Obfuscated Access and Search Patterns in Searchable Encryption

Zhiwei Shang
University of Waterloo
z6shang@edu.uwaterloo.ca

Simon Oya
University of Waterloo
simon.oya@uwaterloo.ca

Andreas Peter
University of Twente
a.peter@utwente.nl

Florian Kerschbaum
University of Waterloo
florian.kerschbaum@uwaterloo.ca

***Abstract*—Searchable Symmetric Encryption (SSE) allows a data owner to securely outsource its encrypted data to a cloud server while maintaining the ability to search over it and retrieve matched documents. Most existing SSE schemes leak which documents are accessed per query, i.e., the so-called access pattern, and thus are vulnerable to attacks that can recover the database or the queried keywords. Current techniques that fully hide access patterns, such as ORAM or PIR, suffer from heavy communication or computational costs, and are not designed with search capabilities in mind. Recently, Chen et al. (INFOCOM'18) proposed an obfuscation framework for SSE that protects the access pattern in a differentially private way with a reasonable utility cost. However, this scheme always produces the same obfuscated access pattern when querying for the same keyword, and thus leaks the so-called search pattern, i.e., how many times a certain query is performed. This leakage makes the proposal vulnerable to certain database and query recovery attacks.**

In this paper, we propose OSSE (Obfuscated SSE), an SSE scheme that obfuscates the access pattern independently for each query performed. This in turn hides the search pattern and makes our scheme resistant against attacks that rely on this leakage. Given certain reasonable assumptions on the database and query distribution, our scheme has smaller communication overhead than ORAM-based SSE. Furthermore, our scheme works in a single communication round and requires very small constant client-side storage. Our empirical evaluation shows that OSSE is highly effective at protecting against different query recovery attacks while keeping a reasonable utility level. Our protocol provides significantly more protection than the proposal by Chen et al. against some state-of-the-art attacks, which demonstrates the importance of hiding search patterns in designing effective privacy-preserving SSE schemes.

I. Introduction

Searchable Symmetric Encryption (SSE) [33] schemes allow a client to securely perform searches on an encrypted database hosted by a server. In a typical SSE scenario, the client first locally produces an encrypted version of the database and a search index, and outsources them to the server. Later, the client can issue queries for keywords that the server can securely run on the index, and retrieve the documents that match the query to decrypt them locally.

Originally published at:
Network and Distributed Systems Security (NDSS) Symposium 2021
21-24 February 2021, San Diego, CA, USA

Even though the content of the queries and the documents are encrypted, during this interaction the server learns which documents are accessed, i.e., the *access pattern*, and which queries are equal, i.e., the *search pattern*. Most existing SSE schemes [7], [9], [12], [21]–[23], [33] allow such leakage for performance considerations. However, recent studies [8], [20], [39] demonstrated that, with some prior knowledge of the outsourced database or a subset of the queries, an honest-but-curious server can recover the underlying keywords of queries with high accuracy, which violates the client's privacy.

There are different techniques that allow enhancing the privacy properties of SSE schemes, but they incur an utility cost which is typically a combination of communication overhead, extra computational complexity, and local client storage requirements. Certain schemes, like those based on Oblivious RAM (ORAM) [19] or Private Information Retrieval (PIR) [11], can fully hide the access pattern when reading a document from a database. However, they incur a large communication and computation overhead, respectively, and are not specifically designed towards securely searching over an encrypted database (except for TWORAM [17]). A recent framework by Chen et al. [10] protects access-pattern leakage in SSE by obfuscating the index of the database before outsourcing it. This way, the server only learns obfuscated access patterns, making it harder to successfully carry out attacks on the client's privacy from such leakage. However, despite its efficiency, this framework cannot hide *search patterns* since the access pattern for each keyword is determined after outsourcing. This search pattern leakage allows different practical attacks [20], [27], [31] to perform remakably well regardless of the access pattern obfuscation (see Sect. IX).

Motivated by this vulnerability of Chen et al.'s scheme [10], in this work we propose OSSE (Obfuscated SSE), a new SSE scheme that protects *both* the access and search patterns. The main idea behind OSSE is that it produces a *fresh obfuscation* per query, instead of just once when outsourcing the database, thus making it hard for the server to decide whether or not two queries are for the same keyword. Our scheme allows to perform queries on the encrypted database and receive the matched documents *in the same communication round* (TWORAM requires at least four rounds [17]). Under some reasonable assumptions on the query and database distribution, OSSE achieves a lower communication overhead than TWORAM (e.g., only a small constant when the keyword distribution is uniform). Our scheme relies on computation-heavy cryptographic techniques and thus its computational cost is considerable (e.g., it can require 30 minutes to run a query

over 30 000 documents). However, it requires very small constant client-side storage and the query process is parallelizable, which means that the server can process several queries in parallel to speed up the search, and return the results in a single round. Our experimental evaluation shows that obfuscating search patterns significantly reduces the performance of known practical query recovery attacks.

Summarizing, our contributions are the following:

- We propose OSSE, an SSE that leverages Inner Product Predicate Encryption (IPPE) [32] whose main features are 1) it obfuscates both access and search patterns; 2) it runs queries in a single communication round, with an overhead that, depending on the database and query distribution, can be as small as a constant (three); 3) its computational complexity is $O(n \log n / \log \log n)$, but can be significantly reduced with parallelization; and 4) OSSE requires a (very small) constant client-side storage, as opposed to ORAM-based SSE (TWORAM) [17] that requires $O(\log^2 n)$ storage.
- We instantiate the notion of differential privacy for queries and documents in SSE, which in turn imply search and access pattern privacy, respectively. We prove the differential privacy guarantees of OSSE.
- We show that, even when OSSE is tuned to provide high utility (and a low differential privacy parameter ϵ), our scheme still provides strong protection against four different query identification attacks [8], [20], [27], [31].

The paper is structured as follows. In Sect. II we review related work, before we introduce preliminaries in Sect. III. We characterize the performance metrics we consider in Sect. IV and present OSSE in Sect. V. We analyze the security, privacy, and complexity of our scheme in Sects. VI, VII, VIII, respectively, and evaluate it against empirical attacks in Sect. IX. We conclude in Sect. X.

II. Related Work

A. Searchable Symmetric Encryption (SSE)

SSE refers to a type of encryption that allows a data owner to outsource an encrypted database to an untrusted server while still preserving search functionalities. SSE was put forward by Song et al. [33], who suggested several practical constructions whose search complexity is linear in the database size and secure under the Chosen Plaintext Attack (CPA). Goh et al. [18] pointed out CPA was not adequate for SSE schemes. Curtmola et al. [12] provided formal notions of security and functionality for SSE, as well as the first constructions satisfying them with search complexity linear in the number of results (sub-linear in the size of the database). Subsequent works provided different security features, efficiency properties, and functionalities [7], [9], [21]–[23], [30]. However, all the above SSE schemes reveal which documents are accessed and returned in each query. This access pattern leakage opened the door to powerful query recovery attacks [6], [8], [20], [31], [39].

B. Query Recovery Attacks

In 2012, Islam et al. [20] demonstrated that when knowing some statistics about a database and the content of a small fraction of queries, a semi-honest server could recover the contents of all queries with more than 90% accuracy. This is the first powerful attack (known as IKK attack) utilizing access-pattern leakage. Subsequent works propose attacks that are effective when the adversary only knows a subset of the database [6], [8] or has imperfect auxiliary information [31]. A different type of attack, called file-injection attack [6], [39], showed that an active adversary could inject only a small number of carefully designed files in order to recover the content of queries by observing the access patterns of the injected files. Liu et al. [27] introduced an attack that leverages prior knowledge about the client's search habits and search-pattern leakage to identify the underlying keywords of the client's queries. Even though these attacks aim at identifying the underlying keywords of the queries (query recovery), this in turn can allow the adversary to know the keywords of each document (database recovery).

C. Oblivious RAM and Private Information Retrieval

Oblivious RAM (ORAM), first introduced by Goldreich and Ostrovsky [19], was designed to hide memory access patterns by a CPU. Goldreich and Ostrovsky showed that a client could hide entirely the access patterns by continuous shuffling and re-encrypting data, with a $poly(\log n)$ communication overhead and $O(\log n)$ client-side storage. Since its proposal, there has been a fruitful line of research on further reducing this overhead [3], [14], [28], [34], [37]. Path ORAM [34], popular for its simplicity and efficiency, achieved $O(\log n)$ communication overhead with $O(\log n)$ client storage but required block size to be $\Omega(\log n)$. Apon et al. [3] showed that one can construct an ORAM scheme with constant communication overhead by leveraging fully homomorphic encryption (FHE). Even though subsequent works optimized this overhead [14], [28], FHE-based ORAM constructions still require a large communication overhead and rely on computationally expensive cryptographic primitives. A $\Omega(\log n)$ lower bound of the overhead has been proven in the computational [26] and the statistical offline setting [19]. Naively combining SSE and ORAM might lead to a communication volume larger than directly downloading the entire database [29]. Garg et al. [17] proposed TWORAM, an ORAM-based SSE scheme to hide access patterns with a communication overhead $O(\log n \cdot \log \log n)$, which requires at least four communication rounds and $O(\log^2 n)$ client-side storage.

Private Information Retrieval (PIR), proposed by Chor et al. [11], allows a group of clients to privately retrieve documents from a public (unencrypted) database in the setting where there are many non-cooperating copies of the same database. The main drawback of these schemes is that they require to touch every bit in the database per access, i.e., a computation overhead of $O(n)$.

We remark that ORAM and PIR provide private database access but, contrary to SSE schemes, they are *not designed to provide search functionalities* by default (except for TWORAM [17]). This means that a client must know beforehand the exact indices of the documents to be accessed. Database search can also be implemented using only FHE [1], [2]. The main drawback of these schemes is that, even though they have built-in search capabilities, they require multiple communication rounds to retrieve variable length results.

Notation	Description
Δ	Keyword universe $\Delta = \{w_{(1)}, w_{(2)}, \ldots, w_{\lvert\Delta\rvert}\}$.
D	A document, $D = \{w_1, w_2, \ldots, w_{\lvert D\rvert}\}$.
$id(D)$	Document identifier (or id) of D.
$\mathcal{D}$	Dataset, list of all documents ordered by their ids.
$\mathcal{D}[i]$	ith document in $\mathcal{D}$, whose id is i.
n	Total number of documents, $n \doteq \lvert\mathcal{D}\rvert$.
$\mathcal{D}(w)$	List of all documents that contain w ordered by id.
I	Secure search index.

TABLE I: Database Notation

D. Differentially Private SSE

Chen et al. [10] proposed a differentially private obfuscation framework to mitigate access-pattern leakage in SSE. The framework obfuscates the search index, i.e., a list of which documents contain keywords, by adding false positives and false negatives to it. Consequently, the server only learns an obfuscated version of the access patterns, from which it is much harder to derive useful information. However, since the obfuscation is fixed after outsourcing the index, querying for the same keyword multiple times results in the same obfuscated pattern. This repetition actually reveals query frequencies of keywords, which can be used to recover the keywords being queried [27].

III. Preliminaries

This section presents the formal definitions related to SSE and IPPE, that we need to characterize the leakage and security of our scheme, OSSE. We give a high-level description of our scheme OSSE in Section V-A.

A. Searchable Symmetric Encryption (SSE)

First, we introduce the notation related to SSE, which we summarize in Table I. Let $\Delta = \{w_{(1)}, w_{(2)}, \ldots, w_{(|\Delta|)}\}$ be the keyword universe, and let $|\Delta|$ denote its size. Let D be a document and let $id(D)$ be its document identifier, which we assume independent of its content. For notational simplicity, we treat documents as a list of the keywords they contain, e.g., $D = \{w_1, w_2, \ldots, w_{|D|}\}$, and $w \in D$ means document D contains keyword w. Let $\mathcal{D}$ be a dataset of n documents, sorted by their ids. $\mathcal{D}[i]$ is the ith document in the dataset; without loss of generality, we assume $id(\mathcal{D}[i]) = i$. Let $\mathcal{D}(w)$ be the list of documents that contain w ordered by id.

In the following, we define Searchable Symmetric Encryption (SSE) schemes and borrow the security definition [12].

Definition 1 (Searchable Symmetric Encryption Scheme (SSE))**.** *An SSE scheme is a collection of the following four polynomial-time algorithms:*

- $\mathsf{Keygen}(1^\lambda)$ *is a probabilistic key generation algorithm that is run by the client to setup the scheme. It takes a security parameter λ and returns a secret key sk such that the length of sk is polynomially bounded in λ.*
- $\mathsf{BuildIndex}(sk, \mathcal{D})$ *is a (possibly probabilistic) algorithm that the client runs using the secret key sk and document collection $\mathcal{D}$ to generate an encrypted index I whose length is polynomially bounded in λ.*
- $\mathsf{Trapdoor}(sk, w)$ *is run by the client to generate a trapdoor τ_w for a keyword w given the secret key sk.*
- $\mathsf{Search}(I, \tau_w)$ *is run by the server in order to search for $\mathcal{D}(w)$. It takes an encrypted index I for a collection $\mathcal{D}$ and a trapdoor τ_w for keyword w as inputs, and returns the set of identifiers of documents containing w.*

In order to define the security of an SSE scheme, we borrow the concepts of *history*, *view* and *trace* [12]. First, the history contains the sensitive information that the client wants to keep private, namely the document plaintexts $\mathcal{D}$ and the sequence of keywords queried $\vec{w}$:

Definition 2 (History)**.** *A history over $\mathcal{D}$ is a tuple*

$$H_t \doteq (\mathcal{D}, \vec{w}), \quad \text{where } \vec{w} \doteq (\vec{w}[1], \vec{w}[2], \ldots, \vec{w}[t]). \tag{1}$$

Here, $\vec{w}$ is the vector of underlying keywords of the t queries.

A *partial history* of H_t, denoted H_t^s for $s \leq t$, contains only the sequence of queries up to the sth query, i.e., $H_t^s \doteq (\mathcal{D}, \vec{w}')$ where $\vec{w}' \doteq (\vec{w}[1], \ldots, \vec{w}[s])$.

Next, the *view* specifies what the server can see when running the SSE protocol, namely the document identifiers $id(\mathcal{D})$, the encryption of each document $\mathcal{E}(\mathcal{D}[i])$, a secure search index I, and the query tokens (trapdoors) τ_i:

Definition 3 (View)**.** *The view of H_t under key sk is the vector*

$$V_{sk}(H_t) \doteq (id(\mathcal{D}), \mathcal{E}(\mathcal{D}[1]), , ..., \mathcal{E}(\mathcal{D}[n]), I, \tau_1, ..., \tau_t), \tag{2}$$

where τ_i is the query token for the ith query. The partial view *$V_{sk}^s(H_t)$ only contains the tokens up to τ_s, with $s \leq t$.*

Finally, the *trace* of a history models the actual leakage of the SSE scheme. Before defining trace, we first define formally two types of leakage of SSE schemes: the access pattern and the search pattern.

Definition 4 (Access Pattern)**.** *The access pattern $\Pi_{\vec{w}}$ given a dataset $\mathcal{D}$ and a query vector $\vec{w}$ is a binary matrix of size $t \times n$ such that*

$$\Pi_{\vec{w}}[i, j] = \begin{cases} 1 & \text{if } \vec{w}[i] \in \mathcal{D}[j]; \\ 0 & \text{otherwise}. \end{cases} \tag{3}$$

Definition 5 (Search Pattern)**.** *The search pattern $\Phi_{\vec{w}}$ given a dataset $\mathcal{D}$ and a query vector $\vec{w}$ is a symmetric binary matrix of size $t \times t$ such that*

$$\Phi_{\vec{w}}[i, j] = \begin{cases} 1 & \text{if } \vec{w}[i] = \vec{w}[j]; \\ 0 & \text{otherwise}. \end{cases} \tag{4}$$

The access pattern reveals which documents contain which of the queried keywords, and the search pattern reveals which queries have the same underlying keyword.

Definition 6 (Trace)**.** *The trace of H_t is the vector*

$$T(H_t) \doteq (id(\mathcal{D}), |\mathcal{D}[1]|, \ldots, |\mathcal{D}[n]|, \Pi_{\vec{w}}, \Phi_{\vec{w}}). \tag{5}$$

We are ready to define the adaptive semantic security for SSE. Informally, the definition states that an SSE scheme is secure if an adversary that observes the view can be simulated by an algorithm that only sees the trace; this implies that the trace contains all the information that is relevant to the adversary.

Definition 7 (Adaptive Semantic Security for SSE [12])**.** *An SSE scheme is adaptively semantically secure if for all $t \in \mathbb{N}$ and for all (non-uniform) probabilistic polynomial-time adversaries $\mathcal{A}$, there exists a (non-uniform) probabilistic polynomial-time algorithm (the simulator) $\mathcal{S}$ such that for all traces T_t of length t, all polynomially samplable distributions $\mathcal{H}_t$ over $\{H_t : T(H_t) = T_t\}$ (i.e., the set of histories with trace T_t), all functions $f : \{0,1\}^{|H_t|} \to \{0,1\}^{poly(|H_t|)}$, all $0 \leq s \leq t$, and sufficiently large λ:*

$$\left|Pr[\mathcal{A}(V^s_{sk}(H_t)) = f(H^s_t)] - Pr[\mathcal{S}(T(H^s_t)) = f(H^s_t)]\right| < \delta(\lambda)$$

where $H_t \leftarrow \mathcal{H}_t, sk \leftarrow$ Keygen(1^λ), and probabilities are taken over $\mathcal{H}_t$ and the internal coins of $\mathcal{A}, \mathcal{S}$ and the underlying Keygen, BuildIndex, Trapdoor, Search algorithms.

Curtmola et al. [12] prove the equivalence of this definition and adaptive indistinguishability for SSE. In Section VI we show that our SSE scheme provides adaptive semantic security.

B. Inner Product Predicate Encryption (IPPE)

We explain IPPE [5], [24], [32], a cryptographic tool that our SSE scheme uses. In IPPE, plaintexts $x \in \Sigma$ and predicates $f \in \mathcal{F}$ are vectors $\Sigma = \mathcal{F} = \mathbb{Z}^s_N$. We say that a plaintext x satisfies a predicate f, denoted $f(x) = 1$, if $\langle x, f\rangle = 0$, where $\langle \cdot, \cdot \rangle$ denotes the inner product; otherwise, $f(x) = 0$.

Definition 8. *(*SYMMETRIC-KEY INNER PRODUCT PREDICATE ENCRYPTION *[32]) (IPPE) An IPPE scheme for the class of predicates $\mathcal{F}$ over the set of attributes Σ consists of the following probabilistic polynomial-time algorithms.*

- *Setup(1^λ) takes as input a security parameter λ and outputs a secret key sk.*
- *Encrypt(sk, x) takes as input a secret key sk and a plaintext $x \in \Sigma$ and outputs a ciphertext ct_x.*
- *GenToken(sk, f) takes as input a secret key sk and a predicate $f \in \mathcal{F}$ and outputs a search token st_f.*
- *Query(st_f, ct_x) takes as input a token st_f for a predicate f and a ciphertext ct_x for plaintext x and outputs $f(x) \in \{0,1\}$.*

Correctness. IPPE correctnes requires that, for all λ, $x \in \Sigma$, and $f \in \mathcal{F}$; letting $sk \leftarrow \mathsf{Setup}(1^\lambda)$, $st_f \leftarrow \mathsf{GenToken}(sk, st_f)$, and $ct_x \leftarrow \mathsf{Encrypt}(sk, x)$,

1) If $\langle x, f\rangle = 0$, then $\mathsf{Query}(st_f, ct_x) = 1$.
2) If $\langle x, f\rangle \neq 0$, then $\Pr[\mathsf{Query}(st_f, ct_x) = 0] \geq 1 - \delta(\lambda)$ where δ is a negligible function.

An IPPE scheme can be used to securely evaluate polynomials [24], as follows. Let $\vec{\alpha} = (a_0, a_1, \ldots, a_d)$ be the coefficients of a polynomial P of degree d. The evaluation of this polynomial at point x is simply $P(x) = \sum_{i=0}^{d} a_i \cdot x^i = \langle \vec{\alpha}, \vec{\beta}\rangle$ where $\vec{\beta} = (x^0, x^1, \ldots, x^d)$. Therefore, the condition $P(x) = 0$ can be verified by checking whether $\mathsf{Query}(st_{\vec{\beta}}, ct_{\vec{\alpha}}) = 1$, where $ct_{\vec{\alpha}} \leftarrow$ IPPE.$\mathsf{Encrypt}(sk, \vec{\alpha})$ and $st_\beta \leftarrow$ IPPE.$\mathsf{GenToken}$ $(sk, \vec{\beta})$.

Security. Shen et al. [32] define the security of an IPPE scheme by the following game G between an adversary $\mathcal{A}$ and a challenger controlling the IPPE.

- **Setup**: The challenger runs IPPE.$\mathsf{Setup}(1^\lambda)$, keeps sk to itself, and picks a random bit b.
- **Queries**: $\mathcal{A}$ adaptively issues two types of queries:
  - Ciphertext query. On the jth ciphertext query, $\mathcal{A}$ outputs two plaintexts $x_{j,0}, x_{j,1} \in \Sigma$. The challenger responds with IPPE.$\mathsf{Encrypt}(sk, x_{j,b})$.
  - Token query. On the ith token query, $\mathcal{A}$ outputs descriptions of two predicates $f_{i,0}, f_{i,1} \in \mathcal{F}$. The challenger responds with IPPE.$\mathsf{GenToken}(sk, f_{i,b})$.

  $\mathcal{A}$'s queries are subject to the restriction that, for all ciphertext queries $(x_{j,0}, x_{j,1})$ and all predicate queries $(f_{i,0}, f_{i,1})$, $f_{i,0}(x_{j,0}) = f_{i,1}(x_{j,1})$.
- **Guess**: $\mathcal{A}$ outputs a guess b' of b.

The advantage of $\mathcal{A}$ is defined as $Adv_{\mathcal{A}} = |Pr[b' = b] - \frac{1}{2}|$.

Definition 9. *(*FULL SECURITY *for IPPE [32]). A symmetric-key inner product predicate encryption scheme is fully secure if, for any probabilistic polynomial adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ in winning the above game is negligible in λ.*

Roughly speaking, full security guarantees that given a set of tokens of predicates $f_1, \ldots, f_k$ and a set of encryptions of plaintexts $x_1, \ldots, x_t$, no adversary can gain any information about any predicate or any plaintext other than the value of each predicate evaluated on each of the plaintexts. The notion of predicate privacy is inherently impossible in the public-key setting, which is the reason why our construction works only in the private-key setting.

A stronger security notion in the context of IPPE, called simulation-based security (SIM-security), requires that every efficient adversary $\mathcal{A}$ that interacts with the real IPPE can be simulated given only oracle access to the inner products between each pair of vectors that $\mathcal{A}$ submits to the real IPPE. SIM-security implies full security, and an IPPE that provides SIM-secure is also called a Function-Hiding IPPE (FHIPPE).

Definition 10. *(*SIM-SECURITY *for IPPE). An IPPE is SIM-secure if it is fully secure and, for any efficient $\mathcal{A}$, there exists an efficient simulator $\mathcal{S}$ such that the following two games are computationally indistinguishable:*

$\mathsf{Real}_{\mathcal{A}}(1^\lambda)$:	$\mathsf{Ideal}_{\mathcal{A},\mathcal{S}}(1^\lambda)$:
$sk \leftarrow$ IPPE.$\mathsf{Setup}(1^\lambda)$	$sk' \leftarrow \mathcal{S}.\mathsf{Setup}(1^\lambda)$
$b' \leftarrow G_{\mathcal{A},\mathsf{IPPE}}(1^\lambda)$	$b'' \leftarrow G_{\mathcal{A},\mathcal{S}}(1^\lambda)$
output b'	*output* b''

Here, game $G_{\mathcal{A},\mathsf{IPPE}}$ represents the game defined previously, played between $\mathcal{A}$ and IPPE, and $G_{\mathcal{A},\mathcal{S}}$ is the game between $\mathcal{A}$ and $\mathcal{S}$.

IV. PERFORMANCE METRICS IN SSE

As explained earlier, existing SSE schemes offer different privacy, performance, and utility trade-offs. Schemes that leak the access and search patterns typically provide high performance and utility, while high protection solutions such as ORAM or PIR incur some computation cost, communication cost, or client storage requirements. Our goal is to find a middle-ground solution that obfuscates the search/access patterns while being lighter in terms of cost. We make a distinction between *utility metrics*, that directly affect the outcome of the protocol, and *performance metrics*, that measure the cost of running the protocol.

From the client's perspective, the outcome of the protocol are the documents received from the server as a query response. Privacy-preserving schemes can sometimes return documents that do not contain the requested keyword (false positives), or miss out some documents that do contain the keyword (false negatives). This utility loss can be characterized by two metrics:

- **True Positive Rate** (TPR): is the probability that the server returns a document that contains the queried keyword as a response to a query.
- **False Positive Rate** (FPR): is the probability that the server returns a document that *does not* contain the requested keyword as a response to a query.

In most cases, the client aims for large TPR values (e.g., > 0.9999) and low FPR values (e.g., < 0.01).

On the other hand, we consider the following performance metrics that measure the cost of running the SSE algorithm:

- **Communication Cost:** is the number of bytes exchanged between the client and the server and the number of communication rounds needed to perform each query.
- **Computational Cost:** is the number of operations that the client and server perform to run the SSE scheme.
- **Client Storage Cost:** is the amount of local memory that the client needs in order to run the SSE scheme.

The TPR and FPR can be palliated at the expense of an increase in the cost of the protocol: The TPR can be increased by replicating documents on the server side, dividing each document into K shards and requiring $k < K$ shards for document recovery [10]. This incurs an extra communication, computation, and server storage cost. Likewise, false positives can be trivially filtered by the client by checking whether or not a returned document contains the queried keyword. Thus, the effective cost of false positives is bandwidth consumption.

Our SSE scheme, OSSE, requires a *single communication round*, a (small) constant client storage, and can run queries in parallel, which can potentially allow for faster running times.

Finally, we consider both theoretical and empirical privacy metrics. From the theoretical perspective, in Sect. VII we study the performance of our scheme and previous proposals in terms of *differential privacy* [15], that has already been applied to our setting [10]. Then, in Sect. IX we use the *empirical accuracy* of known attacks [8], [20], [27], [31] to assess the privacy properties of SSE schemes in practice.

V. Algorithm Description

We describe our proposal, that we call Obfuscated SSE (OSSE). Table II summarizes the notation of this section.

Notation	Description
Π_w	Real access pattern for keyword w (Def. 4).
$\tilde{\Pi}_w$	Obfuscated access pattern for keyword w.
l	Document label, computed as a hash $l \doteq h(id(D))$.
$\texttt{ctr}$	Counter used in our OSSE scheme.
$\texttt{ctr}_{\max}$	Maximum value of $\texttt{ctr}$.
$w_{\text{-}1}$	A dummy keyword such that $w_{\text{-}1} \notin \Delta$.
$I[i]$	Encrypted polynomial that allows computing query matches for document i.
I	Search index, $I = (I[1], I[2], \ldots, I[n])$.
τ_f	Query token for a predicate f.
Γ_w	Set of (τ_f, l) generated when querying for w.

TABLE II: Notation for the Algorithm Description

A. Construction Overview

In a traditional SSE scheme, when the client queries for a keyword w, the server can compute the access pattern Π_w, i.e., a list of the identifiers of the documents in the database that contain w. The premise for our scheme is simple: we want to introduce random false positives (i.e., return documents that do not contain w) and false negatives (i.e., some of the documents that contain w are not returned). By doing so, we only allow the server to observe an *obfuscated access pattern* $\tilde{\Pi}_w$. Each query results in a freshly random access pattern, which in turn also hides the search pattern, since given a set of obfuscated access patterns it is hard to tell whether or not these patterns where generated by queries for the same or different keywords. Despite the simplicity of this premise, achieving this functionality in a single communication round and without client storage is particularly challenging.

OSSE works as follows. The client first generates its private key. Then, for each document in the dataset, the client calls a function that we call $\mathsf{genPoly}$ to generate a polynomial that encodes the keywords of that document. These polynomials are encrypted using the private key, and this collection of ciphertexts forms the *search index*. The client sends the encrypted dataset and the search index to the server. When querying for a keyword w, the client calls a function $\mathsf{genPred}(w)$ to generate a set of predicates, which are then converted into *tokens*. The client sends these tokens to the server. The server evaluates the tokens on the search index (i.e., on the encrypted polynomials of each document); this evaluation returns a match for those documents that meet the query and must be returned to the client. The key to adding random false positives and false negatives to this matching process lies in the construction of the functions $\mathsf{genPoly}$ and $\mathsf{genPred}$, that we explain below.

In order to speed up computation, the client assigns a *label* l to each document by hashing its document identifier, i.e., the label of D is $h(id(D))$. The hash function is publicly know, so the server can compute the label of each encrypted document. Then, the client attaches a label l to each query token, so that the server only needs to evaluate each token on the encrypted polynomials of documents that share the label l.

B. Polynomial Generation ($\mathsf{genPoly}$)

The function $\mathsf{genPoly}(D, id(D))$ generates the polynomial coefficients for document D, that are later encrypted and sent to the server as part of the search index. Before explaining this function, we clarify the following point: when the client sends a query token to the server, we want to avoid multiple document matches for a single token, since this reveals that two documents contain the same keyword and introduces correlations in the query response, which are hard to take into account in a privacy analysis. Therefore, the client designs the polynomials and query tokens so that a token can either match a single document or none at all. As a consequence, the client must generate multiple tokens per query in order to retrieve all the desired documents.

Now we explain how the client crafts the polynomials. Consider a document $D = \{w_1, w_2, \ldots, w_{|D|}\}$ that contains $|D|$ keywords, has identifier $id(D)$, and label $l \doteq h(id(D))$. Let $S_{\max}$ be the maximum number of keywords that any document can have. Then, the roots of the polynomial (i.e., the values for which it should return a match) are the following ($||$ denotes concatenation):

$$\begin{aligned} (w_i||l||\mathtt{ctr}_{w_i,l})\,, &\quad \text{for } i = 1, 2, \ldots, |D|\,, \\ (w_{\text{-}1}||0||0)\,, &\quad \text{for } i = |D|+1, \ldots, S_{\max}\,, \\ (id(D)||0||\text{-}1)\,. & \end{aligned} \tag{6}$$

The first set of roots $(w_i||l||\mathtt{ctr}_{w_i,l})$ include each keyword w_i in the document, the document label l, and a counter $\mathtt{ctr}_{w_i,l}$ that is chosen so that none of the polynomials (one polynomial per document) have a root in common (so as to avoid more than one match per token). This counter starts at zero and keeps increasing as the client builds the search index. For example, two documents $\mathcal{D}[i]$ and $\mathcal{D}[k]$ might both contain a particular keyword w and share a label $l = h(i) = h(k)$. However, their corresponding polynomials will not share a root, since the value in their $\mathtt{ctr}_{w,l}$ field will be different. The maximum value of this counter is fixed before building the search index, and it is chosen such that, with overwhelming probability, the client does not need a higher counter value (more on this on Sect. V-G). These roots $(w_i||l||\mathtt{ctr}_{w_i,l})$ allow the client to find true positive matches by generating tokens using this structure with their desired query keyword w, and looping through all labels l and counter values.

The second set of roots $(w_{\text{-}1}||0||0)$ include a dummy keyword $w_{\text{-}1}$ that is not in the keyword universe $w_{\text{-}1} \notin \Delta$. These roots act as padding, so that every polynomial has the same length (the same number of roots and coefficients) and thus hide the number of keywords that each document has. Finally, the last root $(id(D)||0||\text{-}1)$ allows the client to trigger false positives using only the document identifier $id(D)$ by generating a token with this structure.

Building the search index following (6) ensures that the only root in common between polynomials generated for different document is $(w_{\text{-}1}||0||0)$, which is only used for padding and never queried. This means that a token can only trigger at most one match. When querying for keyword w, the client can generate false negatives by skipping some label/counter values in the generation of tokens $(w||l||\mathtt{ctr})$, force false positives by generating tokens from $(id(D)||0||\text{-}1)$, and force non-matches by generating tokens from values that do not match any polynomial, e.g., $(w_{\text{-}1}||\text{-}1||0)$.

The function genPoly takes as input a document $\mathcal{D}[i]$ and its label i, generates a polynomial based on its keywords as detailed in (6), and returns the vector of polynomial coefficients v_i (which is straightforward to compute from the roots). Since we have $S_{\max} + 1$ roots, the vector of polynomial coefficients will be of size $S_{\max} + 2$.

C. Predicate Generation (genPred)

The client calls genPred when performing a query. This function receives a keyword w and outputs a set of (predicate, label) tuples. The client converts them to (token, label) pairs and sends to the server to run the query.

The procedure to generate the predicate and label pairs is shown in Algorithm 1. The algorithm starts by initializing a multiset that will store the values on which to evaluate the polynomials, and their associated labels. Then, the first part of the algorithm (lines 3-5) attempts to generate a query for each label and counter value, but only adds each with probability p. Typically, p will be close to 1. The goal of this part is to try to obtain a match in those documents that actually contain w, while having a small probability $(1-p)$ of false negative. The remaining code aims at hiding the true positives and non-matches produced by the first part of the algorithm. The second part (lines 6-9) hides the true positives by generating false positives for each document in the dataset following a geometric distribution with parameter $1-q$ (we want q close to zero to avoid a huge amount of false positives). We explain this distribution choice in Sect. V-F. The third part (lines 10-13) hides the non-matches of the first part by generating non-matches for each label $l \in [|h|]$ following a geometric distribution with parameter $1-q$. The remaining lines (14-18) convert the predicates to a vector format such that its inner product with the document's polynomial coefficients returns whether or not the document should be returned.

Algorithm 1 Predicate generation when querying for w

```
1:  procedure GENPRED(w)
2:      TupleSet ← ∅
3:      for l = 1 to |h| do
4:          for ctr = 0 to ctr_max do
5:              with prob. p, TupleSet.add([w||l||ctr, l])
6:      for id = 1 to n do
7:          n_FP ← Geo(1 − q)
8:          for k = 1 to n_FP do
9:              TupleSet.add([id||0||-1, h(id)])
10:     for l = 1 to |h| do
11:         n_NM ← Geo(1 − q)
12:         for k = 1 to n_NM do
13:             TupleSet.add([w_-1||-1||0, l])
14:     PredLabelPairs ← ∅
15:     for x, l ∈ TupleSet do
16:         predicate = (x^0, x^1, x^2, ..., x^(S_max+1))
17:         PredLabelPairs.add([predicate, l])
18:     return PredLabelPairs
```

Note that we can get a true positive if the inital coin flip succeeds (probability p) or if it fails but the geometric distribution generates at least one token, i.e., $\mathtt{TPR} = p + (1-p)q$. The probability of false positive is $\mathtt{FPR} = q$.

D. Formal Interface

We formalize OSSE interface following Definition 1 and using the IPPE functions in Definition 8 as follows:

- Keygen(1^λ): takes as input a security parameter λ and returns FHIPPE.Setup(1^λ), which outputs secret key sk.
- BuildIndex$(sk, \mathcal{D})$: takes the secret key sk and the database $\mathcal{D}$, computes the polynomial coefficients $v_i =$ genPoly$(D, id(D))$ for every document $D \in \mathcal{D}$, and then encrypts them by calling FHIPPE.Encrypt(sk, v_i). The encrypted polynomials of all documents form the *search index* I of the database.

- $\mathsf{Trapdoor}(sk, w)$: takes the secret key sk and a keyword w, then calls $\mathsf{genPred}(w)$ to get a set of (predicate, label) tuples. Every predicate f is transformed into a query *token* τ_f using FHIPPE.$\mathsf{GenToken}(sk, f)$, and the output of $\mathsf{Trapdoor}(sk, \mathsf{w})$ is a set of (token, label) pairs Γ_w, that the client sends to the server.
- $\mathsf{Search}(I, \Gamma_w)$: takes as input the search index I and a set of query tokens and labels Γ_w. Then, for every token and label $(\tau_f, l) \in \Gamma_w$, it calls FHIPPE.$\mathsf{Query}(I[i], \tau_f)$, for all document identifiers i whose label is l; if the result of the inner product is 0 (there has been a match), then it returns the corresponding document identifier i. The output of $\mathsf{Search}(I, \Gamma_w)$ is the set of matched document identifiers $\vec{id}$. The server then sends to the client the encrypted documents that correspond with these identifiers (ignoring possible duplicate ids in $\vec{id}$).

E. Leakage Characterization

As explained above, OSSE hides the access and search patterns by adding random false positives, false negatives, and hiding the number of tokens that yield a non-match. In this section, we characterize the leaked access pattern, that we call *obfuscated access pattern*. Then, we define the obfuscated trace, an update of Def. 6 that characterizes the leakage of OSSE. We use these concepts in the security and privacy analysis of OSSE in Sects. VI and VII.

Since OSSE generates tokens independently per query, we study the adversary's observation when the user queries for a particular keyword w. First, the user calls $\mathsf{genPred}$ to generate a sequence of tokens and labels Γ_w, and sends them to the server. The server evaluates each of the tokens $(\tau, l) \in \Gamma_w$ using the search index and either obtains a single match (in a document that has a label l) or a non-match. Therefore, for each token and label pair (τ, l) there are $n+|h|$ possible adversary observations: either document $\mathcal{D}[i]$ (with label $h(id(\mathcal{D}[i])) = l$) satisfied the token ($n$ possibilities), or there was a non-match ($|h|$ possibilities, one for each label). The adversary observes one of these outcomes for each single token received. Therefore, the obfuscated access pattern $\tilde{\Pi}_w$ for this query can be seen as a multi-nary $n + |h|$ vector. We characterize this vector now, following Alg. 1.

First, let's study the value of $\tilde{\Pi}_w[i]$ when $i \in [n]$, i.e., the number of times the adversary observes a match for the ith document. If this document contains w, then lines 3-5 will generate a true match with probability p. Otherwise, these lines will not generate a match for document i. Then, lines 6-9 add false positive matches for each document following a geometric distribution with parameter $1 - q$. The rest of the algorithm just generates extra non-matches. Therefore,

$$\tilde{\Pi}_w[i] \sim \begin{cases} \mathbf{Bern}(p) + \mathbf{Geo}(1-q) & \text{if } w \in \mathcal{D}[i]\,, \\ \mathbf{Geo}(1-q) & \text{if } w \notin \mathcal{D}[i]\,, \end{cases} \quad \text{for } i \in [n]\,. \tag{7}$$

Now, let's study the number of non-matches the adversary sees with label $l \in [|h|]$, i.e., $\tilde{\Pi}_w[n+l]$. In lines 3-5, every token generated with label l will return a non-match except for those documents with label l that contain w. Since each token is generated with probability p, the non-matches with label l follow a binomial distribution with parameters $g_l \doteq \mathtt{ctr}_{\mathtt{max}} - |\mathcal{D}(w)_l|$ and p, where $\mathcal{D}(w)_l$ is the set of ids of documents with label l that contain keyword w. Additionally, lines 10-13 generate a number of non-matches with label l following a geometric distribution with parameter $1 - q$. Therefore,

$$\tilde{\Pi}_w[n+l] \sim \ \mathbf{Bi}(g_l, p) + \mathbf{Geo}(1-q) \quad \text{for } l \in [|h|]\,. \tag{8}$$

We are ready to define the obfuscated access pattern:

Definition 11 (Obfuscated Access Pattern)**.** *The obfuscated access pattern $\tilde{\Pi}_{\vec{w}}$ over a history H_t is a multi-nary matrix of size $t \times n + |h|$ whose ith row $\tilde{\Pi}_{\vec{w}[i]}$ is characterized by* (7) *and* (8)*.*

The obfuscated trace summarizes the leakage of OSSE:

Definition 12 (Obfuscated Trace)**.** *The trace of a history H_t when searching for t keywords $\vec{w}$ in OSSE, called obfuscated trace and denoted by $\tilde{T}$, is*

$$\tilde{T}(H_t) \doteq (id(\mathcal{D}), |\mathcal{D}[1]|, \ldots, |\mathcal{D}[n]|, h, F_{max}, S_{max}, \tilde{\Pi}_{\vec{w}})\,, \tag{9}$$

where $\tilde{\Pi}_{\vec{w}}$ is the obfuscated access pattern *for the query vector $\vec{w}$, h is the hash function used to generate document labels, and F_{max} and S_{max} are the maximum number of documents that contain a particular keyword and the maximum number of keywords per document, respectively.*

F. Choice of Geometric Distribution for False Positives

As we explain above, OSSE adds false positives following a geometric distribution. The reason for this choice is the following: when the adversary observes $k > 1$ matches for a certain document $\mathcal{D}[i]$, they know for sure that at least $k - 1$ are false positives (because there can only be a single true positive). However, since the geometric distribution is memoryless, the number of matches k does not reveal anything more than just observing a single match about whether a keyword w is in $\mathcal{D}[i]$ or not. Mathematically,

$$\frac{\Pr(\tilde{\Pi}_w[i] = 1 | w \in \mathcal{D}[i])}{\Pr(\tilde{\Pi}_w[i] = 1 | w \notin \mathcal{D}[i])} = \frac{\Pr(\tilde{\Pi}_w[i] = k | w \in \mathcal{D}[i])}{\Pr(\tilde{\Pi}_w[i] = k | w \notin \mathcal{D}[i])}\,, \tag{10}$$

for all $k > 1$. Additionally, as we prove in Sect. VII, the geometric distribution allows OSSE to provide differential privacy guarantees.

G. Label Generation Function h and $\mathtt{ctr}_{max}$.

Our protocol assigns a label to each document by hashing its document identifier using a function $h(\cdot)$. Then, when running $\mathsf{genPoly}$ while building the index ($\mathsf{BuildIndex}$), OSSE uses a counter to ensure that no token can match more than one document. The size of the label (denoted $|h|$) determines how large the counter value must be: a small $|h|$ will cause a lot of documents with a particular keyword w to have the same label l, and thus will make $\mathtt{ctr}_{w,l}$ large. However, the maximum value of $\mathtt{ctr}_{w,l}$, denoted $\mathtt{ctr}_{\mathtt{max}}$, must be fixed before outsourcing the database, otherwise the index building process would leak information about keyword frequency per label. If the client commits to a maximum value $\mathtt{ctr}_{\mathtt{max}}$ that is too small, it will not be possible to generate the coefficients such that their roots are all unique unless some keywords are removed from some documents. We call this a $\mathsf{BuildIndex}$ *failure*. In order to avoid a failure and ensure that $\mathsf{BuildIndex}$

succeeds, we set $|h| = F_{\max}$ and $\mathtt{ctr}_{\max} = 3\ln n/\ln\ln F_{\max}$. The following theorem guarantees an overwhelming success probability with these parameters:

Theorem 1. *Let h be a hash function that outputs values uniformly at random in the range $|h| = F_{max}$, i.e., $h : [n] \to [F_{max}]$. Let $\mathtt{ctr}_{max} = 3\ln n/\ln\ln F_{max}$. Then, the index building process* succeeds *with probability $\geq 1 - 1/n$.*

Proof: Refer to Appendix B. ∎

In practice, $\mathtt{ctr}_{\max}$ can be even smaller. In our experiments in Sect. IX, with a real dataset with $n \approx 30\,000$ documents and $F_{\max} \approx 2\,000$, a value $\mathtt{ctr}_{\max} = 7$ is enough to successfully build the index.

VI. Security

In this section, we prove that OSSE is adaptively semantically secure when the underlying FHIPPE is SIM-secure. For this, we show that a simulator that receives the obfuscated trace (Def. 12) can simulate the view of an adversary who is allowed to adaptively issue queries.

Theorem 2. *OSSE is adaptively semantically secure.*

Proof: Let $\mathcal{S}$ be a simulator that sees a partial obfuscated trace $\tilde{T}(H_t^s)$. We show that this simulator can generate a view $(V_t^s)^*$ that is indistinguishable from the actual partial view of the adversary $V_{sk}^s(H_t)$ for all $0 \leq s \leq t$, all polynomial-bounded functions f_p, all probabilistic polynomial-time adversaries $\mathcal{A}$, and all distributions $\mathcal{H}_t$; except with a negligible probability, where $t \in \mathbb{N}, H_t \xleftarrow{R} \mathcal{H}_t$ and $sk \leftarrow \mathsf{Keygen}(1^k)$.

We recall the notions of (partial) view and (partial) obfuscated trace in OSSE:

$$
\begin{aligned}
V_{sk}(H_t) &\doteq (id(\mathcal{D}), \mathcal{E}(\mathcal{D}[1]), \ldots, \mathcal{E}(\mathcal{D}[n]), I, \Gamma_1, \ldots, \Gamma_t)\\
V_{sk}^s(H_t) &\doteq (id(\mathcal{D}), \mathcal{E}(\mathcal{D}[1]), \ldots, \mathcal{E}(\mathcal{D}[n]), I, \Gamma_1, \ldots, \Gamma_s)\\
\tilde{T}(H_t) &\doteq (id(\mathcal{D}), |\mathcal{D}[1]|..|\mathcal{D}[n]|, h, F_{\max}, S_{\max}, \tilde{\Pi}_{\vec{w}[1]}..\tilde{\Pi}_{\vec{w}[t]})\\
\tilde{T}(H_t^s) &\doteq (id(\mathcal{D}), |\mathcal{D}[1]|..|\mathcal{D}[n]|, h, F_{\max}, S_{\max}, \tilde{\Pi}_{\vec{w}[1]}..\tilde{\Pi}_{\vec{w}[s]})
\end{aligned}
$$

First, note that simulating $id(\mathcal{D})$ is trivial since we assumed that $id(\mathcal{D}) = (1, \ldots, n)$. Also, encrypted documents $\mathcal{E}(\mathcal{D}[i])$ are indistinguishable from a random string $e_i^* \xleftarrow{R} \{0,1\}^{|\mathcal{D}[i]|}$ since $\mathcal{E}$ is semantically secure. However, simulating the index I and the search token an label lists Γ_i is non-trivial.

Since the underlying FHIPPE is SIM-secure, there exists a simulator $\mathcal{S}_{\text{FHIPPE}}$ which can simulate FHIPPE, namely $\mathcal{S}_{\text{FHIPPE}}$ has access to an encryption oracle $\mathcal{S}_{\text{FHIPPE}}.\mathsf{Encrypt}$ and a token generation oracle $\mathcal{S}_{\text{FHIPPE}}.\mathsf{GenToken}$ such that the outputs of the two oracles are indistinguishable from those of FHIPPE.$\mathsf{Encrypt}$ and FHIPPE.$\mathsf{GenToken}$ under the restriction that the inner product predicate of each pair of input to FHIPPE.$\mathsf{Encrypt}$ and FHIPPE.$\mathsf{GenToken}$ is equal to that of each pair of input to $\mathcal{S}_{\text{FHIPPE}}.\mathsf{Encrypt}$ and $\mathcal{S}_{\text{FHIPPE}}.\ \mathsf{GenToken}$.

• **Simulate $I[i]$.** At time $s = 0$, $\mathcal{S}$ only holds $\tilde{T}(H_t^0)$ = $(id(\mathcal{D}),\ |\mathcal{D}[1]|, \ldots, |\mathcal{D}[n]|,\ h,\ F_{\max},\ S_{\max})$ by which it generates $I[i]$ as follows: first, $\mathcal{S}$ generates a vector of polynomial coefficients by calling $\mathsf{genPoly}(\perp, i)$ where $\perp$ denotes an empty list of keywords; then, $\mathcal{S}$ calls $\mathcal{S}_{\text{FHIPPE}}.\mathsf{Encrypt}$ ($\mathsf{genPoly}(\perp, i)$) to compute $I^*[i]$. It should be noted that $I^*[i]$ can be matched by a token generated by $\mathcal{S}_{\text{FHIPPE}}.\mathsf{GenToken}$ when inputting $[i^0, i^1, \ldots, i^{S_{\max}+1}]$. I^* can be obtained by combining all $I^*[i]$ together. We claim that I^* is indistinguishable from I which will be proved below.

• **Simulate Γ_k.** At time $s \leq t$, $\mathcal{S}$ knows $\tilde{T}(H_t^s)$, by which it simulates Γ_s as follows (simulations of Γ_k, for $k < s$, can be constructed similarly). Initialize an empty multiset X. For $i \in [n]$, add $\tilde{\Pi}_{\vec{w}[s]}[i]$ copies of $[i||0||\text{-}1, h(i)]$ to X. Then, for $i \in [|h|]$, add $\tilde{\Pi}_{\vec{w}[s]}[n+i]$ copies of $[w_{\text{-}1}||\text{-}1||0, i]$ to X. Run lines 14 onward of Alg. 1 to get $\mathtt{PredLabelPairs}$. Transform each predicate in $\mathtt{PredLabelPairs}$ into a token by calling $\mathcal{S}_{\text{FHIPPE}}.\mathsf{GenToken}$, and the set of (token, label) pairs is Γ_s^*. We prove next that Γ_s^* and Γ_s are indistinguishable.

• **Indistinguishability.** We prove that I^* and Γ_i^* are indistinguishable from I and Γ_i by contradiction: if there exists an adversary $\mathcal{A}$ which can distinguish I^* and Γ_i^* from I and Γ_i, we show that $\mathcal{A}$ breaks SIM-security. Assume $\mathcal{A}$ can distinguish I^* and Γ_i^* from I and Γ_i. Let $\{x_{j,0}\}$ (resp. $\{x_{j,1}\}$) be the underlying secrets of $\{I[j]\}$ (resp. $\{I^*[j]\}$) and $\{y_{j,0}\}$ (resp. $\{y_{j,1}\}$) be the underlying secrets of Γ_i (resp. Γ_i^*). Consider the following two full security games where one is between $\mathcal{A}$ and FHIPPE, denoted by $G_{\mathcal{A},\text{FHIPPE}}$, and the other one is between $\mathcal{A}$ and $\mathcal{S}_{\text{FHIPPE}}$, denoted by $G_{\mathcal{A},\mathcal{S}_{\text{FHIPPE}}}$. In both games, the adversary $\mathcal{A}$ issues the same queries as follows:

1) Ciphertext query. $\mathcal{A}$ queries $(x_{j,0}, x_{j,1}), \forall j \in [n]$.
2) Token query. $\mathcal{A}$ queries $(y_{j,0}, y_{j,1}), \forall j \in [|\Gamma_i|]$ where $|\Gamma_i|$ means the number of tokens in Γ_i.

In $G_{\mathcal{A},\text{FHIPPE}}$, with probability $1/2$, $b = 0$ and FHIPPE outputs I' and Γ_i' (since FHIPPE is probabilistic) which are indistinguishable from I and Γ_i. Likewise, in $G_{\mathcal{A},\mathcal{S}_{\text{FHIPPE}}}$, with probability $1/2$, $b = 1$ and $\mathcal{S}_{\text{FHIPPE}}$ outputs $I^{*\prime}$ and $\Gamma_i^{*\prime}$ (since $\mathcal{S}_{\text{FHIPPE}}$ is probabilistic) which are indistinguishable from I^* and Γ_i^*.

Since $\mathcal{A}$ can distinguish I^* and Γ_i^* from I and Γ_i, it must be able to distinguish I' and Γ_i' from $I^{*\prime}$ and $\Gamma_i^{*\prime}$, namely $\mathcal{A}$ can distinguish the game with FHIPPE, denoted by $G_{\mathcal{A},\text{FHIPPE}}$ (the real world) from the game with $\mathcal{S}_{\text{FHIPPE}}$, denoted by $G_{\mathcal{A},\mathcal{S}_{\text{FHIPPE}}}$ (the ideal world). This contradicts the fact FHIPPE is SIM-secure. Therefore, $\mathcal{A}$ cannot distinguish I^* and Γ_i^* from I and Γ_i and thus OSSE is adaptively semantically secure.

∎

VII. Differential Privacy Analysis and Discussion

In this section, we compare OSSE and the proposal by Chen et al. [10], henceforth termed CLRZ, [1] under the differential privacy framework. We chose this framework since it is one of the most widely accepted notions of privacy. We first propose differential privacy for documents and keywords, which generalize previous SSE-related differential privacy notions [10]. We explain that they imply access and search pattern protection, respectively. Then, we prove the differential privacy guarantees of OSSE and compare them with CLRZ. We show that none of these mechanisms can simultaneously achieve

[1] This acronym takes the first letter of each author of the paper [10].

high theoretical privacy guarantees while providing high utility. In Sect. IX we show that these mechanisms offer adequate protection even when differential privacy deems them as weak, since practical attacks widely differ from the strong implicit adversary assumed in the differential privacy framework.

A. Differential Privacy Definitions

Differential privacy [15] (DP) is one of the most broadly accepted theoretical notions of privacy, and has been applied to different areas of privacy, including database privacy [10], [35], [36]. Differential privacy is characterized by a privacy parameter ϵ, and ensures that neighboring inputs to the privacy-preserving algorithm produce any given output with similar probability, and thus it is hard to gain information about the secret input given any observation. Chen et al. [10] define differential privacy for access pattern leakage. In the definitions below, we generalize their definition and also instantiate it to account for search-pattern privacy. Here, we represent an SSE scheme as a function $\mathcal{SE}$ that takes the client input (a dataset $\mathcal{D}$ and a query vector $\vec{w}$) and outputs the leakage (a trace T).

Definition 13 (Differential Privacy for Documents)**.** *An SSE characterized by function $\mathcal{SE}$ provides ϵ-DP for* documents *iff for any keyword list of length t, $\vec{w} \in \Delta^t$, for any pair of neighboring databases $\mathcal{D}, \mathcal{D}' \in 2^{2^\Delta}$ (they differ in exactly one position i and exactly one keyword w, i.e., w is in either $\mathcal{D}[i]$ or $\mathcal{D}'[i]$ but not both), and any trace T, the following holds:*

$$Pr[\mathcal{SE}(\mathcal{D}, \vec{w}) = T] \leq e^{t \cdot \epsilon} Pr[\mathcal{SE}(\mathcal{D}', \vec{w}) = T]\,. \qquad (11)$$

We chose to call this notion "differential privacy *for documents*" instead of "for access patterns", since the hidden variable in this definition is the database that contains the documents, $\mathcal{D}$. Intuitively, satisfying the above definition guarantees that no one can determine if a document contains a keyword given the trace. This implies that an SSE that provides ϵ-DP for documents also provides *access pattern privacy*.

Definition 14 (Differential Privacy for Keywords)**.** *An SSE characterized by function $\mathcal{SE}$ provides ϵ-DP for* keywords *iff for any database $\mathcal{D} \in 2^{2^\Delta}$, for any pair of neighboring keyword lists $\vec{w}, \vec{w}' \in \Delta^{|\vec{w}|}$ ($\vec{w}$ and $\vec{w}'$ differ in only one element, $w[i] \neq w'[i]$), and any trace T, the following holds:*

$$Pr[\mathcal{SE}(\mathcal{D}, \vec{w}) = T] \leq e^{d \cdot \epsilon} Pr[\mathcal{SE}(\mathcal{D}, \vec{w}') = T]\,. \qquad (12)$$

Here, d is the number of different documents between $\mathcal{D}(\vec{w}[i])$ and $\mathcal{D}(\vec{w}'[i])$.

A scheme that provides DP for keywords guarantees that no one can determine, through observing the trace (allowed leakage), whether a client is searching for one keyword list or the other. This in turn implies *search pattern privacy*, i.e., it prevents the server from guessing whether or not two queries were performed for the same keyword.

B. Comparison of Differential Privacy Guarantees in SSE

We compare OSSE and CLRZ in terms of differential privacy. We leave ORAM and PIR schemes out of this comparison since, even though they hide the location of individual documents retrieved, they would require adding differentially private noise to the number of documents retrieved per query; doing this, on top of their already high overhead, would make them impractical.

	DP for documents	**DP for keywords**
CLRZ	$\epsilon = \ln\left(\max\left\{\frac{\texttt{TPR}}{\texttt{FPR}}, \frac{1-\texttt{FPR}}{1-\texttt{TPR}}\right\}\right)$	None ($\epsilon = \infty$)
OSSE	$\epsilon = \ln\left(\frac{\texttt{TPR}}{\texttt{FPR}}\frac{1-\texttt{FPR}}{1-\texttt{TPR}}\right)$	$\epsilon = \ln\left(\frac{\texttt{TPR}}{\texttt{FPR}}\frac{1-\texttt{FPR}}{1-\texttt{TPR}}\right)$

TABLE III: Differential Privacy (DP) guarantees.

Theorem 3. *OSSE provides ϵ-differential privacy for documents and keywords, where $\epsilon = \ln\left(\frac{TPR}{FPR}\frac{1-FPR}{1-TPR}\right)$, with $TPR = p + (1-p)q$ and $FPR = q$.*

The proof is in Appendix A. Table III compares the ϵ values provided by OSSE and CLRZ in terms of their true positive and false positive rates (TPR and FPR). The table shows that CLRZ usually provides stronger DP for documents than OSSE (larger ϵ indicates weaker privacy guarantees). On the contrary, OSSE's guarantee for protecting access patterns is weaker, but it also provides differential privacy for keywords, which implies a certain level of search pattern privacy.

Achieving high privacy regimes requires undesirable TPR and FPR values for both schemes. For example, achieving $\epsilon = 1$ with a moderately high $\texttt{TPR} = 80\%$ requires $\texttt{FPR} \approx 45\%$ for CLRZ and $\texttt{FPR} \approx 60\%$ for OSSE, which is prohibitive in terms of bandwidth. However, if the client wants to get a random subset of $\texttt{TPR} = 30\%$ of the documents that match a keyword, they can achieve high privacy ($\epsilon = 1$) with only $\texttt{FPR} = 11\%$ (resp. 13%) with CLRZ (resp. OSSE). In this high-privacy low-utility setting, CLRZ and OSSE offer similar DP for documents, but OSSE additionally provides DP for keywords (i.e., search pattern protection).

There are cases where TPR must be kept close to one and FPR close to zero for performance issues (e.g., false positives imply a bandwidth increase). The fact that OSSE cannot provide a low $\epsilon < 1$ in these cases does not mean it is not effective at deterring attackers. In fact, recent work [13] has shown (for an ORAM-based defense technique) that even a small amount of noise can seriously harm current database and query recovery attacks. We confirm that this is also true for OSSE against different attacks in Section IX. Indeed, we show that $\texttt{TPR} = 0.9999$ and $\texttt{FPR} = 0.025$, which gives a large $\epsilon \approx 13$ for OSSE, is enough to deter different attacks [8], [20], [27], [31]. CLRZ, however, is significantly more vulnerable to these attacks since it does not provide any search pattern protection. The explanation for this high protection despite low differential privacy guarantees is that differential privacy assumes a worst-case scenario where the adversary either knows every single entry in the dataset except for one, or knows all of the user queries but one. Assuming this strong adversary is unrealistic in most cases and, besides that, there exist practical attacks [6], [8], [20], [27], [31] that do not require an adversary this powerful.

VIII. Complexity Analysis

We study the communication and computation complexity of OSSE. Note that OSSE does not require client-side storage after outsourcing the encrypted database and the search index, other than for storing parameters and keys. We omit the

Notation	Description
$F_{\mathtt{max}}$	Maximum keyword frequency (maximum number of documents that contain any given keyword).
$S_{\mathtt{max}}$	Maximum no. of distinct keywords in a document.
$\mathtt{ctr}_{\mathtt{max}}$	Maximum value of $\mathtt{ctr}$ allowed.
τ_{size}	Size of a query token and label pair (e.g., bytes).
D_{size}	Maximum size of a document (e.g., bytes).

TABLE IV: Notation for the Complexity Analysis

initialization from the complexity analysis, since it is only done once. Table IV summarizes the notation of this section.

A. Communication Overhead

The communication complexity refers to the total number of tokens sent from client to server when querying for a keyword w, and the number of documents that the server returns to the client as a response. We study the communication overhead, i.e., the *average* communication cost of OSSE compared to the average cost of a standard SSE scheme. Let $E_w \doteq \mathrm{E}\{\mathcal{D}(w)\}$ be the expected number of documents that contain keyword w. We assume that q is small enough so that $n \cdot q < F_{\mathtt{max}}$, and use $|h| = F_{\mathtt{max}}$ and $\mathtt{ctr}_{\mathtt{max}} = 3 \ln n / \ln \ln F_{\mathtt{max}}$.

First, when querying for keyword w, the client calls genPred (Alg. 1) to generate tokens. The number of tokens generated in the first loop (lines 3-5) is the sum of $|h| \cdot \mathtt{ctr}_{\mathtt{max}}$ Bernoulli distributions with parameter p, while the remaining two loops generate a number of tokens that is the sum of n and $|h|$ Geometric distributions with parameter $1-q$. Therefore, the *expected* number of tokens sent from client to server is

$$\begin{aligned}\#\text{tok} &= |h| \cdot \mathtt{ctr}_{\mathtt{max}} \cdot p + (n + |h|) \cdot \frac{q}{1-q} \\ &\approx |h| \cdot \mathtt{ctr}_{\mathtt{max}} \cdot p + n \cdot q < F_{\mathtt{max}} \cdot (\mathtt{ctr}_{\mathtt{max}} + 1)\,,\end{aligned}$$

where we have used that $n \gg |h|$ and $\frac{q}{1-q} \approx q$ since we are assuming that q is close to zero.

Since the server replies with the documents that are matched *at least once*, the expected number of documents returned is

$$\#\text{doc} = E_w(p+q-pq)+(n-E_w)q \leq E_w p + nq < E_w + F_{\mathtt{max}}\,. \tag{13}$$

The total communication cost of OSSE is $\#\text{tok} \cdot \tau_{\text{size}} + \#\text{doc} \cdot D_{\text{size}}$. A standard SSE just sends a single query message to the server and receives, on average, E_w documents. Therefore, its cost is $\approx E_w \cdot D_{\text{size}}$. Dividing these two expressions and using some basic algebra, we can express the *communication overhead* of OSSE as

$$\text{COMM-OVER}_{\text{OSSE}} < \frac{F_{\mathtt{max}}}{E_w}\left((\mathtt{ctr}_{\mathtt{max}} + 1) \cdot \frac{\tau_{\text{size}}}{D_{\text{size}}} + 1\right) + 1\,. \tag{14}$$

In many cases, $(\mathtt{ctr}_{\mathtt{max}} + 1) < D_{\text{size}}/\tau_{\text{size}}$ (for example, if the database contains images, the encrypted documents can easily be 10 times larger than the token size). In that case, the cost is just $\text{COMM-OVER}_{\text{OSSE}} < 2 \cdot F_{\mathtt{max}}/E_w + 1$.

This cost depends on E_w, which in turn depends on the query and keyword distribution. We perform the analysis for three possible distributions: uniform, Zipfian (typical in natural language [40]), and the worst-case distribution for OSSE that considers that the queried keywords appear in a single document. We show how to compute E_w for each of these distributions in Appendix C. We get

$$\text{COMM-OVER}_{\text{OSSE}} < \begin{cases} 3\,, & \text{uniform,} \\ 1.36 + 0.61 \cdot \log|\Delta|\,, & \text{Zipf,} \\ 1 + 2 \cdot F_{\mathtt{max}}\,, & \text{worst-case.} \end{cases} \tag{15}$$

The cost is increased by a factor no bigger than $\mathtt{ctr}_{\mathtt{max}}$+1 for datasets where each document size is comparable to the size of each keyword.

B. Computational Complexity

We disregard the initialization cost (generating the index) since it is only done once at the beginning of the protocol. We measure the computational complexity of a query as the number of evaluations that the server needs to compute when searching for a keyword w. Every time the server receives a token, it calls the function FHIPPE.Query on average $n/|h|$ times (since this is the average number of documents with a given label). Therefore, the number of calls to FHIPPE.Query is

$$\text{COMP-COMP}_{\text{OSSE}} = \#\text{tok} \cdot \frac{n}{|h|} < n \cdot (\mathtt{ctr}_{\mathtt{max}} + 1)\,. \tag{16}$$

C. Faster OSSE

OSSE uses a hash function h to assign a label to each document. This requires a certain $\mathtt{ctr}_{\mathtt{max}}$ value to ensure that no token can match more than one document. We explain a variation of the algorithm that assigns labels to documents that achieves a lower $\mathtt{ctr}_{\mathtt{max}}$ and therefore reduces the communication and computational cost of OSSE. In this variation, there are two publicly known hash functions h_1 and h_2. When building the search index, the client generates roots $(w||l||\mathtt{ctr}_{w,l})$ by choosing l between $h_1(id(D))$ and $h_2(id(D))$ as the one that minimizes the counter. This way, the value of $\mathtt{ctr}_{\mathtt{max}}$ is only $O(\ln \ln F_{\mathtt{max}})$ [4]. This reduction of $\mathtt{ctr}_{\mathtt{max}}$ can provide significant computational and communication advantages, but at the cost of a considerable reduction of the differential privacy guarantees (we omit this analysis here for space constraints). This modification of OSSE is still highly useful when differential privacy protection is not required (e.g., against known attacks).

D. Comparison with Other Schemes

The only searchable ORAM-based SSE scheme is TWORAM [17], which has a communication overhead of $O(\log n \log\log n)$ (at least four rounds) and client-side storage of $O(\log^2 n)$. As explained above, under some realistic assumptions on the query and keyword distribution, OSSE requires less communication cost. Also, OSSE does not require client-side storage except for storing parameters like p and q, and only uses one communication round.

Even though Fully Homomorphic Encryption (FHE) schemes are competitive performance-wise [1], [2], they can only return a fixed, constant number of results on each query due to the fixed length of each circuit. Adapting FHE schemes

to variable length results requires multiple communication rounds, whereas our scheme requires a single round.

The only practical privacy-preserving SSE scheme we are aware of is CLRZ [10]. In CLRZ, the client sends a single query token, and receives on average the same amount of documents from the server as in OSSE. [2] Thus, the communication overhead of CLRZ is slightly smaller than that of OSSE. The computational cost of CLRZ depends on the underlying SSE that it implements; when combined with an inexpensive SSE, CLRZ can achieve less overhead than OSSE.

IX. Evaluation

We evaluate the effectiveness and efficiency of OSSE. First, we evaluate OSSE and CLRZ [10] against different query recovery attacks:

1) Liu et al.'s *frequency attack* [27], that recovers queries using query frequency information and search patterns.
2) The *IKK attack* [20], that recovers the queries using keyword co-occurrence data, ground-truth information of a subset of the client queries, and access and search patterns. Chen et al. use this attack to evaluate CLRZ [10].
3) Cash et al.'s *count attack* [8], that builds sets of candidate keywords for each query based on the query response volume, and then refines these sets by using co-occurrence information until only one feasible assignment remains.
4) The *graph matching* attack by Pouliot et al. [31], that uses keyword co-ocurrence information and does not require any ground-truth information about the queries or data set to be effective.

Blackstone et al. [6] recently proposed a query recovery attack (*subgraph attack*) that uses access pattern leakage and partial database knowledge. The attack tries to match queries to keywords by identifying patterns from the partial database knowledge in the observed access patterns. Since CLRZ and OSSE randomize the access patterns, adapting the attack against these defenses is not trivial, and therefore we do not include it in our evaluation.

We run CLRZ without using its document redundancy technique [10]: this technique aims at reducing false negatives and is trivially compatible with OSSE; we decide not to use it for simplicity and because we believe it does not affect performance results. For a fair evaluation, we *adapt* the attacks above to perform well against CLRZ and OSSE. We explain the details in Appendix D.[3]

Second, we report the running time of a prototypical implementation of OSSE in Python on an Intel(R) E7-8870 160-core Ubuntu 16.04 machine clocked at 2.40 GHz with 2 TB of system memory.[4]

We use the Enron email dataset in our experiments,[5] which contains 30 109 emails. We follow the keyword extraction process in related work [20], then we ignore keywords that do not appear in the English dictionary and remove stopwords; we use the following $|\Delta|$ most common keywords remaining as keyword universe ($|\Delta|$ varies between experiments).

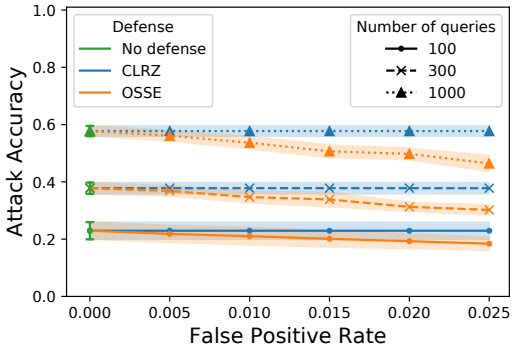

Fig. 1: Accuracy of Query Frequency Attack

A. Query Frequency Attack

We implement and run the first frequency attack by Liu et al. [27]. We take $|\Delta| = 250$ keywords and download their query frequency over 50 consecutive weeks[6] from Google Trends.[7] We store this frequency information in a 50×250 matrix F where $F[i, j]$ is the probability that the client queries for the jth keyword in the ith week. The client generates $N_q \in \{100, 300, 1\,000\}$ queries per week following the distributions in F. We evaluate the case where the client uses no defense, when they use CLRZ, and when they use OSSE. The server observes the (possibly obfuscated) access patterns and groups them into N_c clusters using a k-means algorithm (this is trivial for CLRZ). We give the adversary the *true* number of distinct keywords queried by the client to use as N_c. This is a worst-case against OSSE. The adversary labels all queries in cluster $k \in [N_c]$ as the jth keyword, where j is the column in F that is closest, in Euclidean distance, to the frequency trend of the cluster over the 50 weeks.

Figure 1 shows the attack accuracy (number of correctly identified queries divided over the total number of queries) averaged over 20 runs. The shaded areas are the 95% confidence intervals for the mean. We vary the $\texttt{FPR}$ and query number N_q, and fix $\texttt{TPR} = 0.9999$. Since clustering access patterns (i.e., finding the search pattern) is trivial for CLRZ, the accuracy against this defense is independent of $\texttt{FPR}$. OSSE achieves higher protection against the attack since it obfuscates the access patterns of each query independently.

B. IKK Attack

We run the IKK attack [20] using a keyword universe size $|\Delta| = 500$. This attack relies on the keyword co-occurrence, i.e., a $|\Delta| \times |\Delta|$ matrix M where $M[i, j]$ is the percentage of documents that have both the ith and jth keyword. The attack computes this matrix from a training set, and compares it with a co-occurrence matrix computed from the observed

[2] Chen et al. [10] alleviate the false negatives by introducing document redundancy, but we do not consider this here for a fair comparison. Our proposal is also compatible with this document redundancy approach.

[3] Our evaluation code is available at https://github.com/simon-oya/NDSS21-osse-evaluation

[4] Our prototype implementation is available at https://github.com/z6shang/OSSE

[5] https://www.cs.cmu.edu/~enron/

[6] The last week we take is the second week of April, 2020.

[7] trends.google.com

access patterns, M'. Unlike in the original IKK attack setting, where the queries are unique, we allow repeated queries and modify the attack accordingly. We also adapt the attack to perform better against the obfuscated access patterns generated by OSSE and CLRZ (see Appendix D). We call this improved version IKK*. In each experiment, the client generates $N_q = 400$ queries (using a Zipfian distribution), we give each one with probability 0.15 to the attacker. We use the same Enron data for training and testing [20].[8]

Figure 2 shows the query recovery rate, averaged over 20 runs, for $\texttt{TPR} = 0.9999$ and different $\texttt{FPR}$ values. We plot the average baseline 15% ground-truth queries known to the attacker as reference. We see that IKK* improves over IKK, as expected, and that OSSE reduces the attack accuracy almost by half compared to CLRZ. This is due to two facts: first, OSSE hides the search patterns and makes it hard for the adversary to use the known queries (15%) towards identifying other queries. Second, since each access pattern is generated independently in OSSE, the search space for IKK against this attack is of size $|\Delta|^{200}$, which is much larger than $|\Delta|!/(|\Delta| - 200)!$, the size of the search space in CLRZ.

C. Count Attack

The count attack [8] uses volume information to build a set of candidate keywords for each query and refines them with co-occurrence information until only one possible matching remains. We optimize the attack against OSSE and CLRZ by modifying a generalization of the attack found in the original paper [8] (see Appendix D). We give the adversary the plaintext database to build the auxiliary volume and co-occurrence information. However, the adversary does not see any ground-truth query. We run the experiments for the same parameters as IKK ($|\Delta| = 500$, $N_q = 400$, Zipfian distribution for queries). When the algorithm fails to find any plausible matching, we set the accuracy to $1/|\Delta|$ (random guessing).

Figure 3 shows the accuracy of the count attack (20 runs) as well as the frequency of failure (inconsistency rate) against OSSE. The count attack achieves perfect accuracy when no defense is applied, but its accuracy already decreases to 0.65 against CLRZ with $\texttt{TPR} = 0.9999$ and $\texttt{FPR} = 0$. The accuracy remains stable against CLRZ as $\texttt{FPR}$ increases, but it sharply decreases against OSSE, mostly due to inconsistencies. These inconsistencies stem from the fact that the heuristics of the count attack require that the observed volume and co-occurrences of queries fall inside certain expected intervals. OSSE generates freshly random access patterns, so the probability that at least one observed volume falls outside an interval is high, which explains the high inconsistency rate.

D. Graph Matching Attack

We implement the graph matching attack [31] using the PATH algorithm [38]. We randomly split Enron emails evenly into a training and testing set, and give the adversary the training set only (we redo the split in each run). The adversary does not have any ground truth information about the queries. This is a more realistic scenario than considered by IKK and the count attack. We take $|\Delta| = 250$ keywords instead of 500, since this attack is computationally demanding. This attack requires a large number of queries to work properly, since it has imperfect information about the dataset. We use $N_q = 2\,000$ queries (Zipfian distribution).[9]

Figure 4 shows the query recovery rate of the graph matching attack, averaged over 100 runs, for $\texttt{TPR} = 0.9999$ and different $\texttt{FPR}$ values. The attack achieves an accuracy of $\approx 99\%$ against CLRZ, which is surprisingly large considering that the adversary has imperfect information. The performance against OSSE drops below 50% with only 0.5% false positives and keeps decreasing as $\texttt{FPR}$ grows.

# cores	BuildIndex (min)	Trapdoor (s)	Search (min)
4	272.5	580.7	1099.1
8	136.3	290.5	549.6
16	68.2	145.3	274.8
32	34.1	72.8	137.4
64	17.1	36.4	68.7
128	8.5	18.2	34.4
160	6.9	14.7	27.5

TABLE V: Running Times

E. Running Time

We measure the average running time of the OSSE functions BuildIndex, Trapdoor, and Search on Enron dataset. We use the function-hiding inner product encryption scheme by Kim et al. [25] with the $\texttt{MNT159}$ pairing curve. We limit the number of keywords in each document to $S_{\max} = 300$ by splitting large documents into smaller ones. By default, 97% of the documents have no more than 300 keywords. After splitting, the number of documents increases to $n = 30\,562$ and $F_{\max} \approx 2\,000$. With the single hashing method, we get $\texttt{ctr}_{\max} = 7$, while the dual hashing method explained in Sect. VIII-C yields $\texttt{ctr}_{\max} = 3$. We evaluate the running time of the latter. We use $p = 0.9999$ and $q = 0.01$.

Table V summarizes the running times. Chen et al. [10] report running times for CLRZ in Enron dataset around 100 seconds for obfuscating the search index, and less than 200 milliseconds per query. Even though the running times of OSSE in our experiment (e.g., half an hour for a search) can be reduced by switching from Python to a more efficient language and parallelizing the queries, our scheme is substantially slower than CLRZ. In exchange, OSSE provides search pattern obfuscation, a rare property that we have shown provides significant protection against different query recovery attacks.

F. Summary of Results

Our experiments reveal that OSSE achieves higher privacy protection than CLRZ against a variety of query recovery attacks. This includes a search pattern-based attack [27] and access pattern-based attacks with ground-truth information about queries and dataset (IKK [20]), ground-truth information about the dataset (count [8]), and no ground-truth information (graph matching [31]). The advantage of OSSE over CLRZ comes from the fact that OSSE generates each access pattern independently at random, which hides the search pattern. Interestingly, our evaluation shows that this undermines attacks

[8] We use the following attack-specific parameters: initial temperature 200, cooling rate 0.9999, and reject threshold 1 500.

[9] The objective function of the graph matching attack has an hyperparameter $\alpha \in [0, 1]$. We used $\alpha = 0$ since it yielded the highest accuracy.

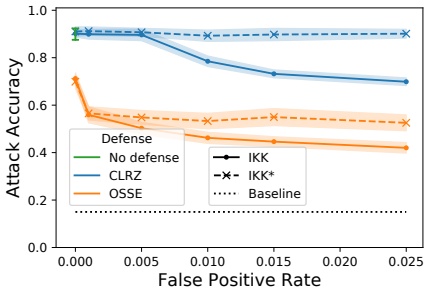

Fig. 2: Accuracy of IKK Attack

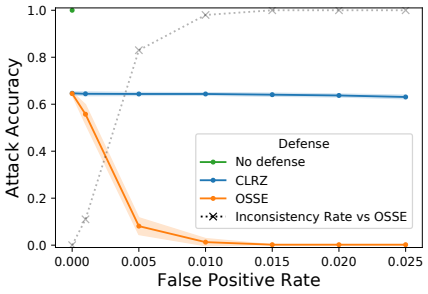

Fig. 3: Accuracy of Count Attack

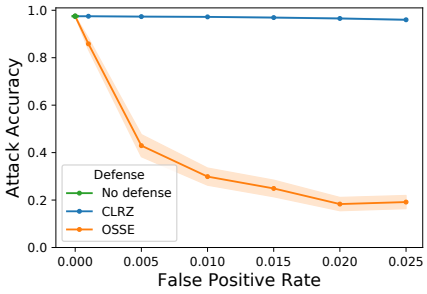

Fig. 4: Accuracy of Graph Matching

that rely on access pattern leakage. This is because these attacks [8], [20], [31] formulate the query recovery problem by implicitly assuming the adversary can distinguish between queries for distinct keywords (i.e., search pattern leakage). When this is not true, the adversary needs to increase their search space (IKK) or perform clustering before running the attack (frequency attack, count attack, and graph matching). This causes an extra source of error for the adversary.

All these privacy benefits come at the cost of a high running time. Our scheme is therefore adequate when the data owner values privacy over running time, and when the system is bandwidth-constrained and cannot afford to use multi-round bandwidth-demanding schemes such as ORAM-based ones [17].

X. Conclusions

Searchable Symmetric Encryption (SSE) allows a data owner to outsource its data to a cloud server while maintaining the ability to search over it. Existing SSE schemes either prevent leakage but are prohibitive in terms of their cost, or are efficient but extremely vulnerable to query and database identification attacks. In 2018, Chen et al. proposed a middle-ground solution that is cost efficient and partially protects which documents match each query, i.e., the access pattern. However, this scheme leaks whether or not the same keyword is being searched multiple times, i.e., the search pattern.

In this work, we propose OSSE, an adaptively semantically secure SSE scheme that obfuscates both access and search patterns by generating each query randomly and independently of previous queries. We prove that the communication overhead of OSSE can be a small constant when the keyword distribution in the dataset is uniform, and $O(\log|\Delta|)$, where $|\Delta|$ is the keyword universe, when the keyword and query distributions are Zipfian. Although it has a large computation complexity, OSSE is easily parallelizable, and performs a search in a single communication round. Our evaluation shows that OSSE is highly effective against current query identification attacks while providing high utility, and demonstrates the importance of hiding search patterns in privacy-preserving SSE schemes.

Acknowledgment

We gratefully acknowledge the support of NSERC for grants RGPIN-05849, CRDPJ-531191, IRC-537591 and the Royal Bank of Canada for funding this research. Andreas Peter was supported by the Netherlands Organisation for Scientific Research (Nederlandse Organisatie voor Wetenschappelijk Onderzoek, NWO) in the context of the SHARE project. This work benefited from the use of the CrySP RIPPLE Facility at the University of Waterloo.

References

[1] A. Akavia, D. Feldman, and H. Shaul, "Secure search on encrypted data via multi-ring sketch," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 985–1001.

[2] A. Akavia, C. Gentry, S. Halevi, and M. Leibovich, "Setup-free secure search on encrypted data: Faster and post-processing free," *Proceedings on Privacy Enhancing Technologies*, vol. 2019, no. 3, pp. 87–107, 2019.

[3] D. Apon, J. Katz, E. Shi, and A. Thiruvengadam, "Verifiable oblivious storage," in *International Workshop on Public Key Cryptography*. Springer, 2014, pp. 131–148.

[4] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal, "Balanced allocations," *SIAM journal on computing*, vol. 29, no. 1, pp. 180–200, 1999.

[5] A. Bishop, A. Jain, and L. Kowalczyk, "Function-hiding inner product encryption," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2015, pp. 470–491.

[6] L. Blackstone, S. Kamara, and T. Moataz, "Revisiting leakage abuse attacks," p. TBD, 2020.

[7] R. Bost, "$\sum$ oφoς: Forward secure searchable encryption," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 1143–1154.

[8] 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*. ACM, 2015, pp. 668–679.

[9] 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*. Springer, 2013, pp. 353–373.

[10] G. Chen, T.-H. Lai, M. K. Reiter, and Y. Zhang, "Differentially private access patterns for searchable symmetric encryption," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 810–818.

[11] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private information retrieval," in *Proceedings of IEEE 36th Annual Foundations of Computer Science*. IEEE, 1995, pp. 41–50.

[12] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," *Journal of Computer Security*, vol. 19, no. 5, pp. 895–934, 2011.

[13] I. Demertzis, D. Papadopoulos, C. Papamanthou, and S. Shintre, "SEAL: Attack mitigation for encrypted databases via adjustable leakage," 2020.

[14] S. Devadas, M. van Dijk, C. W. Fletcher, L. Ren, E. Shi, and D. Wichs, "Onion oram: A constant bandwidth blowup oblivious ram," in *Theory of Cryptography Conference*. Springer, 2016, pp. 145–174.

[15] C. Dwork, "Differential privacy: A survey of results," in *International conference on theory and applications of models of computation*. Springer, 2008, pp. 1–19.

[16] C. Dwork, A. Roth *et al.*, "The algorithmic foundations of differential privacy," *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014.

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

[18] E.-J. Goh *et al.*, "Secure indexes." *IACR Cryptology ePrint Archive*, vol. 2003, p. 216, 2003.

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

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

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

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

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

[24] J. Katz, A. Sahai, and B. Waters, "Predicate encryption supporting disjunctions, polynomial equations, and inner products," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2008, pp. 146–162.

[25] S. Kim, K. Lewi, A. Mandal, H. Montgomery, A. Roy, and D. J. Wu, "Function-hiding inner product encryption is practical," in *International Conference on Security and Cryptography for Networks*. Springer, 2018, pp. 544–562.

[26] K. G. Larsen and J. B. Nielsen, "Yes, there is an oblivious ram lower bound!" in *Annual International Cryptology Conference*. Springer, 2018, pp. 523–542.

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

[28] T. Moataz, T. Mayberry, and E.-O. Blass, "Constant communication oram with small blocksize," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 862–873.

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

[30] M. Naveed, M. Prabhakaran, and C. A. Gunter, "Dynamic searchable encryption via blind storage," in *Security and Privacy (SP), 2014 IEEE Symposium on*. IEEE, 2014, pp. 639–654.

[31] D. Pouliot and C. V. Wright, "The shadow nemesis: Inference attacks on efficiently deployable, efficiently searchable encryption," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 1341–1352.

[32] E. Shen, E. Shi, and B. Waters, "Predicate privacy in encryption systems," in *Theory of Cryptography Conference*. Springer, 2009, pp. 457–473.

[33] 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*. IEEE, 2000, pp. 44–55.

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

[35] R. R. Toledo, G. Danezis, and I. Goldberg, "Lower-cost ϵ-private information retrieval," *Proceedings on Privacy Enhancing Technologies*, vol. 2016, no. 4, pp. 184–201, 2016.

[36] S. Wagh, P. Cuff, and P. Mittal, "Root oram: A tunable differentially private oblivious ram," *arXiv preprint arXiv:1601.03378*, 2016.

[37] X. Wang, H. Chan, and E. Shi, "Circuit oram: On tightness of the Goldreich-Ostrovsky lower bound," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 850–861.

[38] M. Zaslavskiy, F. Bach, and J.-P. Vert, "A path following algorithm for the graph matching problem," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 12, pp. 2227–2242, 2008.

[39] Y. Zhang, J. Katz, and C. Papamanthou, "All your queries are belong to us: The power of file-injection attacks on searchable encryption." in *USENIX Security Symposium*, 2016, pp. 707–720.

[40] G. K. Zipf, "Selected studies of the principle of relative frequency in language," 1932.

APPENDIX

A. Differential Privacy Guarantees of OSSE

Before deriving the proof for Theorem 3, we introduce some results on ratios of probability distributions.

1) Upper Bounds on Probability Ratios: Let $G \sim \mathbf{Geo}(1-q)$ be a random variable that follows a geometric distribution defined over $\{0, 1, 2, \dots\}$, let $A \sim \mathbf{Bern}(p)$ be a Bernoulli random variable, and let $B_n \sim \mathbf{Bi}(n, p)$ be a binomial random variable. We prove the following results.

Lemma 1. *For any non-negative integer $\alpha \in \mathbb{Z}_+$,*

$$\frac{\Pr(G+A=\alpha)}{\Pr(G=\alpha)} \leq \frac{p+q(1-p)}{q}, \tag{17}$$

$$\frac{\Pr(G=\alpha)}{\Pr(G+A=\alpha)} \leq \frac{1}{1-p}. \tag{18}$$

Proof: The ratio in (17) takes the following values depending on α:

$$\frac{\Pr(G+A=\alpha)}{\Pr(G=\alpha)} = \begin{cases} \frac{(1-p)(1-q)}{1-q} = 1-p, & \text{if } \alpha = 0; \\ \frac{pq^{\alpha-1}(1-q)+(1-p)q^{\alpha}(1-q)}{q^{\alpha}(1-q)} = \frac{p+q(1-p)}{q}, & \text{if } \alpha > 0. \end{cases}$$

The bounds in (18) and (17) follow from the fact that $1-p < \frac{p+q(1-p)}{q}$. ∎

Lemma 2. *For any non-negative integers $\alpha, n \in \mathbb{Z}_+$,*

$$\frac{\Pr(G+B_{n+1}=\alpha)}{\Pr(G+B_n=\alpha)} \leq \frac{p+q(1-p)}{q}, \tag{19}$$

$$\frac{\Pr(G+B_n=\alpha)}{\Pr(G+B_{n+1}=\alpha)} \leq \frac{1}{1-p}. \tag{20}$$

Proof: First, we expand the ratio:

$$\frac{\Pr(G+B_{n+1}=\alpha)}{\Pr(G+B_n=\alpha)} \tag{21}$$

$$= \frac{\sum_{\beta=0}^{\min(\alpha,n+1)} \Pr(B_{n+1}=\beta) \cdot \Pr(G=\alpha-\beta)}{\sum_{\beta=0}^{\min(\alpha,n)} \Pr(B_n=\beta) \cdot \Pr(G=\alpha-\beta)} \tag{22}$$

$$= \frac{\sum_{\beta=0}^{\min(\alpha,n+1)} \binom{n+1}{\beta} p^{\beta}(1-p)^{n-\beta+1} q^{\alpha-\beta}(1-q)}{\sum_{\beta=0}^{\min(\alpha,n)} \binom{n}{\beta} p^{\beta}(1-p)^{n-\beta} q^{\alpha-\beta}(1-q)}. \tag{23}$$

We use $\binom{n+1}{k} = \binom{n}{k} + \binom{n}{k-1}$ with the convention that $\binom{n}{k} = 0$ when $k < 0$ or $k > n$, and we perform a change of

variable $\beta' \doteq \beta - 1$ in the numerator. Then, (23) equals:

$$= 1 - p + \frac{p}{q} \cdot \frac{\sum_{\beta'=0}^{\min(\alpha-1,n)} \binom{n}{\beta'} p^{\beta'} (1-p)^{n-\beta'} q^{\alpha-\beta'} (1-q)}{\sum_{\beta=0}^{\min(\alpha,n)} \binom{n}{\beta} p^{\beta} (1-p)^{n-\beta} q^{\alpha-\beta} (1-q)} \tag{24}$$

$$\begin{cases} = 1-p & \text{if } \alpha = 0\,, \\ = 1-p+p/q & \text{if } \alpha > n\,, \\ \in (1-p, 1-p+p/q) & \text{otherwise.} \end{cases} \tag{25}$$

Therefore, the ratio is smaller than $1-p+p/q$, which proves (19), and its inverse is smaller than $1/(1-p)$, which proves (20). ∎

Lemma 3. *For any non-negative integers $\alpha, n, m \in \mathbb{Z}_+$,*

$$\frac{\Pr(G + B_{n+m} = \alpha)}{\Pr(G + B_n = \alpha)} \leq \left(\frac{p + q(1-p)}{q}\right)^m , \tag{26}$$

$$\frac{\Pr(G + B_n = \alpha)}{\Pr(G + B_{n+m} = \alpha)} \leq \left(\frac{1}{1-p}\right)^m . \tag{27}$$

Proof: We can expand the left ratio in (26) as

$$\frac{\Pr(G + B_{n+m} = \alpha)}{\Pr(G + B_n = \alpha)} = \prod_{k=1}^{m} \frac{\Pr(G + B_{n+k} = \alpha)}{\Pr(G + B_{n+k-1} = \alpha)} . \tag{28}$$

Then, by applying the bound in (19) to each of the m terms of this product, we reach (26). Likewise, we can prove (27) by applying (20) m times. ∎

2) Differential Privacy for Documents: We derive the ϵ guarantee that OSSE provides according to Definition 13. We use $\mathcal{M}$ to denote the randomized mechanism that takes the outsourced dataset $\mathcal{D}$ and a *single* query for keyword w and outputs the obfuscated access pattern for that w, i.e,. $\tilde{\Pi}_w$. Recall that $\tilde{\Pi}_w$ is a $n + |h|$-sized random vector characterized by (7) and (8).

Let $\mathcal{D}$ and $\mathcal{D}'$ be two adjacent datasets that are identical except for their kth document ($\mathcal{D}[k]$ and $\mathcal{D}'[k]$, respectively). These documents differ in a single keyword. Let w_* be the keyword that is only in $\mathcal{D}[k]$, and w'_* the keyword that is only in $\mathcal{D}'[k]$. Let $\tilde{\Pi}_w$ be the random output of $\mathcal{M}(\mathcal{D}, w)$, and $\tilde{\Pi}'_w$ be the random output of $\mathcal{M}(\mathcal{D}', w)$. Let π be a particular observed obfuscated access pattern. Then, we want to find an upper bound for the ratio

$$\frac{\Pr(\mathcal{M}(\mathcal{D},w)=\pi)}{\Pr(\mathcal{M}(\mathcal{D}',w)=\pi)} = \frac{\Pr(\tilde{\Pi}_w=\pi)}{\Pr(\tilde{\Pi}'_w=\pi)} = \prod_{i=1}^{n+|h|} \frac{\Pr(\tilde{\Pi}_w[i]=\pi[i])}{\Pr(\tilde{\Pi}'_w[i]=\pi[i])} \tag{29}$$

If $w \notin \{w_*, w'_*\}$, then this ratio is equal to one. Now consider the case where $w = w_*$. In that case, the distribution of $\tilde{\Pi}_w[i]$ and $\tilde{\Pi}'_w[i]$ is the same, except for $i = k$ (since w is in $\mathcal{D}[k]$ but not in $\mathcal{D}'[k]$) and $i = l_k$, where $l_k = h(k)$ is the label of the kth document. Therefore, we have

$$\frac{\Pr(\mathcal{M}(\mathcal{D},w_*)=\pi)}{\Pr(\mathcal{M}(\mathcal{D}',w_*)=\pi)} = \frac{\Pr(\tilde{\Pi}_{w_*}[k]=\pi[k])}{\Pr(\tilde{\Pi}'_{w_*}[k]=\pi[k])} \cdot \frac{\Pr(\tilde{\Pi}_{w_*}[l_k]=\pi[l_k])}{\Pr(\tilde{\Pi}'_{w_*}[l_k]=\pi[l_k])} \tag{30}$$

Since $\tilde{\Pi}_{w_*}[k] \sim \textbf{Bern}(p) + \textbf{Geo}(1-q)$ and $\tilde{\Pi}'_{w_*}[k] \sim \textbf{Geo}(1-q)$, we can bound the left coefficient using (17). Likewise, since $\tilde{\Pi}_{w_*}[l_k] \sim \textbf{Bi}(g,p) + \textbf{Geo}(1-q)$ and $\tilde{\Pi}'_{w_*}[l_k] \sim \textbf{Bi}(g+1,p) + \textbf{Geo}(1-q)$ for some positive integer constant g, we can bound the right coefficient using (20).

Therefore, using (17) and (20) we obtain

$$\frac{\Pr(\mathcal{M}(\mathcal{D}, w_*) = \pi)}{\Pr(\mathcal{M}(\mathcal{D}', w_*) = \pi)} \leq \frac{p + (1-p)q}{q} \cdot \frac{1}{1-p} = 1 + \frac{p}{q(1-p)} . \tag{31}$$

We can follow the same procedure to prove the same bound when the client queries for w'_*.

Finally, if we consider a sequence of t queries, in the worst case all of them are for either w_* or w'_*, so the differential privacy guarantee according to Definition 13 is

$$\epsilon = \ln\left(1 + \frac{p}{q(1-p)}\right) . \tag{32}$$

Using $\texttt{TPR} = p + q(1-p)$ and $\texttt{FPR} = q$, we obtain the value in Theorem 3.

3) Differential Privacy for Keywords: Consider a database $\mathcal{D}$, and a pair of neighbouring keyword lists $\vec{w}, \vec{w}' \in \Delta^{|\vec{w}|}$ that are identical except for their kth element, that is w in $\vec{w}$ and w' in $\vec{w}'$. Let $\tilde{\Pi}_w$ be the output of $\mathcal{M}(\mathcal{D}, w)$ and $\tilde{\Pi}_{w'}$ be the output of $\mathcal{M}(\mathcal{D}, w')$. Then, we want to find an upper bound for

$$\frac{\Pr(\mathcal{M}(\mathcal{D},w)=\pi)}{\Pr(\mathcal{M}(\mathcal{D},w')=\pi)} = \frac{\Pr(\tilde{\Pi}_w=\pi)}{\Pr(\tilde{\Pi}_{w'}=\pi)} = \prod_{i=1}^{n+|h|} \frac{\Pr(\tilde{\Pi}_w[i]=\pi[i])}{\Pr(\tilde{\Pi}_{w'}[i]=\pi[i])} . \tag{33}$$

For $i \in [n]$, the distribution of $\tilde{\Pi}_w[i]$ and $\tilde{\Pi}_{w'}[i]$ will be different if $\mathcal{D}[i]$ contains only one of $\{w, w'\}$. Let $D_{0,1}$ be the indices of documents that do not contain w but contain w'. For $i \in D_{0,1}$, $\tilde{\Pi}_w[i] \sim \textbf{Geo}(1-q)$ and $\tilde{\Pi}_{w'}[i] \sim \textbf{Bern}(p) + \textbf{Geo}(1-q)$. The ratio between the probabilities of these distributions can be bounded using (18). Likewise, define $D_{1,0}$ (indices of documents that contain w but do not contain w'), $D_{1,1}$ (contain both), and $D_{0,0}$ (contain neither). Using (17) and (18) we can get:

$$\frac{\Pr(\tilde{\Pi}_w[i] = \pi[i])}{\Pr(\tilde{\Pi}_{w'}[i] = \pi[i])} \leq \begin{cases} 1 & \text{if } i \in D_{0,0} \\ \frac{1}{1-p} & \text{if } i \in D_{0,1} \\ \frac{p+q(1-p)}{p} & \text{if } i \in D_{1,0} \\ 1 & \text{if } i \in D_{1,1} \end{cases} \tag{34}$$

Therefore,

$$\prod_{i=1}^{n} \frac{\Pr(\tilde{\Pi}_w[i]=\pi[i])}{\Pr(\tilde{\Pi}_{w'}[i]=\pi[i])} \leq \left(\frac{1}{1-p}\right)^{|D_{0,1}|} \left(\frac{p+q(1-p)}{p}\right)^{|D_{1,0}|} . \tag{35}$$

Now, we focus on the variables $\tilde{\Pi}_w[i+n]$ for $i \in [|h|]$. Let $D^i_{0,1}$ be the indices of the documents in $D_{0,1}$ whose label is i (same for the other sets $D_{0,0}$, $D_{1,0}$, and $D_{1,1}$, for $i \in [|h|]$). Then, we can write

$$\tilde{\Pi}_w[i+n] \sim \textbf{Bi}(g_{i+n} + |D^i_{0,1}|, p) + \textbf{Geo}(1-q)\,; \tag{36}$$

$$\tilde{\Pi}_{w'}[i+n] \sim \textbf{Bi}(g_{i+n} + |D^i_{1,0}|, p) + \textbf{Geo}(1-q)\,, \tag{37}$$

for some constants $g_{1+n}, g_{2+n}, \ldots, g_{|h|+n}$. If $|D^i_{0,1}| < |D^i_{1,0}|$, we can define a different set of constants $g'_{i+n} \doteq g_{i+n} + |D^i_{0,1}|$

and write

$$\tilde{\Pi}_w[i+n] \sim \mathbf{Bi}(g'_{i+n}, p) + \mathbf{Geo}(1-q)\,; \tag{38}$$

$$\tilde{\Pi}_{w'}[i+n] \sim \mathbf{Bi}(g'_{i+n} + |D^i_{1,0}| - |D^i_{0,1}|, p) + \mathbf{Geo}(1-q)\,, \tag{39}$$

Using (27), we can write

$$\frac{\Pr(\tilde{\Pi}_w[i+n] = \pi[i+n])}{\Pr(\tilde{\Pi}_{w'}[i+n] = \pi[i+n])} \leq \left(\frac{1}{1-p}\right)^{|D^i_{1,0}|-|D^i_{0,1}|} \tag{40}$$

On the contrary, if $|D^i_{0,1}| > |D^i_{1,0}|$, using (26) we get

$$\frac{\Pr(\tilde{\Pi}_w[i+n] = \pi[i+n])}{\Pr(\tilde{\Pi}_{w'}[i+n] = \pi[i+n])} \leq \left(\frac{p+q(1-p)}{q}\right)^{|D^i_{0,1}|-|D^i_{1,0}|} \tag{41}$$

Now, using the fact that $\sum_{i=1}^{|h|} |D^i_{0,1}| = |D_{0,1}|$ and $\sum_{i=1}^{|h|} |D^i_{1,0}| = |D_{1,0}|$, we write

$$\prod_{i=1}^{|h|} \frac{\Pr(\tilde{\Pi}_w[i+n] = \pi[i+n])}{\Pr(\tilde{\Pi}_{w'}[i+n] = \pi[i+n])} \leq \left(\tfrac{1}{1-p}\right)^{|D_{1,0}|} \left(\tfrac{p+q(1-p)}{q}\right)^{|D_{0,1}|}. \tag{42}$$

Here, we have used the fact that (40) and (41) are maximized when $D^i_{0,1}$ is empty when $D^i_{1,0}$ is not, and vice-versa. Finally, multiplying (35) and (42), we get

$$\frac{\Pr(\mathcal{M}(\mathcal{D}, w) = \pi)}{\Pr(\mathcal{M}(\mathcal{D}, w') = \pi)} = \leq \left(\frac{1}{1-p} \cdot \frac{p+q(1-p)}{q}\right)^{|D_{0,1}|+|D_{1,0}|}, \tag{43}$$

and therefore according to Definition 14,

$$\epsilon = \ln\left(1 + \frac{p}{q(1-p)}\right), \tag{44}$$

which implies Theorem 3.

Note that our ϵ bounds in (32) and (44) can be made smaller by allowing a small probability of failure δ and using the advanced composition rule [16] of differential privacy.

B. Analysis of $\mathtt{ctr}_{max}$ in OSSE.

This analysis is equivalent to the balls-and-bins problem with $F_{\mathtt{max}}$ balls and $F_{\mathtt{max}}$ bins. We assume $|h| = F_{\mathtt{max}}$ and that the hash function outputs values uniformly at random. We sat that the search index construction *succeeds* if we pick a $\mathtt{ctr}_{\mathtt{max}}$ value that is *strictly larger* (since the counter starts at 0) than the maximum number of documents that share a label and keyword in common. Then, we want to prove that, when $\mathtt{ctr}_{\mathtt{max}} = c \cdot \ln F_{\mathtt{max}} / \ln\ln F_{\mathtt{max}}$ (for some constant c that we will determine), the probability of success is overwhelming (larger than $1 - 1/n$).

Let S denote the success event, and let $n_{i,j}$ be the number of documents that have keyword $w_{(i)}$ and label j ($i \in [|\Delta|], j \in [F_{\mathtt{max}}]$). Then,

$$\Pr(S) = \Pr\left(\cap_{i=1}^{|\Lambda|} \cap_{j=1}^{F_{\mathtt{max}}} \{n_{i,j} < \mathtt{ctr}_{\mathtt{max}}\}\right) \tag{45}$$

$$= 1 - \Pr\left(\cup_{i=1}^{|\Lambda|} \{\max_j n_{i,j} \geq \mathtt{ctr}_{\mathtt{max}}\}\right) \tag{46}$$

$$\geq 1 - \sum_{i=1}^{|\Lambda|} \cdot \Pr\left(\max_j n_{i,j} \geq \mathtt{ctr}_{\mathtt{max}}\right). \tag{47}$$

Now, we bound $\Pr(\max_j n_{i,j} \geq \mathtt{ctr}_{\mathtt{max}})$. Note that, for a keyword $w_{(i)}$, the maximum number of documents that have that keyword is $F_{\mathtt{max}}$.

$$\Pr(\max_j n_{i,j} \geq \mathtt{ctr}_{\mathtt{max}}) \leq \binom{F_{\mathtt{max}}}{\mathtt{ctr}_{\mathtt{max}}} \left(\frac{1}{F_{\mathtt{max}}}\right)^{\mathtt{ctr}_{\mathtt{max}}} \tag{48}$$

$$\leq \left(\frac{e}{\mathtt{ctr}_{\mathtt{max}}}\right)^{\mathtt{ctr}_{\mathtt{max}}}. \tag{49}$$

Now, using that $\mathtt{ctr}_{\mathtt{max}} = c \cdot \ln F_{\mathtt{max}} / \ln\ln F_{\mathtt{max}}$,

$$\begin{aligned}
&\Pr(\max_j n_{i,j} \geq \mathtt{ctr}_{\mathtt{max}}) \\
&\leq \left(\frac{e}{\mathtt{ctr}_{\mathtt{max}}}\right)^{\mathtt{ctr}_{\mathtt{max}}} \\
&= \exp\left(\frac{c \cdot \ln F_{\mathtt{max}}}{\ln\ln F_{\mathtt{max}}} \cdot \ln \frac{e \ln\ln F_{\mathtt{max}}}{c \ln F_{\mathtt{max}}}\right) \\
&\overset{(a)}{\leq} \exp\left(\frac{c \cdot \ln F_{\mathtt{max}}}{\ln\ln F_{\mathtt{max}}} \cdot (\ln\ln\ln F_{\mathtt{max}} - \ln\ln F_{\mathtt{max}})\right) \\
&= \exp\left(-c \ln F_{\mathtt{max}} + \frac{c \ln F_{\mathtt{max}} \cdot \ln\ln\ln F_{\mathtt{max}}}{\ln\ln F_{\mathtt{max}}}\right) \\
&\leq \exp(-(c-1)\ln F_{\mathtt{max}}) = \frac{1}{F^{c-1}_{\mathtt{max}}},
\end{aligned}$$

where (a) holds when $c \geq e$. Then,

$$\Pr(S) \geq 1 - |\Lambda| \cdot \Pr(\max_j n_{i,j} \geq \mathtt{ctr}_{\mathtt{max}}) \geq 1 - \frac{|\Lambda|}{F^{c-1}_{\mathtt{max}}}. \tag{50}$$

This probability is larger than $1 - 1/n$ when

$$c \geq \frac{\ln|\Delta| + \ln n}{\ln F_{\mathtt{max}}} + 1\,. \tag{51}$$

Since $n \gg F_{\mathtt{max}}$ and typically $|\Delta| \geq F_{\mathtt{max}}$, then the assumption that $c \geq e$ in (a) above holds.

This implies that we need

$$\mathtt{ctr}_{\mathtt{max}} = \left(\frac{\ln|\Delta| + \ln n}{\ln F_{\mathtt{max}}} + 1\right) \cdot \frac{\ln F_{\mathtt{max}}}{\ln\ln F_{\mathtt{max}}} \tag{52}$$

$$= \frac{\ln|\Delta| + \ln n + \ln F_{\mathtt{max}}}{\ln\ln F_{\mathtt{max}}} \leq \frac{3 \cdot \ln n}{\ln\ln F_{\mathtt{max}}} \tag{53}$$

C. Analysis of Expected Number of Documents with a Keyword w Under Different Distributions

We study $E_w \doteq \mathrm{E}\{\mathcal{D}(w)\}$, i.e., the expected number of documents that have a particular keyword w, for different keyword and query distributions.

The general expression for E_w is

$$E_w = \sum_{i=1}^{|\Delta|} \Pr(w_{(i)}) \cdot |\mathcal{D}(w)|\,, \tag{54}$$

where $\Pr(w_{(i)})$ is the probability that the client queries for keyword $w_{(i)}$.

1) Uniform Distribution: When all keywords have the same (maximum) frequency of appearance $|\mathcal{D}(w)| = F_{\mathtt{max}}$ then, regardless off $\Pr(w_{(i)})$, we have

$$E_w = \sum_{i=1}^{|\Delta|} \Pr(w_{(i)}) \cdot F_{\mathtt{max}} = F_{\mathtt{max}}\,. \tag{55}$$

2) Zipfian Distribution: Zipf's law states that the frequency of a individual word in a corpus of natural language utterances is inversely proportional to its rank (the position of it in a sorted list in decreasing order of frequency) [40]. Assume the keywords in Δ are sorted in descending frequency order. We know that $|\mathcal{D}(w_{(1)})| = F_{\texttt{max}}$. Then, if keyword frequency follows Zipf's law, we can write

$$|\mathcal{D}(w_{(i)})| = \frac{F_{\texttt{max}}}{i}\,. \tag{56}$$

When query frequencies also follow Zipf's law, we get

$$\Pr(w_{(i)}) = \frac{1}{i \cdot H_{|\Delta|}}\,, \quad \text{where } H_{|\Delta|} \doteq \sum_{i=1}^{|\Delta|} \frac{1}{i}\,. \tag{57}$$

Then,

$$E_w = \sum_{i=1}^{|\Delta|} \frac{1}{i \cdot H_{|\Delta|}} \cdot \frac{F_{\texttt{max}}}{i} \approx \frac{F_{\texttt{max}}}{\ln|\Delta| + \gamma} \cdot \frac{\pi^2}{6}\,, \tag{58}$$

where we have used that the harmonic number $H_n \approx \log n + \gamma$ where $\gamma \approx 0.58$ is the Euler-Mascheroni constant, and $\sum_{i=1}^{|\Delta|} i^{-2} \approx \pi^2/6$.

3) Worst-Case Distribution: OSSE's communication overhead is inversely proportional to E_w, so the worst-case distribution for OSSE is the one that minimizes E_w. Since each keyword is in at least one document, and the maximum keyword frequency is $F_{\texttt{max}}$, the worst-case will happen when there is a set of keywords with frequency one and those keywords are the only ones that the client queries, so $E_w \approx 1$.

D. Adapting co-occurrence attacks against OSSE and CLRZ.

In our evaluation, we consider three attacks that use co-occurrence and volume information, namely IKK [20], count attack [8], and graph matching [31]. These attacks compute the co-occurrence matrices M and M' from the observations and the auxiliary (training) information, respectively, and use these matrices to find an assignment of queries to keywords. We explain how to compute these matrices, and then we delve into specifics of each attack.

First, consider a case where no defense is applied. Let m be the number of *distinct* obfuscated access patterns observed by the adversary ($m \leq |\Delta|$), and let $\tilde{\Pi}_{(i)}$ be the ith distinct obfuscated access pattern observed ($i \in [m]$). Recall that n is the number of documents in the dataset. Then, M is an $m \times m$ matrix such that its i,jth entry contains the *probability* that a document is returned for both the ith and jth distinct queries. Mathematically,

$$M[i][j] = \langle \tilde{\Pi}_{(i)}, \tilde{\Pi}_{(j)} \rangle / n\,, \tag{59}$$

where $\langle \cdot, \cdot \rangle$ is the dot product.

Matrix M' is a $|\Delta| \times |\Delta|$ matrix whose i,jth entry contains the probability that a document contains both $w_{(i)}$ and $w_{(j)}$, estimated using the training set. Let $\Pi_{(i)}$ be the true access pattern of keyword $w_{(i)}$ in the training data and let n' be the total number of documents in the training data. Then,

$$M'[i][j] = \langle \Pi_{(i)}, \Pi_{(j)} \rangle / n'\,. \tag{60}$$

In order to adapt the attacks against CLRZ and OSSE, we modify the computation of M' by taking the true positive and false positive rates (TPR and FPR) into account. A document is returned as response for queries $w_{(i)}$ and $w_{(j)}$ with probability $\texttt{TPR}^2$ when that document contains both keywords, with probability $\texttt{TPR} \cdot \texttt{FPR}$ when it only contains one of the keywords, and with probability $\texttt{FPR}^2$ when it contains neither. Mathematically, let $n_{i,j} \doteq \langle \Pi_{(i)}, \Pi_{(j)} \rangle / n'$ be the normalized number of documents that have both $w_{(i)}$ and $w_{(j)}$, and let $\bar{n}_{i,j} \doteq \langle 1 - \Pi_{(i)}, 1 - \Pi_{(j)} \rangle / n'$ be the normalized number of documents that do not have $w_{(i)}$ nor $w_{(j)}$. Then, we update the computation of M' as follows:

$$M'_{i,j} = \begin{cases} i \neq j: & \texttt{TPR}^2 \cdot n_{i,j} + \texttt{FPR}^2 \cdot \bar{n}_{i,j} \\ & +\texttt{TPR} \cdot \texttt{FPR} \cdot (1 - n_{i,j} - \bar{n}_{i,j})\,, \\ i = j: & \texttt{TPR} \cdot n_{i,i} + \texttt{FPR} \cdot \bar{n}_{i,i}\,. \end{cases} \tag{61}$$

All the attacks we evaluate, except for the frequency attack [27], use these matrices. We explain particularities of each attack below.

*1) **Adapting IKK**:* The naive version of the attack, that we simply call IKK in our experiments, uses M and M' as in (59) and (60), respectively. Against CLRZ, we use the original implementation [20] where each distinct access pattern is assigned to a unique distinct keyword. Against OSSE, however, the same keyword can generate different access patterns. Therefore, in that case we modify the heuristic IKK annealing algorithm so that it allows assigning the same keyword to different observed access patterns.

The improved version of the attack, that we call IKK* in our experiments, uses M' from (61).

*2) **Adapting the count attack**:* The count attack [8] first builds a list of candidate keywords for each observed query based on the query response volume and background information about the dataset (in our experiments, we give the adversary the full plaintext database). Then, the count attack rules out assignments using co-occurrence information until all of the queries are disambiguated and only one possible matching remains.

Cash et al. propose a generalization of the attack when the background information is imperfect. In this variation, they assume that the number of documents returned in response to a query follows a Binomial distribution parametrized by the background (training) information, and use confidence intervals derived from Hoeffding bounds to build keyword candidates and disambiguate queries. This version of the count attack applies naturally against CLRZ and OSSE, since adding false positives and false negatives to the access patterns actually causes the query volume and co-occurrences to follow a Binomial distribution.

The attack against CLRZ proceeds as follows:

1) Compute the co-occurrence matrices M and M' as in (59) and (61), respectively.
2) Compute the confidence intervals such that the observed volumes and co-occurences (in M) are within that interval (centered around the values in M') with probability $p = 0.95$ (c.f. [8], Theorem 4.1).

3) Build the set of candidate keywords for each observed query based on the volume information (diagonal elements of M and M'). If any candidate set is empty, the attack fails.
4) Disambiguate queries by following the brute-force approach by Cash et al. (c.f. [8], Section 4.4). We first select the set of the 10 most-frequently observed queries, build all the possible assignments of those queries to their candidate keywords, and try to disambiguate queries for each of those assignments using co-occurrence information. The attack returns the assignment that disambiguated the highest number of queries. If *all* of the assignments resulted in an inconsistency (the candidate set of a keyword became empty), the attack fails.

Running the count attack against OSSE is not straightforward, since the attack heuristics assume that each distinct query belongs to a different keyword. In OSSE, however, the same keyword will likely generate different access patterns every time it is queried. Therefore, we apply the clustering technique that we used for the frequency attack in Section IX-A: the adversary takes all the observed access patterns and clusters them into N_c groups, where N_c is the true number of distinct queries issued by the client (we give the adversary this ground-truth information so that its performance is a worst-case against OSSE). Then, the adversary takes the center of each cluster as the "representative" access pattern of that group (note that this is still consistent with (61)) and runs the attack as explained above. All the elements within a cluster are assigned the keyword of their representative access pattern.

*3) **Adapting the graph matching attack**:* The graph matching attack [31] uses keyword co-occurrence information, similar to IKK. We evaluate the adapted attack only, i.e., using M and M' from (59) and (61), respectively. Since this attack also assumes that each distinct access pattern is assigned to a different keyword, we use the clustering technique to group access patterns in OSSE and then run the graph matching algorithm using the cluster centers.