oplus e$
2: Send ρ_i to server S_i for $1 \le i \le \ell$
Server: each S_l storing $\mathbf{b}_l = (b_{l1}, \ldots, b_{lm})$ on receiving ρ_l:
3: $r_l \leftarrow \bigoplus_{j \in \mathcal{J}} b_{lj}$ where $\mathcal{J} = \{j : \rho_l[j] = 1\}$ and send r_l to client
Client: on receiving $(r_1, \ldots, r_\ell)$:
4: **return** $b \leftarrow \bigoplus_{l=1}^{\ell} r_l$

Fig. 3. XOR-based PIR [10].

$b \leftarrow \mathsf{PIR}^{\mathsf{sss}}(x, \langle \mathbf{b}_1, \ldots, \mathbf{b}_\ell \rangle)$: Retrieve a data item indexed x from public database
Client: Let $\mathbf{e} = (e_1, \ldots, e_m)$, where $e_x \leftarrow 1$, $e_i \leftarrow 0$ for $1 \le i \ne x \le n$
1: **for** $i = 1, \ldots, m$ **do**
2: $\quad ([\![e_i]\!]_1, \ldots, [\![e_i]\!]_\ell) \leftarrow \mathrm{SSS.}\mathsf{CreateShare}(e_i, t)$
3: $[\![\mathbf{e}]\!]_l \leftarrow ([\![e_1]\!]_l, \ldots, [\![e_m]\!]_l)$ and send $[\![\mathbf{e}]\!]_l$ to server S_l for $1 \le l \le \ell$
Server: each S_l storing $\mathbf{b}_l = (b_{lm}, \ldots, b_{lm})$ on receiving $[\![\mathbf{e}]\!]_l$:
4: $[\![b]\!]_l \leftarrow [\![\mathbf{e}]\!]_l \cdot \mathbf{b}_l$ and send $[\![b]\!]_i$ to client
Client: on receiving $[\![b]\!]_1, \ldots, [\![b]\!]_\ell$:
5: **return** $b \leftarrow \mathrm{SSS.}\mathsf{Recover}([\![b]\!]_1, \ldots, [\![b]\!]_\ell, t)$

Fig. 4. SSS-based PIR [13].

Write-Only ORAM. ORAM allows the user to hide the access patterns when accessing their encrypted data on the cloud. In contrast to generic ORAM where both read and write operations are hidden, Blass et al. [3] proposed a Write-Only ORAM scheme, which only hides the write pattern in the context of hidden volume encryption. Intuitively, $2n$ memory slots are used to store n blocks, each assigned to a distinct slot and a position map is maintained to keep track of block's location. Given a block to be rewritten, the client reads λ slots chosen uniformly at random and writes the block to a dummy slot among λ slots. Data in all slots are encrypted to hide which slot is updated. By selecting λ sufficiently large (e.g., 80), one can achieve a negligible failure probability, which might occur when all λ slots are non-dummy. It is possible to select a small λ (e.g., 4). In this

case, the client maintains a stash component S of size $\mathcal{O}(\log n)$ to temporarily store blocks that cannot be rewritten when all read slots are full.

3 The Proposed ODSE Schemes

Intuition. In DSSE, keyword search and file update on $\mathbf{I}$ are read-only and write-only operations, respectively. This property permits us to leverage specific bandwidth-efficient oblivious access techniques for each operation such as multi-server PIR (for search) and Write-Only ORAM (for update) rather than using generic ORAM. The second requirement is to identify an appropriate data structure for $\mathbf{I}$ so that the above techniques can be adapted. We found that forward index and inverted index are the ideal choices for the file update and keyword search operations, respectively as proposed in [14]. However, doing search and update on two isolated indexes can cause an inconsistency, which requires the server to perform synchronization. The synchronization operation leaks significant information [14]. To avoid this problem, it is necessary to integrate both search index and update index in an efficient manner. Fortunately, this can be achieved by leveraging a two-dimensional index (i.e., matrix), which allows keyword search and file update to be performed in two separate dimensions without creating any inconsistency at their intersection. This strategy permits us to perform computation-efficient (multi-server) PIR on one dimension, and communication-efficient (Write-Only) ORAM on the other dimension to achieve oblivious search and update, respectively, with a high efficiency.

3.1 ODSE Models and Data Structures

System Model. Our model comprises a client and ℓ servers $\boldsymbol{\mathcal{S}} = (\mathcal{S}_1, \ldots, \mathcal{S}_\ell)$, each storing a version of the encrypted index. In our system, the encrypted files are stored on S', a separate server different from $\boldsymbol{\mathcal{S}}$ (as in [15]), which can be obliviously accessed via a generic ORAM (e.g., [25]). In this paper, we only focus on oblivious access of the encrypted index on $\boldsymbol{\mathcal{S}}$.

Threat Model. In our system, the client is trusted and the servers $\mathcal{S}$ are untrusted. We consider the servers to be semi-honest, meaning that they follow the protocol faithfully, but can record the protocol transcripts to learn information regarding the client's access pattern. However, our system can be easily extended to deal with malicious servers that attempt to tamper the input data to compromise the correctness and the security of the system (see §4). We allow upto $t < \ell$ (privacy parameter) servers among $\mathcal{S}$ to be colluding, meaning that they can share their own recorded protocol transcripts with each other. We present the formal security model in §4.

Data Structures. Assume that the outsourced database can store up to N distinct files and M unique keywords, our index is an incidence matrix $\mathbf{I}$, where each cell $\mathbf{I}[i, j] \in \{0, 1\}$ represents the relationship between the keyword at row i and the file at column j. Each keyword and file is assigned to a unique row and column index, respectively. Each row of $\mathbf{I}$ represents the search result of a keyword while the content (unique keywords) of a file is represented by a column.

Since we use Write-Only ORAM for file update, the number of columns in $\mathbf{I}$ are doubled and a stash S is used to store columns of $\mathbf{I}$ during the update. Therefore, the size of search index $\mathbf{I}$ is $M \times 2N$.

We leverage two static hash tables T_w, T_f as in [28] to keep track of the location of keywords and files in $\mathbf{I}$, respectively. They have the following structure: $T := \langle \mathsf{key}, \mathsf{value} \rangle$, where key is a keyword or file ID and $\mathsf{value} \leftarrow T[\mathsf{key}]$ is the (row/column) index of key in $\mathbf{I}$. Since there are $2N$ columns in $\mathbf{I}$ while only N files, we denote $\mathcal{D}$ as the set of dummy columns that are not assigned to a particular file.

3.2 $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$: Fast ODSE

We introduce $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$ that harnesses XOR-based PIR and Write-Only ORAM to achieve low search and update latency.

Setup. Let Π and Π' be random permutations on $\{1, \ldots, 2N\}$ and $\{1, \ldots, M\}$ respectively. The procedure to setup encrypted index for $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$ is as follows.

$(\mathcal{I}, \sigma) \leftarrow \mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}.\mathsf{Setup}(\mathcal{F})$: Create distributed encrypted index from input files $\mathcal{F}$

1. Initialize a matrix $\mathbf{I}'$ of size $M \times 2N$, Set $\mathbf{I}'[*, *] \leftarrow 0$
2. Extract unique keywords $(w_1, \ldots, w_m)$ from files $\mathcal{F} = \{f_{id_1}, \ldots, f_{id_n}\}$
3. Construct $\mathbf{I}'$ for $i = 1 \ldots, m$ and $j = 1, \ldots, n$:
   (a) $T_f[id_j] \leftarrow \Pi(j)$, $T_w[w_i] \leftarrow \Pi'(i)$, $x \leftarrow \Pi'[w_i]$ and $y \leftarrow \Pi[id_j]$
   (b) If w_i appears in f_{id_j}, set $\mathbf{I}'[x, y] \leftarrow 1$
4. Generate master key as $\kappa \leftarrow \mathsf{Gen}(1^\theta)$
5. Encrypt $\mathbf{I}'$ for $i = 1, \ldots, M$ and $j = 1 \ldots, 2N$:
   (a) $\tau_i \leftarrow \mathsf{KDF}_\kappa(i)$
   (b) $\mathbf{I}[i, j] \leftarrow \mathcal{E}.\mathsf{Enc}_{\tau_i}(\mathbf{I}'[i, j])$
6. Let $\mathcal{D}$ contain column indexes that are not assigned to any file IDs
7. Output $(\mathcal{I}, \sigma)$, where $\mathcal{I} \leftarrow \{\mathbf{I}_1, \ldots, \mathbf{I}_\ell\}$ with $\mathbf{I}_i = \mathbf{I}$ and $\sigma \leftarrow (\kappa, T_w, T_f, \mathbf{c})$

Once $\mathcal{I}$ is constructed, the client sends $\mathbf{I}_i$ to server $\mathcal{S}_i$, and keeps σ as secret.

Search. Intuitively, to search for a keyword w, the client and server execute the XOR-based PIR protocol on the row dimension of $\mathbf{I}$ to privately retrieve the row data of w. Since the row is encrypted *rather than being public as in the traditional PIR model*, the client performs decryption on the retrieved data and filter dummy column indexes to obtain the search result. The detail is as follows.

$\mathcal{R} \leftarrow \mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}.\mathsf{Search}(w, \mathcal{I}, \sigma)$: Search for keyword w

1. Get row index x of the searched keyword w as $x \leftarrow T_w[w]$
2. Execute $\mathbf{I}[x, *] \leftarrow \mathsf{PIR}^{\mathsf{xor}}(x, \langle \mathbf{I}_1, \ldots, \mathbf{I}_\ell \rangle)$ protocol (Figure 3) with ℓ servers:
   (a) Each server inputs its encrypted index $\mathbf{I}_i$, where each row of $\mathbf{I}_i$ is interpreted as an item in the database
   (b) Client inputs x, and receives $\mathbf{I}[x, *]$ from protocol's output
3. Decrypt $\mathbf{I}[x, *]$ for $j = 1, \ldots, 2N$:
   (a) $\mathbf{I}'[x, j] \leftarrow \mathcal{E}.\mathsf{Dec}_{\tau_x}(\mathbf{I}[x, j])$ where $\tau_x \leftarrow \mathsf{KDF}_\kappa(x)$
4. Output $\mathcal{R} \leftarrow id'$ in Stash S and id s.t. $T_f[id] = j$ where $\mathbf{I}[x, j] = 1$ and $j \notin \mathcal{D}$

Update: The overall strategy is to perform a Write-Only ORAM on the column of $\mathbf{I}$ to achieve oblivious file update operations as follows.

$\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}.\mathsf{Update}(f_{id}, \mathcal{I}, \sigma)$: Update file f_{id}

1. Initialize a new column as $\hat{I}[i] \leftarrow 0$, for i $= 1, \ldots, M$
2. Set $\hat{I}[x_i] \leftarrow 1$, where $x_i \leftarrow T_w[w_i]$ for each keyword w_i appearing in f_{id}
3. Add $\langle id, \hat{I} \rangle$ to Stash S, add $T_f[id]$ to dummy set $\mathcal{D}$
4. Download λ random columns of encrypted index $\boldsymbol{I}$ from a server:
   (a) Randomly select λ column indexes $\mathcal{J} \leftarrow \{j_1, \ldots j_\lambda\}$
   (b) Get λ columns $\{\mathbf{I}_l[*, j]\}_{j \in \mathcal{J}}$ from random server $\mathcal{S}_l$
5. Decrypt each column $\mathbf{I}_l[*, j]$ for each $j \in \mathcal{J}$ and for $i = 1, \ldots, M$:
   (a) $\tau_i \leftarrow \mathsf{KDF}_\kappa(i)$
   (b) $\mathbf{I}'[i, j] \leftarrow \mathcal{E}.\mathsf{Dec}_{\tau_i}(\mathbf{I}_l[i, j])$
6. For each dummy column $\mathbf{I}'[*, \hat{j}]$:
   (a) Pick a pair $\langle id, \hat{I} \rangle$ from stash S, and set $\mathbf{I}'[*, \hat{j}] \leftarrow \hat{I}$
   (b) Set $T_f[id] \leftarrow \hat{j}$, and remove $\hat{j}$ from dummy set $\mathcal{D}$
7. Re-encrypt λ columns as $\hat{\mathbf{I}}[i, j] \leftarrow \mathcal{E}.\mathsf{Enc}_{\tau_i}(\mathbf{I}'[i, j])$ for $i = 1 \ldots, M$ and $\forall j \in \mathcal{J}$
8. Send λ columns $\{\hat{\mathbf{I}}[*, j]\}_{j \in \mathcal{J}}$ to ℓ servers, where each server S_i updates its encrypted index as $\mathbf{I}_i[*, j] \leftarrow \hat{\mathbf{I}}[*, j]$, for each $j \in \mathcal{J}$

3.3 $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{it}}$: Robust and IT-Secure ODSE

Although $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$ offers highly-efficient search and update operations, it has the following security limitations: *(i)* it can only (at most) detect but cannot recover from malicious servers, which might tamper the data to compromise the privacy and correctness of the protocol. In privacy-critical applications, it is desirable to recover from malicious servers to improve the robustness of the protocol; *(ii)* the encrypted index and update operations on it are only computationally-secure due to the IND-CPA encryption.

To address the limitations of $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$, we introduce $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{it}}$ that offers *(i)* improved robustness against malicious servers with a partial recover capability, and *(ii)* the highest level of security (i.e., information-theoretic) for both $\mathbf{I}$ and operations on it. The main idea is to share the index with SSS, and harness SSS-based PIR to conduct private search. The robustness comes from the ability to recover the secret shared by SSS in the presence of incorrect shares (see §4).

Setup: The client first constructs an index $\mathbf{I}'$ representing keyword-file relationships as in $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}.\mathsf{Setup}$. Instead of encrypting $\mathbf{I}'$, the client creates shares of $\mathbf{I}'$ by SSS. Since SSS operates on elements in $\mathbb{F}_p$, each row of $\mathbf{I}'$ is split into $\lfloor \log_2 p \rfloor$-bit chunks before SSS computation. So, the index $\mathbf{I}_i$ is the SSS share of $\mathbf{I}'$ for server $\mathcal{S}_i$, which is a matrix of size $M \times 2N'$, where $\mathbf{I}_i[i, j] \in \mathbb{F}_p$ and $N' = N/\lfloor \log_2 p \rfloor$. The detail is as follows.

$(\mathcal{I}, \sigma) \leftarrow \mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{it}}.\mathsf{Setup}(\mathcal{F})$: Create distributed share index from input files $\mathcal{F}$

1. Construct $\mathbf{I}'$ by executing steps 1–3 in $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}.\mathsf{Setup}$ procedure
2. Create SSS of $\mathbf{I}'$ for $i = 1, \ldots, M$ and $j = 1, \ldots, 2N'$:
   (a) $\hat{\mathbf{I}}[i, j]_{\mathsf{bin}} \leftarrow \mathbf{I}'[i, (j-1) \cdot \lfloor \log_2 p \rfloor + 1, \ldots, j \cdot \lfloor \log_2 p \rfloor]$
   (b) $(\mathbf{I}_1[i, j], \ldots, \mathbf{I}_\ell[i, j]) \leftarrow \mathrm{SSS}.\mathsf{CreateShare}(\hat{\mathbf{I}}[i, j], t)$
3. Output $(\mathcal{I}, \sigma)$, where $\mathcal{I} \leftarrow \{\mathbf{I}_1, \ldots, \mathbf{I}_\ell\}$ and $\sigma \leftarrow (T_w, T_f, \mathcal{D})$

Similar to $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$, the client sends $\mathbf{I}_i$ to server $\mathcal{S}_i$ and keep σ as secret.

Search. The client executes the SSS-based PIR protocol on the row dimension of encrypted index to retrieve the row of searched keyword as follows.

$\mathcal{R} \leftarrow \mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{it}}.\mathsf{Search}(w, \mathcal{I}, \sigma)$: Search for keyword w

1. Get row index x of the searched keyword w as $x \leftarrow T_w[w]$
2. Execute $\hat{\mathbf{I}}[x, j] \leftarrow \mathsf{PIR}^{\mathsf{sss}}(x, \langle \mathbf{I}_1[*, j], \ldots, \mathbf{I}_\ell[*, j] \rangle)$ protocol (Figure 4) with ℓ servers for $j = 1, \ldots, 2N'$:
   (a) Each server S_i inputs a column of its shared index $\mathbf{I}_i[*, j]$, where each cell $\mathbf{I}_i[x, j]$ is interpreted as an item in the database
   (b) Client inputs x, and receives $\hat{\mathbf{I}}[x, j]$ from protocol's output. Note that client executes SSS.Recover with privacy parameter of $2t$, instead of t (step 5 in Figure 4) to recover $\hat{\mathbf{I}}[x, j]$ correctly.
3. Form the row as $\mathbf{I}'[x, *] \leftarrow \hat{\mathbf{I}}[x, 1]_{\mathsf{bin}} || \ldots || \hat{\mathbf{I}}[x, 2N']_{\mathsf{bin}}$
4. Output $\mathcal{R} \leftarrow id'$ in Stash S and id s.t. $T_f[id] = j$ where $\mathbf{I}[x, j] = 1$ and $j \notin \mathcal{D}$

Update: We execute Write-Only ORAM on the column dimension of the encrypted index for the file update. Recall that in $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$, λ random columns of the original index $\mathbf{I}'$ are read to update one column. In $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{it}}$, each column of the index $\mathbf{I}_i$ on $\mathcal{S}_i$ contains the share of $\lfloor \log_2 p \rfloor$ successive columns of $\mathbf{I}'$. Therefore, the client reads $\lambda' = \lceil \frac{\lambda}{\lfloor \log_2 p \rfloor} \rceil$ random columns of $\mathbf{I}_i$ from $t+1$ servers to recover λ columns of $\mathbf{I}'$ before performing update. The detail is as follows.

$\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{it}}.\mathsf{Update}(f_{id}, \mathcal{I}, \sigma)$: Update file f_{id}

1. Initialize $I'[i] \leftarrow 0$ for $i = 1 \ldots, M$
2. Set $\hat{I}[x_i] \leftarrow 1$, where $x_i \leftarrow T_w[w_i]$ for each keyword w_i appearing in f_{id}
3. Add $\langle id, \hat{I} \rangle$ to Stash S, add $T_f[id]$ to dummy set $\mathcal{D}$
4. Download λ random columns of shared index $\boldsymbol{I}$ from $t+1$ servers:
   (a) Randomly selected λ' column indexes $\mathcal{J} \leftarrow \{j_1, \ldots j_{\lambda'}\}$
   (b) Get λ' columns $\{\mathbf{I}_l[*, j]\}_{j \in \mathcal{J}, l = 1 \ldots t+1}$ from $t+1$ servers
5. Recover λ' columns for each $j \in \mathcal{J}$ and $i = 1 \ldots, M$:
   (a) $\hat{\mathbf{I}}[i, j] \leftarrow \mathrm{SSS.Recover}(\langle \mathbf{I}_1[i, j], \ldots, \mathbf{I}_\ell[i, j] \rangle, t)$
   (b) $\mathbf{I}'[i, j \cdot \lceil \log_2 p \rceil, ..., (j+1) \cdot \lceil \log_2 p \rceil] \leftarrow \hat{\mathbf{I}}[i, j]_{\mathsf{bin}}$
6. For each dummy column $\mathbf{I}'[*, \hat{j}]$:
   (a) Pick a pair $\langle id, \hat{I} \rangle$ from stash S, and set $\mathbf{I}'[*, \hat{j}] \leftarrow \hat{I}$
   (b) Set $T_f[id] \leftarrow \hat{j}$, and remove $\hat{j}$ from dummy set $\mathcal{D}$
7. Create SSS for λ' column for each $j \in \mathcal{J}$, and $i = 1 \ldots, M$:
   (a) $\mathbf{I}[i, j]_{\mathsf{bin}} \leftarrow \mathbf{I}'[i, j \cdot \lceil \log_2 p \rceil, \ldots, (j+1) \cdot \lceil \log_2 p \rceil]$
   (b) $(\hat{\mathbf{I}}_1[i, j], \ldots, \hat{\mathbf{I}}_\ell[i, j]) \leftarrow \mathrm{SSS.CreateShare}(\hat{\mathbf{I}}[i, j], t)$
8. Send $\hat{\mathbf{I}}_l[*, j])$ to $\mathcal{S}_l$ for each $j \in \mathcal{J}$ and $l = 1 \ldots, \ell$. Each server $\mathcal{S}_l$ updates its share index as $\mathbf{I}_l[*, j] \leftarrow \hat{\mathbf{I}}_l[*, j]$ for each $j \in \mathcal{J}$

4 Security

Definition 1 (ODSE security). *Let* $\boldsymbol{op} = (\mathsf{op}_1, \ldots, \mathsf{op}_q)$ *be an operation sequence over the distributed encrypted index* $\mathcal{I}$*, where* $\mathsf{op}_i \in \{\mathsf{Search}(w), \mathsf{Update}(f_{id})\}$*,* w *is a keyword to be searched and* f_{id} *is a file with keywords to be updated. Let* $\boldsymbol{ODSE}_j(\boldsymbol{o})$ *represent the ODSE client's sequence of interactions with server* $\mathcal{S}_j$*, given an operation sequence* $\boldsymbol{o}$*.*

An ODSE is t-secure *if* $\forall \mathcal{L} \subseteq \{1, \ldots, \ell\}$ *s.t.* $|\mathcal{L}| \leq t$*, for any two operation sequences* $\boldsymbol{op}$ *and* $\boldsymbol{op}'$ *where* $|\boldsymbol{op}| = |\boldsymbol{op}'|$*, the views* $\{\boldsymbol{ODSE}_{i\in\mathcal{L}}(\boldsymbol{op})\}$ *and* $\{\boldsymbol{ODSE}_{i\in\mathcal{L}}(\boldsymbol{op}')\}$ *observed by a coalition of up to* t *servers are (perfectly, statistically or computationally) indistinguishable.*

Remark 1. One might observe that search and update operations in ODSE schemes are performed on rows and columns of the encrypted index, respectively. This access structure might enable the adversary to learn whether the operation is search or update, even though each operation is secure. Therefore, to achieve security as in Definition 1, where the query type should also be hidden, we can invoke both search and update protocols (one of them is the dummy operation) regardless of whether the intended action is search or update.

We argue the security of our proposed schemes as follows.

Theorem 1. ODSE^{wo}_{xor} *scheme is computationally* $(\ell-1)$*-secure by Definition 1.*

Proof. (Sketch) *(i) Oblivious Search*: $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$ leverages XOR-based PIR and therefore, achieves $(\ell - 1)$-privacy for keyword search as proven in [10]. *(ii) Oblivious Update*: $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$ employs Write-Only ORAM which achieves negligible write failure probability and therefore, it offers the statistical security without counting the encryption. The index in $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$ is IND-CPA encrypted, which offers computational security. Therefore in general, the update access pattern of $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$ scheme is computationally indistinguishable. $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$ performs Write-Only ORAM with an identical procedure on ℓ servers (e.g., the indexes of accessed columns are the same in ℓ servers), and therefore, the server coalition does not affect the update privacy of $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$. *(iii) ODSE Security:* By Remark 1, $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$ performs both search and update regardless of the actual operation. As analyzed, search is $(\ell-1)$-private and update pattern is computationally secure. Therefore, $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$ achieves computational $(\ell-1)$-security by Definition 1. □

Theorem 2. ODSE^{wo}_{it} *scheme is statistically* t*-secure by Definition 1.*

Proof. (Sketch) *(i) Oblivious Search*: $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{it}}$ leverages an SSS-based PIR protocol and therefore, achieves t-privacy for keyword search due to the t-privacy property of SSS [13]. *(ii) Oblivious Update*: The index in $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{it}}$ is SSS-shared, which is information-theoretically secure in the presence of t colluding servers. $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{it}}$ also employs Write-Only ORAM, which offers statistical security due to negligible write failure probability. Therefore in general, the update access pattern of $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{it}}$ scheme is information-theoretically (statistically) indistinguishable in the coalition of up to t servers. *(iii) ODSE Security*: By Remark 1, $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{it}}$ performs both search and update protocols regardless of the actual operation. As analyzed above, search is t-private and update pattern is statistically t-indistinguishable. Therefore, $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{it}}$ is information-theoretically (statistically) t-secure by Definition 1. □

4.1 Malicious Input Tolerance

We have shown that ODSE schemes offer a certain level of collusion-resiliency in the honest-but-curious setting where the server follows the protocol faithfully.

In some privacy-critical applications, it is necessary to achieve data integrity in the malicious environment, where the adversary can tamper the query and data to compromise the correctness and privacy of the protocol. We show that ODSE schemes can be extended to detect and be robust against malicious servers as follows. In $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$, we can leverage Message Authentication Code (e.g., HMAC) as presented in [19], where authenticated tag for each row and each column of $\mathbf{I}$ is generated. The server will perform operations (i.e., PIR, Write-Only ORAM) on such tags as similar to encrypted index data and send the result to the client. The client can recover/decrypt the row/column as well as its authenticated tag verify the integrity.

Since $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{it}}$ relies on SSS as the building block, we can not only detect but also be robust against malicious server. The main idea is to leverage list decoding algorithm as in [13], given that the Lagrange interpolation in SSS.$\mathsf{Recover}$ algorithm does not return a consistent value. Such techniques also allow to determine precisely which server has tampered the data. We refer readers to [13] for detailed description. In general, the list decoding allows $t_m \leq t < \ell - \lceil\sqrt{\ell t}\rceil$ number of incorrect shares of $[\![\alpha]\!]^{(t)}$.

5 Experimental Evaluation

5.1 Configurations

Implementation details. We implemented all ODSE schemes in C++. Specifically, we used `Google Sparsehash` to implement hash tables T_f and T_w. We utilized `Intel AES-NI` library to implement AES-CTR encryption/decryption in $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$. We leveraged `Shoup's NTL` library for pseudo-random number generator and arithmetic operations over finite field. We used `ZeroMQ` library for client-server communication. We used multi-threading technique to accelerate PIR computation at the server. Our code is publicly available at

`https://github.com/thanghoang/ODSE`

Hardware and network settings. We used Amazon EC2 with `r4.4xlarge` instance for server(s), each equipped with 16 vCPUs Intel Xeon @ 2.3 GHz and 122 GB RAM. We used a laptop with Intel Core i5 @ 2.90 GHz and 16 GB RAM as the client. All machines ran Ubuntu 16.04. The client established a network connection with the server via WiFi. We used a real network setting, where the download and upload throughputs are 27 and 5 Mbps, respectively.

Dataset. We used subsets of the `Enron` dataset to build $\mathbf{I}$ containing from millions to billions of keyword-file pairs. The largest database in this study contain around 300,000 files with 320,000 unique keywords. Our tokenization is identical to [21] so that our keyword distribution and query pattern is similar to [21].

Instantiation of compared techniques. We compared ODSE with a standard DSSE scheme [9], and the use of generic ORAM atop the DSSE encrypted index. The performance of all schemes was measured under the same setting and in the

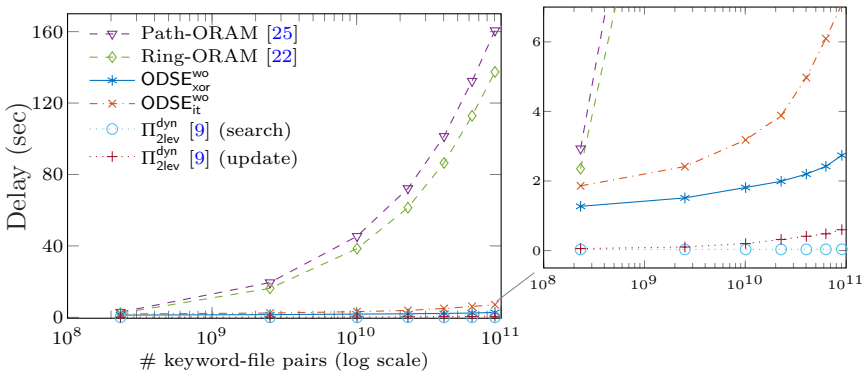


Fig. 5. Latency of ODSE schemes and their counterparts.

average-case cost, where each query involves half of the keywords/files in the database. We configured ODSE schemes and their counterparts as follows.

• *ODSE*: We used two servers for ODSE^{wo}_{xor} and three servers for ODSE^{wo}_{it} scheme. We selected $\lambda = 4$ for ODSE^{wo}_{xor}, and $\lambda' = 4$ with $\mathbb{F}_p$ where p is a 16-bit prime for ODSE^{wo}_{it}. We note that selecting larger p (up to 64 bits) can reduce the PIR computation time, but also increase the bandwidth overhead. We chose a 16-bit prime field to achieve a balanced computation vs. communication overhead.

• *Standard DSSE*: We selected one of the most efficient DSSE schemes by Cash et al. in [9] (i.e., Π^{dyn}_{2lev} variant) to showcase the performance gap between ODSE and standard DSSE. We estimated the performance of Π^{dyn}_{2lev} using the same software/hardware environments and optimizations as ODSE (e.g., parallelization, AES-NI acceleration). Note that we did not use the Java implementation of this scheme available in Clusion library [1] for comparison due to its lack of hardware acceleration support (no AES-NI) and the difference between running environments (Java VM vs. C). Our estimation is conservative in that, we used numbers that would be better than the Clusion library.

• *Using generic ORAM atop DSSE encrypted index*: We selected *non-recursive* Path-ORAM [25] and Ring-ORAM [22], rather than recent ORAMs as ODSE counterparts since they are the most efficient generic ORAM schemes to date. Since we focus on encrypted index rather than encrypted files in DSSE, we did not explicitly compare our schemes with TWORAM [12] but instead, used one of their techniques to optimize the performance of using generic ORAM on DSSE encrypted index. Specifically, we applied the selected ORAMs on the dictionary index containing keyword-file pairs as in [21] along with the round-trip optimization as in [12]. Note that our estimates are also conservative where memory access delays were excluded, and cryptographic operations were optimized and parallelized to make a fair comparison between the considered schemes.

5.2 Overall Results

Figure 5 presents the end-to-end delays of ODSE schemes and their counterparts, where both search and update are performed in ODSE schemes to hide the actual type of operation (see Remark 1). ODSE offers a higher security than standard DSSE at the cost of a longer delay. However, ODSE schemes are 3×-57× faster

Table 1. Comparison of ODSE and its counterparts for oblivious access on **I**.

Scheme	Security				Delay (s)		Distributed Setting†	
	Forward privacy	Backward privacy	Hidden access pattern	Encrypted index*	Search	Update	Privacy level	Improved Robustness
Standard DSSE [9]	✗	✗	✗	Computational	0.036	0.62	-	-
Path-ORAM[25]	✓	✓	Computational	Computational	160.6		-	-
Ring-ORAM [22]	✓	✓	Computational	Computational	137.4		-	-
$\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$	✓	✓	Computational‡	Computational	**2.8**		$\ell - 1$	✗
$\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{it}}$	✓	✓	**Information theoretic**	**Information theoretic**	**7.1**		$< \ell/2$	✓

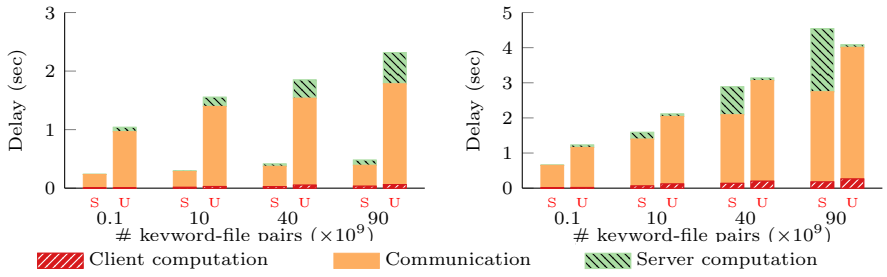
This delay is for encrypted index with 300,000 files and 320,000 keywords regarding network and configuration settings in §5.1.
* The encrypted index in $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{it}}$ is information-theoretically (IT) secure because it is SSS. Other schemes employ IND-CPA encryption so that their index is computationally secure (see §4).
‡ All ODSE schemes perform search and update protocols to hide the actual query type. In $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$, search is IT-secure due to SSS-based PIR and update is computationally secure due to IND-CPA encryption. Hence, its overall security is computational.
† ℓ is # servers. In $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{it}}$, encrypted index and search query are SSS with the same privacy level. Generic ORAM-based solutions have a stronger adversarial model than ours since they are not vulnerable to collusion that arises in the distributed setting.

than the use of generic ORAMs to hide the access patterns. Specifically, with an encrypted index containing ten billions of keyword-file pairs, $\Pi^{\mathsf{dyn}}_{\mathsf{2lev}}$ cost 36 ms and 600 ms to finish a search and update operation, respectively. $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$ and $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{it}}$ took 2.8 s and 7.1 s respectively, to accomplish both keyword search and file update operations, compared with 160 s by using Path-ORAM with the round-trip optimization [12]. $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$ is the most efficient in terms of search, whose delay was less than 1 s. This is due to the fact that $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$ only requires XOR operations and the size of the search query is minimal (i.e., a binary string). $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{it}}$ is more robust (e.g., malicious tolerant) and more secure (e.g., unconditional security) than $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$ at the cost of higher search delay (i.e., 4 s) due to the larger search query and SSS arithmetic computations. For the file update, $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{it}}$ costs 3 s, which is slightly higher than $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$ (i.e., 2.2 secs) since it needs to transmit more data (4 blocks vs. 4 columns) to more servers (3 vs. 2). We further provide a comparison of ODSE schemes with their counterparts in Table 1. We dissected the total cost to investigate which factors contributed the most to the latency of ODSE schemes as follows.

5.3 Detailed Cost Analysis

Figure 6 presents the total delays of separate keyword search and file update operations, as well as their detailed costs in ODSE schemes. Note that ODSE performs both search and update (one of them is dummy) to hide the actual type of operation performed by the client.

• *Client processing:* As shown in Figure 6, client computation contributes the least amount to the overall search delay (less than 10%) in all ODSE schemes. The client computation comprises the following operations: (1) Generate select queries (with SSS in $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{it}}$ and PRF in $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$); (2) SSS recovery and IND-CPA decryption (in $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$); (3) Filter dummy columns. Note that the client delay of ODSE schemes can be further reduced (by at least 50%-60%) via pre-computation of some values such as row keys and select queries (only contain shares of 0 or 1). For the file update, the client performs decryption and re-encryption on λ columns (in $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$), or SSS over λ' blocks (in $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{it}}$). Since we used crypto acceleration (i.e., Intel AES-NI) and highly optimized number



(a) $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$ **(b)** $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{it}}$

Fig. 6. Detailed search (S) and update (U) costs of ODSE schemes.

theory libraries (i.e., NTL), all these computations only contributed to a small fraction of the total delay.

• *Client-server communication:* Data transmission is the dominating factor in the delay of ODSE schemes. The communication cost of $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$ is smaller than that of other ODSE schemes, since the size of search query and the data transmitted from servers are binary vectors. In $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{it}}$, the size of components in the select vector is 16 bits. The communication overhead of $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{it}}$ can be reduced by using a smaller finite field, but at the cost of increased PIR computation on the server side.

• *Server processing:* The cost of PIR operations in $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$ is negligible as it uses XOR. The PIR computation of $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{it}}$ is reasonable, as it operates on a bunch of 16-bit values. For update operations, the server-side cost is mainly due to memory accesses for column update. $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{it}}$ is highly memory access-efficient since we organized the memory layout for column-friendly access. This layout minimizes the memory access delay not only in update but also in search, since the inner product in PIR also accesses contiguous memory blocks by this organization. In $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$, we stored the matrix for row-friendly access to permit efficient XOR operations during search. However, this requires file update to access non-contiguous memory blocks. Hence, the file update in $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$ incurred a higher memory access delay than that of $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{it}}$ as shown in Figure 6.

• *Storage overhead:* The main limitation of ODSE schemes is the size of encrypted index, whose asymptotic cost is $\mathcal{O}(N \cdot M)$, where N and M are the number of files and unique keywords, respectively. Given the largest database being experimented, the size of our encrypted index is 23 GB. The client storage includes two hash tables of size $\mathcal{O}(M)$ and $\mathcal{O}(N \log N)$, the stash of size $\mathcal{O}(M \cdot \log N)$, the set of dummy column indexes of size $\mathcal{O}(N \log N)$, a counter vector of size $\Omega(N)$ and a master key (in $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$ scheme). Empirically, with the same database size discussed above, the client requires approximately 22 MB in both ODSE schemes.

5.4 Experiment with various query sizes

We studied the performance of our schemes and their counterparts in the context of various keyword and file numbers involved in search and update operations that we refer to as "query size". As shown in Figure 7, ODSE schemes are more

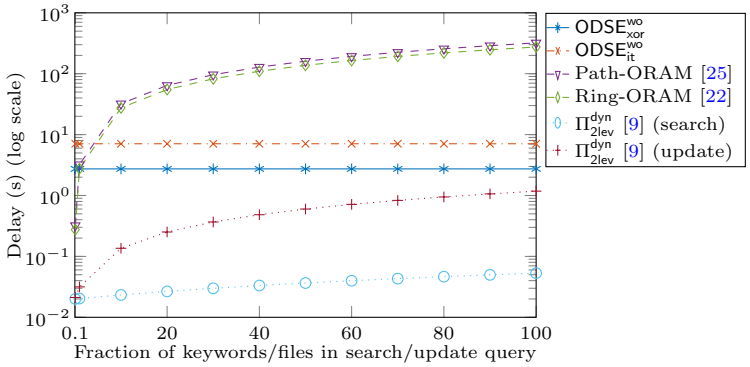


Fig. 7. Latency of ODSE schemes and their counterparts with different fraction of keywords/files involved in a search/update operation.

efficient than using generic ORAMs when more than 5% of keywords/files in the database are involved in the search/update operations. Since the complexity of ODSE schemes is linear to the number of keywords and files (i.e., $\mathcal{O}(M+N)$), their delay is constant and independent from the query size. The complexity of ORAM approaches is $\mathcal{O}(r \log^2(N \cdot M))$, where r is the query size. Although the bandwidth cost of ODSE schemes is asymptotically linear, their actual delay is much lower than using generic ORAM, whose cost is poly-logarithmic to the total number of keywords/files but linear to the query size. This confirms the results of Naveed et al. in [21] on the performance limitations of generic ORAM and DSSE composition, wherein we used the same dataset for our experiments.

6 Conclusion

We proposed a new set of Oblivious Distributed DSSE schemes called ODSE, which achieve full obliviousness, hidden size pattern, and low end-to-end delay simultaneously. Specifically, $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{xor}}$ achieves the lowest end-to-end delay with the smallest communication overhead among all of its counterparts with the highest resiliency against colluding servers. $\mathsf{ODSE}^{\mathsf{wo}}_{\mathsf{it}}$ achieves the highest level of privacy with information-theoretic security for access patterns and the encrypted index, along with the robustness against malicious servers. Our experiments demonstrated that ODSE schemes are one order of magnitude faster than the most efficient ORAM techniques over DSSE encrypted index. We have released the full implementation of our ODSE schemes for public use and wide adaptation.

References

1. The clusion library. https://github.com/encryptedsystems/Clusion/.
2. I. Abraham, C. W. Fletcher, K. Nayak, B. Pinkas, and L. Ren. Asymptotically tight bounds for composing oram with pir. In *IACR Public Key Cryptography*, pages 91–120. Springer, 2017.
3. E.-O. Blass, T. Mayberry, G. Noubir, and K. Onarlioglu. Toward robust hidden volumes using write-only oblivious ram. In *Proceedings of the 2014 ACM CCS*, pages 203–214. ACM, 2014.

4. C. Bösch, P. Hartel, W. Jonker, and A. Peter. A survey of provably secure searchable encryption. *ACM Computing Surveys (CSUR)*, 47(2):18, 2015.
5. 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), 12th International Conference on*, pages 330–337. IEEE, 2014.
6. R. Bost, B. Minaud, and O. Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. Technical report, IACR Cryptology ePrint Archive 2017, 2017.
7. N. Cao, C. Wang, M. Li, K. Ren, and W. Lou. Privacy-preserving multi-keyword ranked search over encrypted cloud data. *IEEE Transactions on parallel and distributed systems*, 25(1):222–233, 2014.
8. D. Cash, P. Grubbs, J. Perry, and T. Ristenpart. Leakage-abuse attacks against searchable encryption. In *Proceedings of the 22nd ACM CCS*, pages 668–679, 2015.
9. 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.
10. B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *Journal of the ACM (JACM)*, 1998.
11. R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proceedings of the 13th ACM CCS*, pages 79–88. ACM, 2006.
12. S. Garg, P. Mohassel, and C. Papamanthou. Tworam: Round-optimal oblivious ram with applications to searchable encryption. *IACR Cryptology ePrint Archive*, 2015:1010, 2015.
13. I. Goldberg. Improving the robustness of private information retrieval. In *IEEE Symposium on Security and Privacy*, pages 131–148. IEEE, 2007.
14. 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.
15. T. Hoang, A. Yavuz, and J. Guajardo. Practical and secure dynamic searchable encryption via oblivious access on distributed data structure. In *Proceedings of the 32nd Annual Computer Security Applications Conference (ACSAC)*. ACM, 2016.
16. M. S. Islam, M. Kuzu, and M. Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *NDSS*, 2012.
17. 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.
18. C. Liu, L. Zhu, M. Wang, and Y.-a. Tan. Search pattern leakage in searchable encryption: Attacks and new construction. *Information Sciences*, 2014.
19. T. Moataz, E.-O. Blass, and T. Mayberry. Chf-oram: a constant communication oram without homomorphic encryption. Technical report, Cryptology ePrint Archive, Report 2015/1116, 2015.
20. T. Moataz, I. Ray, I. Ray, A. Shikfa, F. Cuppens, and N. Cuppens. Substring search over encrypted data. *Journal of Computer Security*, (Preprint):1–30, 2018.
21. M. Naveed. The fallacy of composition of oblivious ram and searchable encryption. In *Cryptology ePrint Archive, Report 2015/668*, 2015.
22. L. Ren, C. W. Fletcher, A. Kwon, E. Stefanov, E. Shi, M. van Dijk, and S. Devadas. Ring oram: Closing the gap between small and large client storage oblivious ram. *IACR Cryptology ePrint Archive*, 2014.
23. A. Shamir. How to share a secret. *Communications of the ACM*, 1979.
24. D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pages 44–55. IEEE Computer Society, 2000.

25. 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 CCS*, pages 299–310. ACM, 2013.
26. W. Sun, B. Wang, N. Cao, M. Li, W. Lou, Y. T. Hou, and H. Li. Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking. In *ACM SIGSAC AsiaCCS*, pages 71–82. ACM, 2013.
27. C. Wang, N. Cao, J. Li, K. Ren, and W. Lou. Secure ranked keyword search over encrypted cloud data. In *IEEE 30th International Conference on Distributed Computing Systems*, pages 253–262. IEEE, 2010.
28. 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. August 2015.
29. R. Zhang, R. Xue, T. Yu, and L. Liu. Dynamic and efficient private keyword search over inverted index–based encrypted data. *ACM Transactions on Internet Technology (TOIT)*, 16(3):21, 2016.
30. 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 Symposium (USENIX Security 16)*, pages 707–720, 2016.
31. F. Zhou, Y. Li, A. X. Liu, M. Lin, and Z. Xu. Integrity preserving multi-keyword searchable encryption for cloud computing. In *International Conference on Provable Security*, pages 153–172. Springer, 2016.